Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

Robot diagnostics based on monitoring its kinematic variables

by

Bc. Ondřej Novák

A diploma thesis submitted to
the Faculty of Electrical Engineering, Czech Technical University in Prague,
in partial fulfillment of the requirements for the "Engineer's degree"
abbreviated as "Ing."

Master's degree study programme: Cybernetics and Robotics
Specialization: Systems and control

Prague, December 2018

Supervisor:
Ing. Pavel Burget, Ph.D.
Department of Control Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2
160 00 Prague 6
Czech Republic

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Novák Ondřej** Personal ID number: **420390**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Systems and Control**

II. Master's thesis details

Master's thesis title in English:

Robot diagnostics based on monitoring its kinematic variables

Master's thesis title in Czech:

Diagnostika robota na základě sledování jeho pohybových veličin

Guidelines:

1. Prepare a workplace for operation data acquisition from a six-axis industrial robot on a PC, for which an existing PROFINET-based communication system will be used. Be sure to keep the existing communication relation of the robot to a superordinate controller, realized in a PLC.
2. Design and implement a mechanism of collecting the data from the robot with a period of several milliseconds, their local processing and transfer to a cloud environment.
3. Using available literature design methods for local processing of the data and their aggregation and compression in order for the data transfer to the cloud to be as effective as possible, while loosing as little information as possible.
4. Implement analytical functions for the cloud application, which allow for long-term monitoring of robot operations with respect to its future maintenance operations.

Bibliography / sources:

[1] Bruno Siciliano, Oussama Khatib (Eds.). Springer Handbook of Robotics. 2008. ISBN: 978-3-540-23957-4.
[2] M. Ron and P. Burget, 'Stochastic modelling and identification of industrial robots,' 2016 IEEE International Conference on Automation Science and Engineering (CASE), Fort Worth, TX, 2016, pp. 342-347. doi: 10.1109/COASE.2016.7743426

Name and workplace of master's thesis supervisor:

Ing. Pavel Burget, Ph.D., Testbed, CIIRC

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **16.01.2018** Deadline for master's thesis submission: **08.01.2019**

Assignment valid until: **30.09.2019**

Ing. Pavel Burget, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Declaration

I declare that I elaborated this thesis on my own and that I mentioned all the information sources and literature that have been used in accordance with the Guideline for adhering to ethical principles in the course of elaborating an academic final thesis.

In Prague, ...

Bc. Ondřej Novák

Abstract

This diploma thesis focuses on the development of a framework for analyzing data from industrial robotic manipulators. Data are gathered via an industrial network by a PC, where they are pre-processed. The pre-processing is done to reduce the data volume. After the reduction, data are sent to a cloud where the data are classified and a robotic operation is assigned to the measurement. Machine learning methods are applied to identify performed operations.

A modular framework for processing data from industrial devices is introduced. The framework supports PROFINET industrial network and Azure Cloud, but it is designed to be easily extended to support different communication protocols and cloud platforms.

Keywords:

Industrial robot diagnostics, edge computing, cloud computing, operation identification, real-time data analysis

Abstrakt

Tato diplomová práce se zabývá vývojem frameworku pro analýzu data z průmyslových robotických manipulátorů. Data jsou sbírána pomocí průmyslové sítě počítačem, kterým jsou následně předzpracována. Předzpracování je provedeno kvůli snížení objemu dat. Po snížení objemu jsou data poslána do cloudového prostředí, ve kterém jsou z nich klasifikovány robotické operace. Pro klasifikaci jsou použity metody strojového učení.

Pro zpracování dat je představen modulární framework podporující průmyslovou síť PROFINET a Azure Cloud. Framework je navržen tak, aby byl snadno rozšiřitelný o jiné komunikační protokoly a cloudové platformy.

Klíčová slova:

Diagnostika průmyslových robotů, výpočty na edge, výpočty v cloudu, identifikace operací, analýza dat v reálném čase

Acknowledgements

I would like to express gratitude to my supervisor Ing. Pavel Burget, Ph.D. for providing encouragement as well as valuable advice and insights. I would also like to thank Ing. Martin Ron for several consultations regarding the machine learning part of the thesis. Last but not least I want to also thank all my colleagues from the laboratory for making the mood so nice to work in.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this chapter, the problem definition and the motivation are stated, as well as the goals and thesis structure.

1.1 Motivation

Importance of industrial robots has been growing worldwide in recent years mainly because of a lack of qualified workers [1], the need for higher efficiency and convenient cost of mass production. The trend of bringing robotization, automation and IT to industry that begun to grow after the so-called third industrial revolution at the end of the 20th century ceases to be enough. More and more focus is given on smart systems, to catch up with companies' needs. This trend is called the Fourth Industrial Revolution, or Industry 4.0 [2, 3].

One of the main effects of the trend is the interconnection between the industrial systems factory-wide - e.g., IoT connected manufacturing systems - and even inter-factory-wide - e.g., a cloud platform for managing the distribution of resources between different companies. Along with the interconnection trend, modern methods from the IT world, such as machine learning and automatic planning, are to be adopted by the manufacturing process.

This thesis focuses on applying some of these concepts for diagnostics of the industrial robots. Although all the experiments have been carried out in the laboratory environment, the emphasis has been put on using industrial hardware and standards.

Methods for solving problems like increasing the efficiency of the manufacturing process and predictive maintenance are more and more relevant because every second that is saved in the process lowers the manufacturing cost and increases the throughput that yields in higher earnings. Similarly, every minute that the production line is inoperable usually causes a significant financial loss. Moreover, an anomalous behavior of the robot can lead to an inferior quality of the product or damage to the product, the robot or its surroundings.

Nowadays, robotic systems are often programmed to do simple repetitive tasks without any advanced intelligence. However, as flexible manufacturing [4, 5] becomes more ap-

pealing for the companies, the proportion of robots doing tasks depending on the current factory goal, may increase.

1.2 Problem Statement

The robotic program can be viewed as a state machine that repeatedly runs operations as a reaction to events. An independent feedback system is usually necessary to detect an anomalous behavior not only at the level of measured variables, but also at the level of robotic operations.

One of the feasible ways to model the behavior is to use machine learning techniques, that are in recent years widely used in the IT world. In order not to store and process all the data locally, part of the processing pipeline should be located off-site, in a cloud environment.

To collect and process data from various industrial devices, a modular framework is needed. The modularity allows to implement support for different communication protocols as well as for various pre-processing techniques. To avoid problems with integration of libraries for industrial communication, which are often complicated and implemented for out-of-date systems and language specifications, a simple framework, that is easy to debug and modify is needed.

1.3 Goals

The goal of this thesis is to implement a chain (also called pipeline) of programs that would gather data from an industrial robot, pre-process the data on-premise (i.e. on a computer located in a laboratory or a factory, also called Edge PC) and then do the final processing in a cloud. In Figure 1.1 a proposed solution is depicted. Keep in mind, that the gray parts are not intended to be implemented in this thesis, but it should be possible to add them later. This framework should be modular, so that it is possible to use it for different industrial processes without changing the core functionality.

On the Edge PC, a program that will reduce the data volume will run. The reduced data are next going to be sent to a cloud to be processed and evaluated by a model. The model should classify the operation, that has been carried out by the robot from the measured data.

1.4 Contribution

The key contribution of the work is the design, implementation and specific application of a modular framework for processing data from industrial devices. Due to the modular concept, it is easily extensible, and support for various communication protocols can be added without modifying existing code. The modularity also allows to implement various pre-processing methods in the form of separate modules, that can be interchanged and

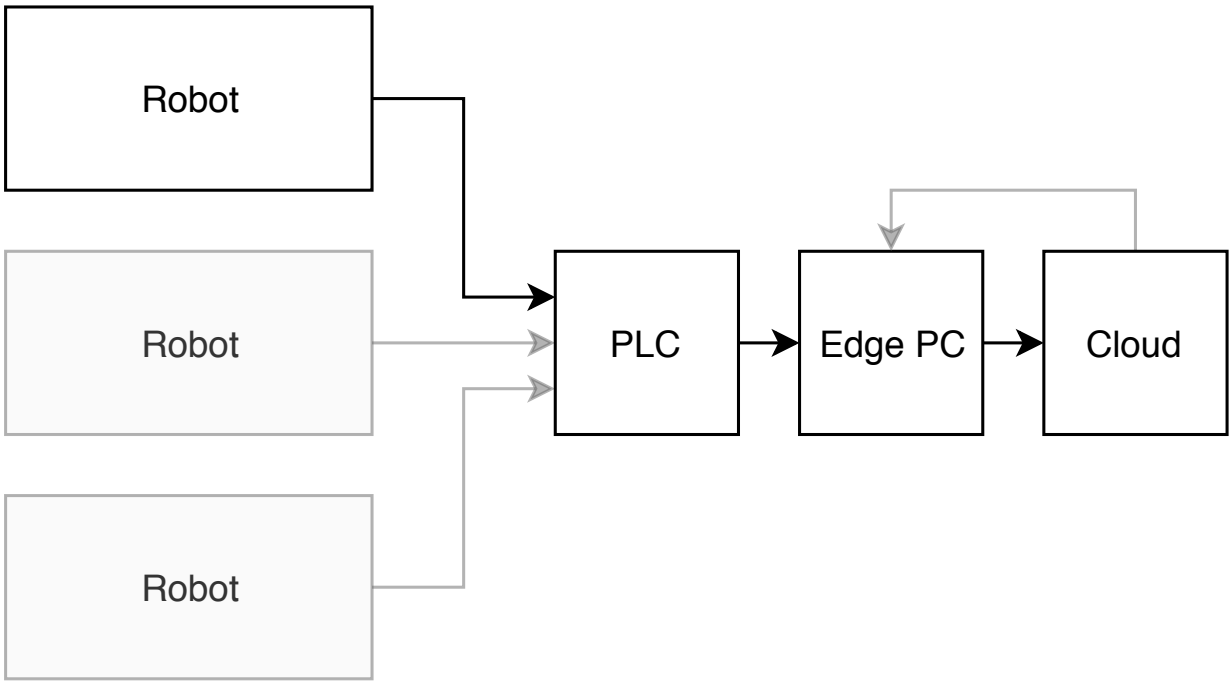


Figure 1.1: A simple overview of the desired architecture.

reused, which may prove useful for research purposes. The framework also connects to a cloud, where the data may be further processed and stored. Similarly to the communication interface extensibility, support for different cloud platform can be added.

For the framework, an application was developed. It gathers data from industrial robots via PROFINET, pre-processes it and sends it to the Azure Cloud. The pre-processing is done to reduce the volume of the data. In the cloud, the operation done by the robot is identified from the measured data.

1.5 Structure of the Thesis

The thesis is organized into five chapters as follows:

1. *Technologies and Methods*: Used technologies such as Microsoft Azure Cloud, and various machine learning techniques important for the thesis are discussed.

2. *Modular Framework and Industrial Components*: Describes the architecture and implementation of the robotic, pre-processing and cloud parts irrespective to the data processed.

3. *Data and Model Evaluation*: Deals with the description of the machine learning model used for the classification.

4. *Conclusion*: Summarizes the outcomes of this thesis and suggests next steps to carry out in the following phases of the project.

Chapter 2

Technologies and Methods

KUKA robots, Cloud and statistical methods used throughout the project are relatively complicated and feature rich. It is out of scope of the thesis to describe all the details here, hence only crucial parts are described and relevant references are provided.

2.1 Testbed for Industry 4.0

Located at CIIRC CTU in Prague, Testbed for Industry 4.0, abbreviated just as Testbed, is a laboratory founded in the summer of 2017. Its purpose is to show new technologies to the Czech industry and to provide a shared infrastructure for research focused on Industry 4.0.

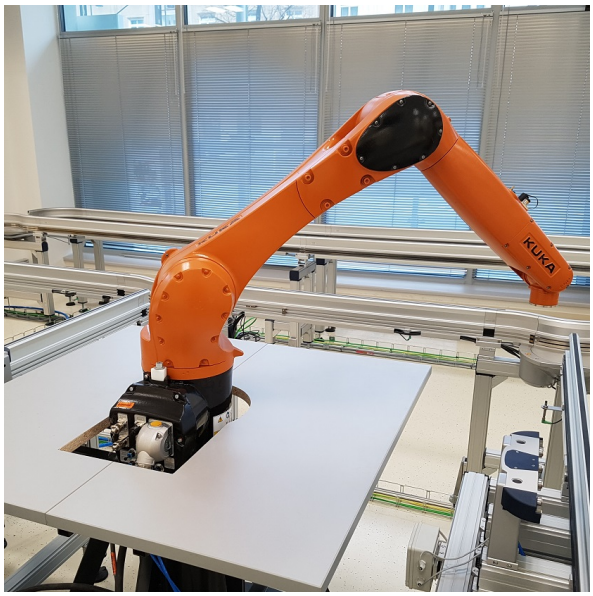
The laboratory consists of multiple workplaces and showcases such as a virtual reality workplace or a workplace with a collaborative robot. The lab also contains a flexible production line consisting of a flexible monorail based conveyor system 2.1c operated by shuttles and several robots. The conveyor system produced is by Montratec company, the robots by the KUKA company. One of the robots is KUKA LBR iiwa (Figure 2.1b), three others are KUKA Agilus KR 10 1100 sixx (Figure 2.1a). The production line serves as a basis for all the development and experiments performed within this thesis.

2.2 KUKA KR

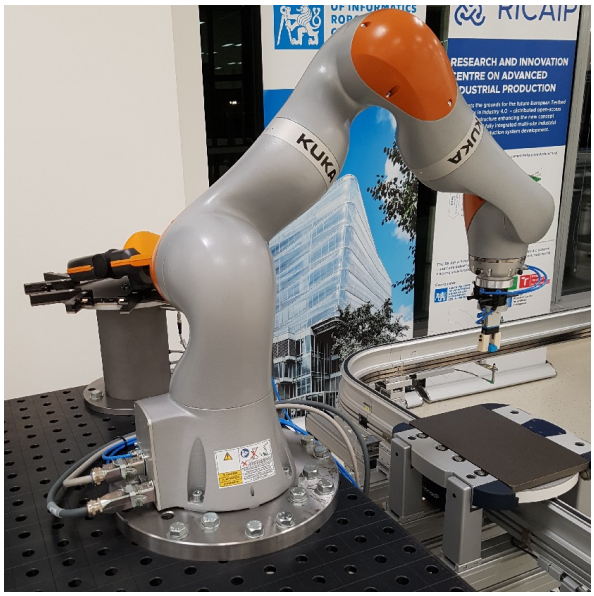
The abbreviation KR is a general denotation of a series of industrial robots by KUKA company. The series is further divided into a model series' as KR Quantec, KR Titan, KR 60, KR Agilus and many more.

2.2.1 KUKA KRC

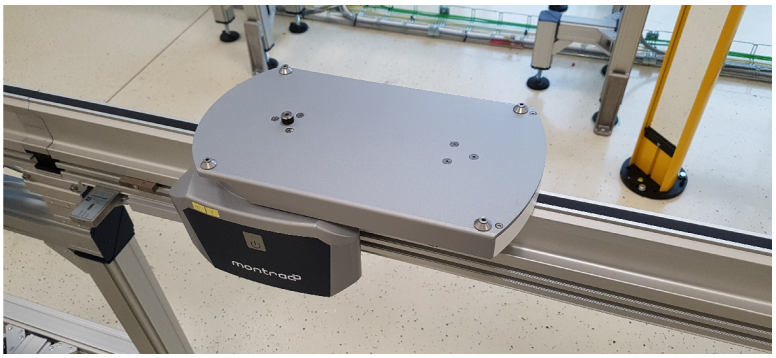
The KR robots require a controller to operate. Usually, the controller, called KR C, is a specialized PC enclosed in an industrial cabinet that is running a control program on top of



(a) KUKA Agilus KR 10 R1100 sixx



(b) KUKA LBR iiwa 14 R820



(c) Montrac monorail with a shuttle

Figure 2.1: Some of the Testbed equipment

Note 1 - Highlight on the term Industry 4.0

Industry 4.0 is a broadly used term that has no strict definition and therefore, every company, team or even person may have their own meaning for the term. A concept description from [3] will be explained here.
To summarize: Industry 4.0 is about bringing the technologies from the IT world, such as the Internet, unified means of communication and other IT principles that caused the IT revolution. Another aspect is to prepare production facilities for the new type of demand - low volume manufacturing and tailored orders - via flexible manufacturing. A plant following such a concept might use 3D printers and mills to manufacture and robots with generic programs to automatically fulfill customers needs.

an operating system. The controller has its communication interfaces that can be used to communicate with other devices such as a superior control system or robot's peripherals. In the cabinet, there is also a power source and power electronics to control the robot drives [6].

The controller is running KSS operating system, that allows to calibrate the tool that the robot wields, to set the communication parameters and edit the robotic programs.

Aside from the setup and the main robotic program, there is `Submit interpreter` (or SPS) functionality in the control software [7, p. 453]. It is a built-in program loop that is run every robot cycle, which may be used to write a simple cyclic program. The interpreter shares resources with the main robotic program with a lower priority so its run time may be irregular [7, p. 453].

2.2.2 KUKA WorkVisual

A PC software used to configure and program KUKA KR robots is called `WorkVisual` [8]. Although some tasks, such as writing the robotic programs can also be done on the `SmartPAD`, there are specific tasks, where it has to be used. The pad, for example, does not allow configuring all parameters such as `PROFINET` communication parameters or `PROFINET` variable mapping.

2.2.3 KUKA KRL

The control system of KUKA KR C4 has its programming language, KRL, that is described in [7]. Using the language, a programmer may write standard declarative programs. It also allows interacting with robot peripherals using various industrial communication buses, for example, `ProPROFINETfiNET` IO or `EtherCAT`.

In this project, timers and variable mapping were used beside standard statements, such as if/else constructs or cycles, whose description and explanation is out of the scope of this thesis and can be found, for example, in [9]. Timers and variable mapping are

described in the following subsections and their use for this project is further described in section 3.2.

2.2.3.1 Timers

A feature used to measure a specific time interval [9]. A Timer can be set to specific negative time interval by setting variable `$TIMER[t]` (in milliseconds), where `t` is the timer number, and started by setting variable `$TIMER_START[t]`. The `$TIMER[t]` variable then counts up the time elapsed and after reaching zero, variable `$FLAG[t]` is set.

2.2.3.2 Variable Mapping

When a set of bits is used as a communication bus input or output, it is often convenient to map these bits to one variable, for example, 32-bit Integer, that can be then used more easily in the program.

A set of bits can be merged to a 32-bit output Integer using the following line of code

```
GLOBAL SIGNAL <VARIABLE_NAME> $OUT[<FIRST_BIT>] TO
   $OUT[<LAST_BIT>]
```

2.3 PROFINET

As the way of communication between the industrial components a `PROFINET` network was chosen as it has been already used by a significant part of the laboratory equipment. `PROFINET` protocol is an Industrial Ethernet-based network that is capable of both real-time and non-real-time communication using both synchronous and asynchronous transfers [10, 11].

There are multiple technologies built on top of `PROFINET`, as is `PROFIsafe`, a protocol for safety-related data transfer. Nevertheless, only the `PROFINET IO` communication was adjusted within the project, therefore only this technology will be described here.

2.3.1 PROFINET IO

From the programmer's point of view, there are three ways of the data transfer possible - cyclic real-time data exchange, acyclic real-time data transfer and TCP/IP.

With the cyclic data exchange, the IO Controller has an image of its IO Devices, i.e. inputs and outputs that are periodically refreshed. The data exchange cycle itself is controlled by the IO Controller and transferred data are mapped to the user address space of the IO Controller, which is available to the user application.

2.3.2 Shared Device

A PROFINET IO device consists of modules, such as Digital Input module or Analog Output module. Each of these modules can be assigned and controlled by a different IO Controller, whereas the maximum number of IO Controllers is limited by the device's capabilities. Such a Device is called a Shared Device [12]. This can be used to save cabinet space, save costs as there is no need to buy multiple communication interfaces. It can also be used to split a Device between different Controllers when there is a need to change the logical structure of a project without a need to do hardware changes.

2.3.3 I-Device

An I-Device is an IO Controller that has additional "Intelligent IO Device" functionality [13]. The functionality enables the IO Controller to be also an IO Device. So, for example, a PLC can control its peripherals, but it can also be controlled by a higher-tier PLC at the same time. Another example is a KUKA KR robot, that can be connected to a higher-tier control system as a PROFINET IO Device, but at the same time, it can be an IO Controller for its own peripherals [14] (e.g., pneumatic valves controlling work table clamps).

2.4 Microsoft Azure

Microsoft Azure is a cloud platform developed by Microsoft Corporation offering Infrastructure as a Service, for example, virtual servers, and Platform as a Service as different data storages (SQL and NoSQL databases, Blob storages, ...), IoT services, data analysis, and visualization services and more. As the Azure offers dozens of services and only a few were actually used within the project, the description below is limited to these.

As Microsoft provides free online documentation [15] along with tutorials and example code, the information in this section has been obtained from that information source if not stated otherwise.

2.4.1 IoT Hub

Azure IoT Hub is a gateway to the cloud environment for IoT Devices. It serves as an access point for sending telemetry from the devices to the cloud as well as sending a configuration to the devices from the cloud. Various protocols, such as HTTP, MQTT, AMQPS can be used to transfer the messages, while JSON is used for the payload that is transfered [16]. The access point runs on the Azure, but can also be deployed to the Edge device directly, so it can serve as a local gateway for devices that cannot run Azure stack directly. It can also route the messages to other Azure services, that are locally reduced to, for example, reduce network traffic.

IoT Hub allows message routing based on a query that is done on the message received. If a message complies to a rule, it is forwarded to one of the specified endpoints such as Azure Blob Storage or Azure Event Hubs.

2.4.2 Event Hubs

Azure Event Hubs is a scalable service for streaming event requests between various Azure services that is capable of processing millions of messages per second. The service can be connected to various other services such as Azure IoT Hub, Azure Functions or Apache Kafka.

A single Event Hubs service can host multiple Event Hub services while each of these can provide the messages received to multiple applications via consumer groups.

2.4.3 Azure Functions

Azure Functions is a serverless compute service that can process stateless requests from different sources. A Function can be programmed using a wide selection of programming languages, such as C#, F#, JavaScript, Python, Bash or Java. It is also possible to bind inputs and outputs of the functions to different data sources and data targets. Some of the supported bindings are IoT Hub, Events Hub, Time Trigger, HTTP Trigger, and Table Storage.

2.4.4 Azure ML Service

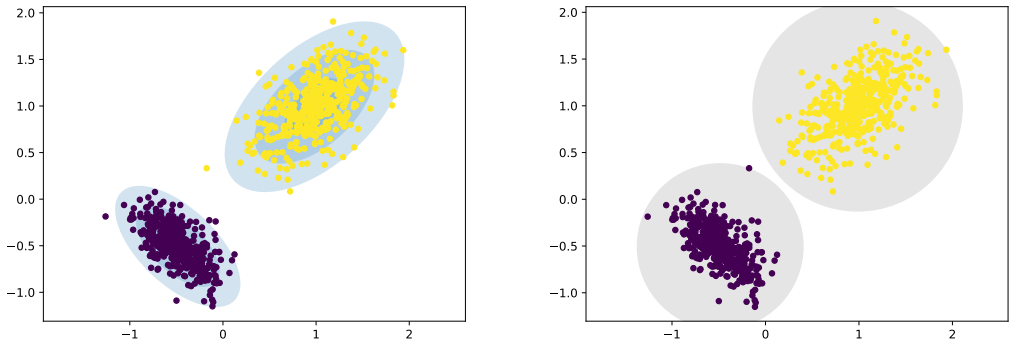
Azure Machine Learning Service is one of Machine Learning (ML) frameworks offered by Azure. Other frameworks are, for example, Azure Machine Learning Studio and Data Science Virtual Machine. These three frameworks should not be confused, as these are different services. Within the Service, Python API called Azure Machine Learning SDK for Python is included. This API can be used in a Python script or within Jupyter notebook.

The SDK is a set of Python libraries to be used by ML engineers to train, test and deploy ML models and can be used side by side with 3rd party or even custom Python packages. Therefore it is possible to use standard packages such as scikit-learn or numpy. Models can be programmed in a usual way using these or custom packages. In addition to the usual code, several calls has to be made within the program. First, it is necessary to initiate a session. Then it is possible to log model outputs, such as accuracy or plots, and save the serialized model. The serialized model can be then retrieved by another application. With the model created, it is possible to deploy these models as a web service.

2.4.5 Azure Table Storage

Azure Table Storage is one of many storage types supported by the Azure platform next to Azure Blob Storage, Cosmos DB or open-source services like MySQL or PostgreSQL. The Table Storage is a NoSQL storage that can store structured data while it can be easily scaled. Azure Table Storage is a part of Azure Storage service that in addition to Tables enables to use Blob, Queue or File storage.

The storage also allows changing the structure of the table. It is only necessary to create the table; its structure is inherited automatically from the data inserted.



(a) EM (b) K-means

Figure 2.2: A comparison between EM and K-means algorithm outputs

2.5 Data analysis algorithms and statistical methods

To analyze the data, several well known algorithms and methods was together used to produce the final model. These methods are described in general in this section while its specific usage for this project will be described in Chapter 4.

2.5.1 K-means

K-means is a variant of Expectation-Maximization (EM) algorithm for Gaussian Mixed Models (GMM) that does so-called hard clustering. In this thesis is the algorithm used to locate clusters of data points representing measurements. Data points that are within one cluster are then likely to be representing the same operation.

EM for GMM algorithm is searching for a Gaussians' parameters - mean, covariance and weight $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and π_k called $\boldsymbol{\theta}$ - of the GMM in an iterative manner repeating two steps, expectation step and maximization step. The algorithm outputs parameters $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ from which it is possible to calculate a posterior probability that each point belongs to every cluster [17].

K-means simplifies the procedure by assuming $\boldsymbol{\Sigma}_k = \sigma^2 I_d$ and $\pi_k = 1/K$. It is then possible to calculate just the mean

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{p \in P_k} p, \tag{2.1}$$

where P_k is a set consisting of points belonging to cluster k. The algorithm repeatedly assigns all data instances to the nearest cluster center (mean) and then recalculates new centers to match the assigned data (eq. 2.1). Algorithm 1 shows pseudo-code [17, 18] of the modified variant called K-means. In Figure 2.2, comparison between the output of EM (Figure 2.2a) and K-means (Figure 2.2b) can be seen.

Algorithm 1: K-means pseudo-code

Data: number of clusters k, N data points $P_i \in P$
Result: centers (means μ_j) of the clusters
$X \leftarrow$ initial centers of the clusters $\{X_1, \ldots, X_k\}$;
while *algorithm has not converged* **do**
  for $p \in P$ **do**
    assign point p to its nearest center $X_j \in X$;
  end
  for $x \in X$ **do**
    $x \leftarrow$ mean points $P_i \in P$ assigned to cluster center x ;
  end
end

2.5.2 PCA

Reducing a dimension of, for example, ten-dimensional vector to two dimensions is useful for visualizing the data. We can easily plot 2D vector, but we are unable to plot 10D vector. Similarly, the reduced data take less computer memory to store. On the other hand, by reducing the data, we may lose some information that the data carries (if we do not only remove redundant data) and we usually also lose the physical meaning of the variables, as the velocities, currents, and pressures measured are converted to abstract latent variables.

Principal Component Analysis, PCA, is a method to reduce a dimension of a data set by projecting a data set onto a a space of the desired dimension [19].

In this project the method was used to visualize a data, so let's have its principle explained by an example of using it for such a purpose. Assume we have a data set, that we can view as a matrix $\mathcal{M}^{m\times n}$, of m data instances each consisting of n measured features. Lets also assume $n = 4$, that means we have 4 dimensional data. We would like to project these onto a plane to be able to display it on the computer screen.

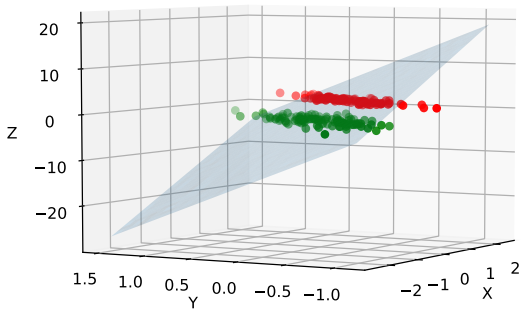
By projecting the points onto a n_r dimensional space (that is a plane in case of $n_r = 2$), we are able to reduce the dimension. PCA looks for the projection that gives minimum error when doing inverse projection back to the n dimensional space. The projection is found by minimizing

$$J(\boldsymbol{W}, \boldsymbol{Z}) = \frac{1}{N}\sum_{i=1}^{N} \|\tilde{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_i\|^2, \tag{2.2}$$

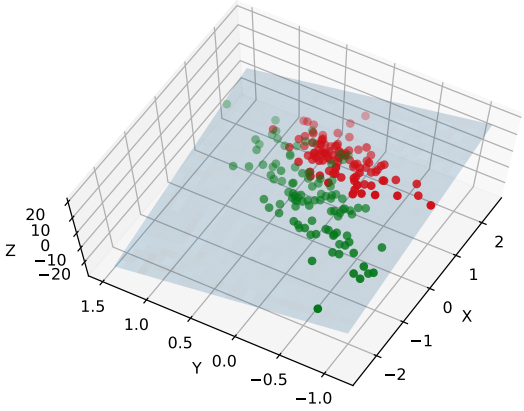
where $\hat{\boldsymbol{x}}_i = \boldsymbol{W}\boldsymbol{z}_i$ and $\boldsymbol{W}$ is orthonormal, $\boldsymbol{z}_i$ is a point from the latent space that we are projecting to and $\tilde{\boldsymbol{x}}_i$ is a point from the original space. $\tilde{\boldsymbol{x}}_i$ has to be transformed to zero-mean data by calculating mean $\boldsymbol{\mu}$ of the whole dataset and then subtracting it from the original data $\boldsymbol{x}_i$:

$$\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i - \boldsymbol{\mu}. \tag{2.3}$$

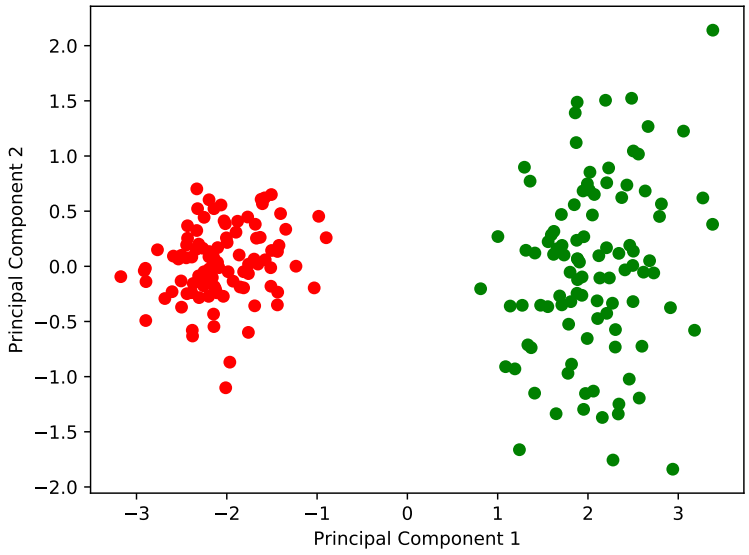
A visualization of transformation from a 3D space to 2D space is shown on Figure 2.3.



(a) 3D data set - point of view 1



(b) 3D data set - point of view 2



(c) Transformed 2D data set

Figure 2.3: Transformation of a 3D data set (visible from two different points of view) to 2D using PCA

2.5.3 Feature Agglomeration Based on Scikit-learn Library

Feature Agglomeration implemented in Python Scikit-learn library is another method used to reduce the number of columns for a data set (i.e. number of features) that uses an Agglomerative Clustering method on a set of features instead of clusters [20]. The features that are grouped are then merged into one number by some transformation, e.g., mean.

In the Agglomerative Clustering, with individual instances of data and with a similarity measure to compare any two instances of data, Euclidean or Cosine similarity can be used for example. This measure is then applied to all possible pairs of individual data instances while merging pairs with the highest similarities to groups of two. These groups of two data instances are to be compared to each other repeatedly until a desired level of merging is achieved [21].

To apply the measure on a pair of groups of data instances, which might be undefined, for example, we cannot measure the Euclidean distance between two groups of points G_1 and G_2, directly, a linkage is used. The similarity between all possible (x_1, x_2), $x_1 \in G_1$, $x_2 \in G_2$ is calculated and a linkage is applied. The linkage has multiple variants. For example, single-linkage, complete-linkage, average-linkage [21] or ward-linkage [20]. Single-, complete- and average-linkage takes the highest, lowest and average similarity of similarities between all the pairs, respectively. Ward-linkage minimizes the sum of squared differences within all groups [22].

2.5.4 K-folds

Cross-validation methods are used to verify the accuracy of a model independently on training data set, i.e. how well the model generalizes. One of these methods is K-folds [23]. The algorithm splits input data set into k distinct sets. From this data, one set is used as a validation set, while the other $k-1$ sets are used as training data. This process then repeats for all of k sets. Algorithm 2 depicts pseudo-code of the algorithm.

In each iteration, the accuracy of the model can be calculated to quantify, how well the model works on the given validation set. Stability of the accuracy or minimum accuracy level can be tested in each of the subsequent runs to verify the model.

Algorithm 2: K-folds pseudo-code

Data: number of splits k, N data points $P_i \in P$, model M to be validated
Result: accuracy of the model for each of the k runs
$P_{split} \leftarrow$ split the data set to k distinct sets ;
for $i \leftarrow 0$ **to** k **do**
    $X_{training} \leftarrow$ select all but i-th set of P_{split} ;
    $X_{validation} \leftarrow$ select i-th set of P_{split} ;
    $M_i \leftarrow$ train model M on the $X_{training}$ dataset ;
    $a_i \leftarrow$ calculate the accuracy of M_i on $X_{validation}$ data set ;
end

2.5.5 Mahalanobis Distance

To express, how distant a point from a distribution is, Mahalanobis distance can be used. It can be useful, for example, to verify if a point belongs to a cluster or to test for sufficient separation distance between clusters.

Let $\mathbf{\Sigma}$ and $\boldsymbol{\mu}$ be a covariance matrix and mean of a Normal distribution $D \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$. Then

$$d = \sqrt{(\boldsymbol{x}_i - \boldsymbol{\mu})\, \mathbf{\Sigma}^{-1}\, (\boldsymbol{x}_i - \boldsymbol{\mu})^T} \tag{2.4}$$

is a Mahalanobis distance of point $\boldsymbol{x}_i$ from distribution D [24]. Square of the Mahalanobis distance, d^2 follows χ^2 distribution [25], therefore it is possible to get a likelihood $\ell(\boldsymbol{\mu}, \mathbf{\Sigma} \mid \boldsymbol{x}_i) = p(\boldsymbol{x}_i \mid \boldsymbol{\mu}, \mathbf{\Sigma})$ that quantizes, how likely point $\boldsymbol{x}_i$ follows distribution D by calculating a value of cumulative distribution function of the χ^2_n distribution in point $\boldsymbol{x}_i$, where n is number of features (i.e. size of vector $\boldsymbol{x}_i$).

Chapter 3

Modular Framework and Industrial Components

The output of the project has been designed in a way that it is possible to use it in the industrial environment. Therefore standard industrial tools and hardware are to be used as much as possible. It might not be feasible to fulfill these needs in all parts of the project, as things as an AI or cloud computing are not much used in the industry, so there are not many standard solutions available. The solution should be flexible and easily extendable to other platforms – from hardware, edge computer and cloud point of view.

The project is divided into two main parts that are further divided into sub-parts. The first part, let's call it an Industrial Part, is supposed to be a set of programs and associated configurations that will be running on industrial hardware. Specifically, it is the program running on the robot itself and also the program running on a PLC. The second part that is called a Data Processing Part within the project is a set of programs running on a regular PC and within a cloud.

In this chapter, these components are ordered from the input of the system, that is the robot, to the output of the system, that is the cloud.

3.1 Project Structure

3.1.1 Industrial Part

First, it is necessary to gather the data in the robot so it can be sent to further processing. For that, an interface between industrial standard tools and an IT world has to be selected.

There are multiple ways to transfer data between industrial systems and personal computers. One of these is OPC UA, but this technology can introduce delays [26] that are significant for our sampling period that is in the order of units of milliseconds. For this kind of data, a fast real-time communication, e.g. EtherCAT or Profibus, is necessary instead. PROFINET IO has been chosen, because the current setup in the laboratory uses this type of communication.

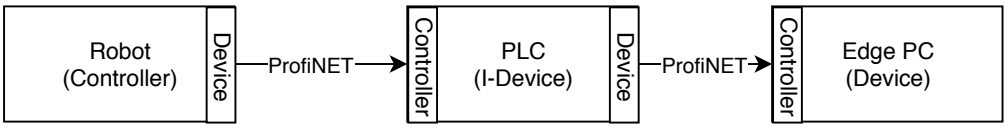


Figure 3.1: Data-flow diagram of the Industrial Part of the project.

For this project, robot KUKA Agilus KR10 R1100 sixx [27], was available. It is a 6 DOF industrial robot with 10 kg payload running KUKA OS.

Even though the robot supports both relevant roles of the PROFINET IO network, IO Device and IO Controller, there was not a feasible way to connect the robot to a PC directly. The IO Device (w.r.t. the data-gathering PC) method was ruled out because in the current setup the robot was controlled by a PLC while acting as an IO Device and as the robot does not support shared-device functionality [14], it cannot be bound to two different IO Controllers. To make the IO Device attitude work, the system structure would have had to be changed.

The second method where the robot would be in the IO Controller role w.r.t. data-gathering PC would require the PC to be IO Device. This attitude could also be promising, but there was not a necessary software (an IO Device stack for either MS Windows or GNU/Linux) available in the laboratory at the time of writing this thesis.

For these reasons, a workaround had to be made. The PLC that is acting as the IO Controller for the robot has to relay the data from the robot to the PC. The structure of the industrial part is depicted in Figure 3.1.

3.1.2 Data Processing Part

The data are gathered by the computer through PROFINET and then preprocessed so it can be sent to the cloud. The preprocessing is used to reduce the data volume in order of relieving the local network and Internet connection and also lowering the throughput of the data sent to a cloud, as it may be reflected in the cloud cost.

The data are processed both in the Edge PC and in the cloud in a pipe-flow manner through a number of consequent modules. It has also been kept in mind, that there could be multiple paths for data from one data source that can be split and processed independently. The data from different branches of the pipeline could also be merged. The general structure idea is depicted in Figure 3.2, where individual modules can represent either local edge PC process or a cloud infrastructure module.

3.1.2.1 Edge PC

A modular framework was designed for the Edge PC using the C# programming language with a small portion of the project also written in the C language used to wrap a PROFINET library. In this section, just the top-level abstract view will be described, while specific implementation notes will be elaborated in chapter 3.5.

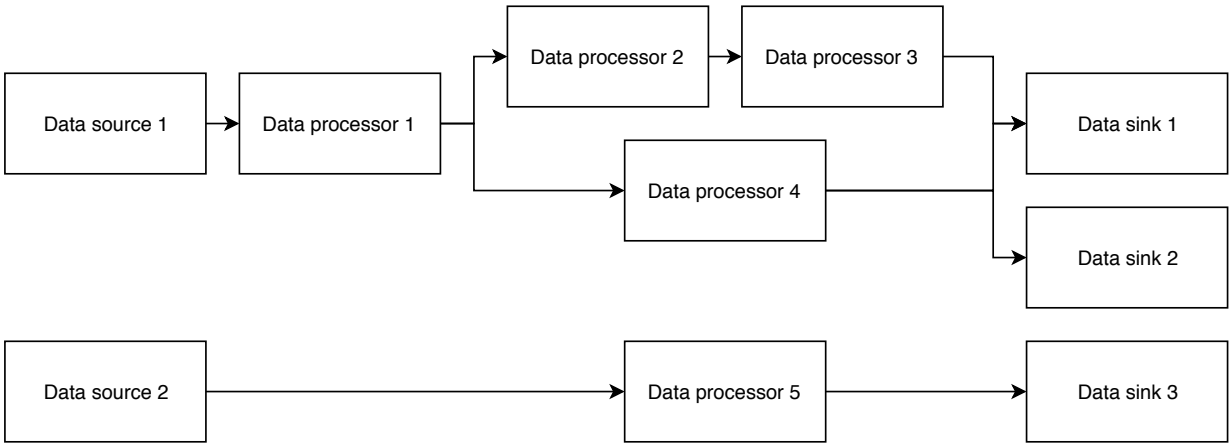


Figure 3.2: Data-flow idea structure of the Data Processing Part of the project.

To achieve the flexibility, a modular concept was chosen that was directly derived from Figure 3.2 where every module (i.e. rectangle in the Figure) is viewed as either data reading module - called the Data Sink - or as data producing module - called the Data Source - or both of these. Modules with both Sink and Source functionality will be further called Data Processor modules, while the modules with only Sink or Source functionality will be called Data Storage and Data Generator modules, respectively.

Thanks to this kind of representation, every module in the pipe can view its predecessor the same regardless if the predecessor is a Storage or a preceding Processor. The similar principle applies to the successor-side view. Framework's modular concept is also advantageous because it can be independent on the cloud solution and communication with the hardware - it is just necessary to implement an interface module to integrate new external resources, that is another industrial protocol or another cloud interface, and the rest of the framework can remain the same.

The architecture allows the framework to be modified to assign individual modules to different processing cores or even to different computers in a unified way. Also, a programmer can implement own modules just by extending an abstract module without knowledge about the core of the program. These modules can be then added to the pipeline that could be, with an extension, configured by an external configuration file or even a configuration tool.

The solution also has its disadvantages. For example, the concept is harder to grasp and to implement and therefore it is less prone to a human error. Also, there is more overhead when compared with a solution made specifically for an application as there is a need for a common interface between the modules.

3.1.2.2 Cloud

In the cloud, the data-flow structure follows similar mindset as in the Edge PC framework. Various modules that can be programmed and tested individually are used and intercon-

nected. Combination of Python and C# programming languages has been used. Python language for the machine learning specific tasks while C# for software engineering related tasks.

Microsoft Azure was selected as the cloud platform to use. The platform offers a wide selection of IoT services and machine learning services, as well as many other services like storages and virtual servers.

Alternatives to the Azure like AWS, Google Cloud or Siemens Mindsphere were considered too. From these, Microsoft Azure was chosen, because of its extensive support for IoT, also including support for deployment of various services directly onto the IoT devices. In addition to that, a broad selection of documentation and example applications is available online. In comparison to AWS and Google Cloud, the Azure offers a more comprehensive range of supported programming languages from high-level languages such as C#, Java, and Python to low-level C, which may prove advantageous in the diverse industrial environment.

Mindsphere offers industrial connectivity by providing PLC libraries. On the other hand, it lacks advanced features, like continuous data processing. Therefore a combination of Azure and Mindsphere could be considered as a future extension to integrate the application more into the industrial process.

3.2 Robot Configuration and Robotic Program

The robot was already configured to operate at the beginning of the project communicating over PROFINET. In this project, it was extended to transfer measurement data and also simple data sending mechanism was implemented.

3.2.1 PROFINET Interface Extension

In the WorkVisual software, it was first necessary to set larger PROFINET address space for the robot, from 512 bits to 2032 bits (Figure 3.3), as 512 bits was insufficient for sending two 32-bit variables for each of its six axes along with other control variables. Next, the extended address space had to be mapped to the robot's internal bit variables (Figure 3.4), the procedures for setting the number of PROFINET I/O bits and mapping a bus' bits to the internal variables are described in [14, p. 17] and [8, p. 81] respectively.

These bits was also expressed as variables so that it could be written easily from the robotic program. This was done in the file `Program/MACROS/ciirc_io.dat` by `SIGNAL` directive for each of the variables. For example, to map bits 1265 to 1296 a 32-bit integer variable named `OUT_POS_A1`, command `GLOBAL SIGNAL OUT_POS_A1 $OUT[1265] TO $OUT[1296]` was used. Numbers transferred from the robot were for the sake of simplicity transferred as a 32-bit integer in two's complement, as this representation is used in both the KUKA KRC [7, p. 439] and Windows 10 PC running on x86_64 platform.

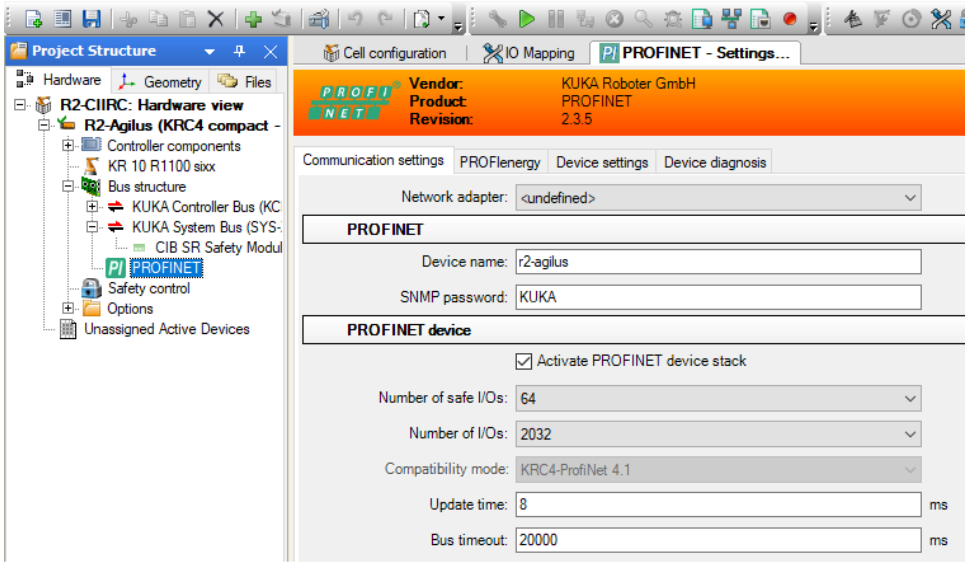


Figure 3.3: WorkVisual window with the size of the robot's address space

3.2.2 Data Acquisition Loop

The data used were acquired by the robot's internal sensors and accessed using system variables [28]. Two values for each axis were measured. Namely, `$CURR_ACT` (electric current of each axis) and `$VEL_AXIS_ACT` (actual angular velocity of each axis). There were also other variables that was not eventually measured, but could be worth investigating in a future project as the temperature of each motor (`$MOT_TEMP`), axes acceleration (`$ACC_AXIS`), gear jerk (`$GEAR_JERK`) or position of the manipulator in Cartesian space (`$POS_ACT_MES`) or the joint space (`$AXIS_ACT_MEAS`).

In addition to the measured variables, also a cycle number and operation number were transmitted. The cycle number is a value of a counter that counts up to a certain number and then it is reset. It is used by the Edge PC to distinguish new a measurement. The operation number is manually entered number of current operation used for algorithm evaluation.

3.3 PLC Data Relaying

The control system for the robot, Siemens Simatic PLC was used. It takes care of the safety matters and also serves as an OPC UA server to control the robots. For this project, it played a minor role. Nevertheless, it should be described, mainly from the system architecture point of view.

In the initial state, robots were connected to the PLC and controlled by PROFINET. In the set of the information transferred between the control system and the robots were

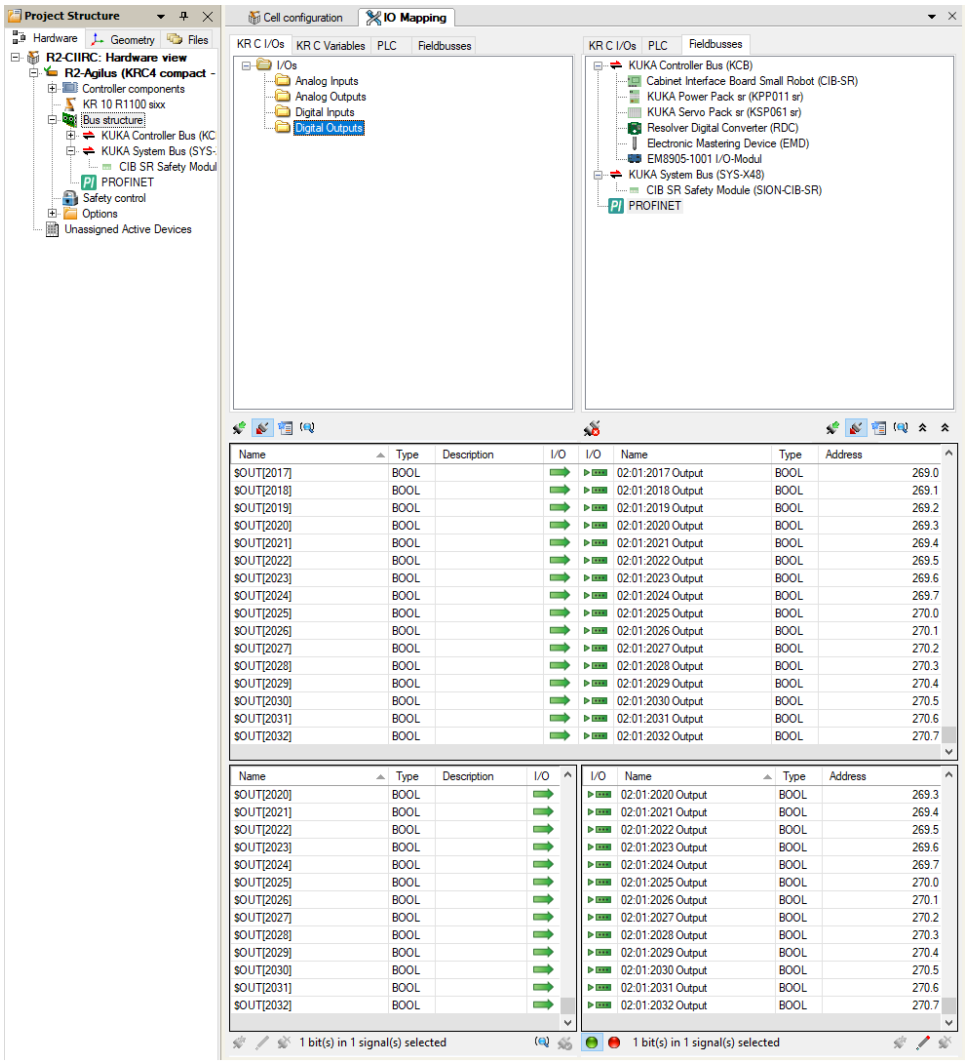


Figure 3.4: WorkVisual window showing some of robot's internal variables mapped to PROFINET address space

commands and flags, as `MOTORS_ENABLE` and also a set of different parameters of the robot movement, like a target coordinates in Cartesian space, reference frame and more.

This project required an extension of this interface to transmit the measurement data and some additional information (as described in section 3.1.1). To follow these targets, PROFINET relations between the components were set-up as follows - the robot was an IO Device of the control PLC and the control PLC was an IO Device of the measurement PC. The PLC was, therefore, set-up as an I-Device.

3.3.1 Variable Mapping and Data Relaying

The PLC setup introduced only minor changes into the architecture of the line. First, the mapping of the PROFINET variables had to be done. In TIA Portal a `Tag table` named `R<x>_MEASUREMENT_IO` was added for every robot to contain all the measurement tags. These tags were mapped to PROFINET addresses (input w.r.t. PLC, output w.r.t. robot) to match the structure the robot sends.

One confusing thing must be emphasised here. The difference in an addressing scheme in the Siemens Simatic and KUKA KRC and their developer programs TIA Portal and WorkVisual, respectively, must be noted. Siemens uses *bytes starting at zero*, while KUKA uses *bits beginning at one*. Moreover, KUKA is always counting from the *beginning of its address range*, while Siemens' addressing is *relative to the address of the module*.

For example, if there is a robot with an (input) address 100-200 in the TIA Portal, they would address the first bit of the first byte as `100.0` and the second bit of the third byte as `102.1` while in KUKA they would address these bits as `$IN[1]` and `$IN[26]`. Furthermore, this only holds if PROFINET addresses are mapped to the robot 1:1 (e.g., first PROFINET input bit is mapped to the variable `$IN[1]`, the second bit is mapped to `$IN[2]` and so on). Nevertheless, not the segmentation, nor the shift of the addresses was configured in this project so we can always assume, that the internal and PROFINET addresses match.

The input tags were subsequently written to the output of the PLC, the input of the measurement PC, using a `Transfer area`. The Area was set up to copy respective robot outputs (PLC inputs) to the measuring PC outputs (PLC outputs) in the `PROFINET interface [X1]` → `I-device communication` → `Transfer areas` in the `Device configuration` menu of PLC's configuration screen.

3.3.2 Data Representation

The PLC Siemens Simatic CPU1512 that was used to control the robots, and possibly also other PLCs of that series utilize another format of representing negative numbers than two's complement. Therefore the data read directly from the robot, which is visible in the input and output tags, do not represent the value that is stored in the memory.

To be able to read these values it would be necessary to convert these values in the robot and then convert them back to the two's complement in the measurement PC. Because the correct interpretation of the numbers in the PLC was not crucial for this project, this was omitted.

3.4 Overview of the System Architecture

The overall architecture of the project including the industrial devices, the Edge device, and the cloud looks as follows. This section summarises the architecture that is described step by step in the following sections.

Industrial Devices are used to collect the data from the robots (selection of the specific variables will be described in section 4.1). The robot is configured as PROFINET IO Device under a PROFINET IO Controller. The *Submit interpreter* in the robot periodically writes measured data to PROFINET mapped variables in a 32-bit integer formatted as little endian two's complement. Because the values measured are in order of tens or hundreds with several decimal points precision, it was possible to multiply these by a factor of 1000 to transmit more information without using any real number format (e.g., IEEE754 or similar). These values are read by the PLC and relayed to the Edge PC.

In the Edge PC, the data flow through a data processing pipeline. After the data are read by a PROFINET IO Device driver in bytes, they are converted to appropriate integer values by decoding byte arrays.

The converted values are then tested for a cycle number. The number of the robotic program is read to detect, whether the data read from the network contain a new measurement. The data that fulfill that are then converted once more by division by a factor of 1000 to real number (`double`) values.

The values are then aggregated to a set of time series for each operation by waiting for the operation number to change. The operation change could be also triggered by a signal from the control system.

From the individual time series, statistical moments (described in Chapter 4), the features of the data analysis algorithm, are calculated. That produces an array of size $number\ of\ time\ series * number\ of\ moments$.

When the program is a teaching mode, data are collected to teach the data model, therefore this array is saved in the cloud. If the model is already trained, the array is sent to a scaler and a dimensionality reduction module that reduces its dimension and only then it is sent to the cloud by Azure IoT Hub.

When the data reach the cloud, data are queried for a property, that specifies the route that data should take. In both cases, IoT Hub routes the message to an Event Hub that triggers a Function. If the data are marked by the property as non-processed data, the Function triggered saves the payload of the message to a storage. In the case of processed, that is scaled and reduced, the Function triggered queries a web service containing ML model that predicts an operation from the supplied data. The data and the output of the model are then saved to a storage.

In the following sections in this chapter, the implementation of the whole pipeline will be described. Only thing that is going to be omitted in this chapter is the ML model itself. The model will be described in Chapter 4.

3.5 Edge Computing

As discussed previously in section 3.1.2.1, a modular architecture has been chosen for data processing software to allow anyone to use the framework without a necessity to understand every part of the software. If there is, for example, a data processing functionality to be added, the programmer does not need to know, how the cloud API works, and when support for another cloud has to be added, there is no need to dig deep into data processing part of the program. The architecture was also designed to be extensible, configurable from a single place, possibly parallel, and also to be extended to a distributed system if necessary. The source code of the framework is located in the `ModularDataHarvester` directory.

3.5.1 Solution Structure

Software used to design the framework, Visual Studio 2017, allows separating a single software project to several smaller projects while encapsulating these into a greater structure called `Solution`. This functionality is used to make the project more readable and to separate different parts of the project.

The most important project within the solution is `HarvesterCore` that implements passing the messages via queuing mechanism between modules, the message structure and abstract structure of all three fundamental modules (Processor, Sink, and Source).

Specific implementations of different interfaces, such as Azure connector or PROFINET communication are implemented as separate projects. AzureModule implements the connection to the Azure cloud using the IoT hub functionality. It consists of multiple classes, but focus should be put on the class `AzureConnector` that extends class `CloudConnector` implemented in the Core. We can see the similar approach in the `ProfinetIOCModule` that implements the PROFINET communication by extending Core's `DataGenerator` class as `ProfinetIOCDataGenerator` class. Also, every data processing application is intended to be a separate project, so the data processing part of the thesis is implemented as `RobotDiagnosticsModule`. All of these modules will be described in the next few sections. Also, structure of the message will be described.

3.5.2 HarvesterCore

In the program, two essential interfaces and five classes are implemented along with many other implementations that are using these abstract bases. The two interfaces, `DataSource` and `DataSink`, describe the API necessary to connect these two modules one to each other. Next, there are classes that represent the three base module types - `DataGenerator`,

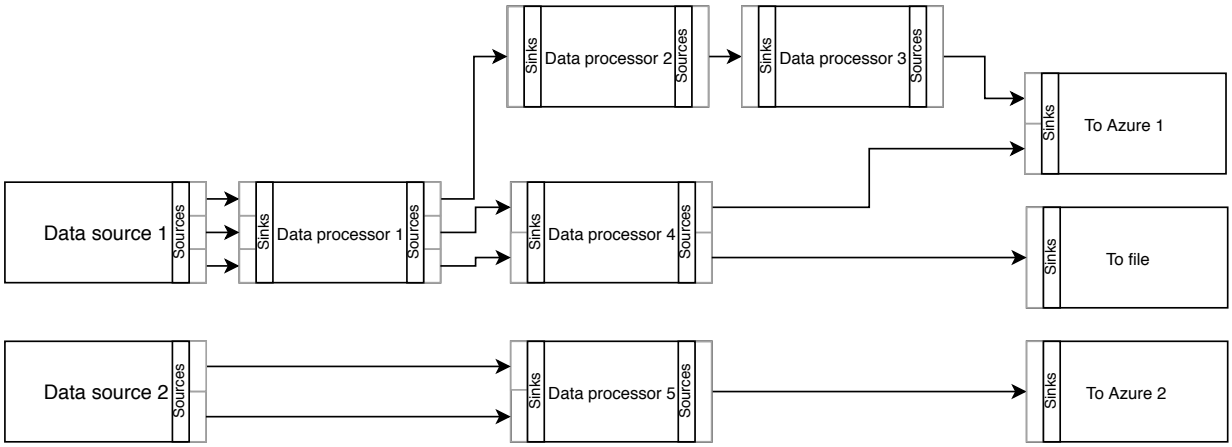


Figure 3.5: Data-flow structure of the data processing framework with Source and Sink modules.

`DataStorage` and `DataProcessor` - and two classes that represent input or output of a module - `SinkModule` and `SourceModule`, respectively.

Every `DataProcessor` has two lists consisting of inputs and outputs - number of `SinkModule` and `SourceModule` instances - that are bound to its predecessors and successors. Every `SourceModule` can be bound to multiple `SinkModule` instances of a succeeding `DataProcessor`, but `SinkModule` instances can be tied up with only one `SourceModule` instance. If we extend the example in Figure 3.2 also include these input and output modules, we get a more complex graph representing the structure of the pipeline (Figure 3.5).

3.5.2.1 DataSource and DataSink

Classes implementing one or both of these interfaces represent a `DataProcessor`, `DataGenerator` or `Storage` as a part of the pipeline. Both of these classes are bound to a module to handle data propagation (`DataSink` with `SinkModule` and `DataSource` with `SourceModule`) and are providing an API to work with these from a programmer-configuring-the-pipeline point of view. To be able to construct the pipeline easily, the programmer needs to access both the Sink and Source Modules and to interconnect them. To access the modules, methods `SinkModule GetInput(int num)` and `SourceModule GetOutput(int num)` are used. Additionally, the DataSource has method `PipelineBuilder RegisterSuccessor(DataSink successor)` that is used to register one or multiple links between this DataSource and a DataSource. The procedure of building the pipeline will be described in section 3.5.2.9.

3.5.2.2 SourceModule

The module serves as a single output-producing unit. The module has to be registered with its successor (`SinkModule`) first by method `void RegisterSuccessor(SinkModule`

`successor)`, that accepts the successor module to be registered as its argument. The module is then saved to a successor's database.

After that, the module can be used to propagate the data. When a `DataProcessor` or a `DataGenerator` wants to send a piece of data, it calls method `void WriteData(DataMessage msg)` of respective `SourceModule`. This method then calls `void BroadcastSingleMessage(DataMessage msg)` method that passes the message to all successors of this `SourceModule` by their `void Receive(DataMessage msg)` method.

`SourceModule` can be bound with an arbitrary number of successors. If the number of successors is zero, the message is lost. Nevertheless, there is no optimization done in this matter. Therefore the calculations that had been performed to process the message, will be done even if the module has no successors. This will result in an unnecessary load on the computer's resources and therefore it must be handled by configuring the application correctly.

3.5.2.3 SinkModule

When the input-consuming module, `SourceModule`, receives the data by `void Receive(DataMessage msg)` called by its predecessor, the message is enqueued, and a callback is triggered as a new task to release predecessors thread. The callback with signature `delegate void IncomingDataCallback(SinkModule m)` can be registered after initializing the module, although it is not completely necessary, as the presence of the data in the queue may be polled. The registration is done by calling `SinkModule`'s method `void RegisterIncomingDataCallback(IncomingDataCallback cbk)`.

Regardless of the approach used (callback or polling), the `DataSink` implementing object can then use methods `DataMessage GetData()`, `bool HasData()` or `IEnumerable<DataMessage> GetBufferEnumerator()` to work with the data.

3.5.2.4 DataProcessor

The processor is used to implement different data manipulating members of the pipeline while interfacing with the rest of the pipeline by `SourceModule`s and `SinkModule`s. It can be for example implementing a moving average filter (1 Sink, 1 Source), difference calculator (2 Sinks, 1 Source) or data type converter (N Sinks, N Sources). The data from the Sources are processed by a function and passed to given Sink.

When it is initialized, the Sink and Source modules have to be created and then registered by calling `void RegisterInputModule(SinkModule m)` and `void RegisterOutputModule(SourceModule m)`. The registration procedure adds the modules to a database and also registers `SinkModule`'s callback described in 3.5.2.3. The function that is called when a message is received by a `SinkModule`, `virtual void OnNewData(SinkModule module)`, checks whether there is a data on all inputs by calling function `virtual bool IncomingDataReady()`. If the condition is true, `abstract void ProcessData(SinkModule module)` is called.

`int GetNumOutputs()`	Get number of registered output modules.
`int GetNumInputs()`	Get number of registered input modules.
`bool InputEmpty(int num)`	Get `true` if the given input channel is empty.
`DataMessage GetDataFromInput(int num)`	Get a message from the given input.
`IEnumerable GetInputEnumerator(int num)`	Get an enumerator of the given input buffer.
`void WriteDataToOutput(int num, DataMessage msg)`	Write the given message to the given output.

Table 3.1: `DataProcessor`'s method that can be used to work with its inputs and outputs

`abstract` [29] `void ProcessData(SinkModule module)` method is responsible for the data processing itself. Therefore it is necessary to implement it in a way so that the `DataProcessor` performs the desired operation(s). Methods that are marked as `virtual` [30] are methods that can be overridden by another implementation in a child class, but in contrast to `abstract` methods, `virtual` methods can have a default implementation that is used if there is not a function in a child class to override it.

In the data processing function `ProcessData`, functions working with Sinks and Sources can be used as well as the direct access to `SourceModules`, and `SinkModules`. This is done by accessing elements in `Dictionary<int, SourceModule> SourceModules` and `Dictionary<int, SinkModule> SinkModules` dictionaries that are used to hold all the used modules. Methods available to work with the modules are listed in a table 3.1.

Two subclasses of the `DataProcessor` also worth mentioning are `BatchParallelDataProcessor` and `ParallelDataProcessor`. These are simple modifications of the standard `DataProcessor` and it is possible to work with them in the same manner as with its superclass. The `BatchParallelDataProcessor` is nothing more than a `DataProcessor` with the same number of inputs and outputs. The messages are processed in batches, as it is done in the superclass - all inputs must contain at least one message for processing to start. With the `ParallelDataProcessor` the situation is a bit different, as individual inputs are processed separately and the message is processed right after it is received.

3.5.2.5 DataGenerator

The Generator is a data-producing-only component of the pipeline, i.e. it is a block in a pipeline that is providing data to the pipeline and, generally, has no input. The data produced are sent as messages to the successive modules in the pipeline. In the message, in addition to the data itself, a time of the acquisition and originating module is sent.

This component can be usually used as an interface to a process or a device which the data are gathered from. It can generally be any device connected to the Edge PC by a network, bus or any other interface. Also, a local process running on the PC, a file or external database could be used.

In the project the class implementing this functionality is called `DataGenerator`. In contrast to the `DataProcessor` that implements both `DataSource` and `DataSink` interfaces, the class only implements `DataSource` interface. Therefore it provides `SourceModule GetOutput(int num)` and `PipelineBuilder RegisterSuccessor(DataSink successor)` methods to a pipeline programmer.

In the class itself, there is only a database of `SourceModule`s that has to be filled by `int addOutput(string name)` protected method by calling it from the subclass implementing specific functionality.

The specific functionality is intended to be independent of the `DataGenerator`s structure. Therefore programmers of the module shall implement their functionality based on threading, asynchronous callback, or by any other internal means. Then, the programmer only has to call `void WriteData(DataMessage msg)` of the specific `SourceModule` to propagate the data further. All the required `SourceModule`s have to be, already initialized.

3.5.2.6 DataStorage

On the contrary to the Generator and the Processor, the Storage *only consumes the data*, i.e. generally, it only has an input. The module represents a part of the pipeline, where the data are stored. For example, it can be an interface to a database, file or an internal buffer that is utilized by a processing mechanism which is not included in the pipeline.

The Storage mechanism is split into multiple classes of `HarvesterCore`: `Storage`, `StorageRecord` and `Storage`'s extensions `SingleStorage` and `MultiStorage`. First of those, `Storage`, is the essential class of the mechanism. `SingleStorage` and `MultiStorage` are extensions that specify, how the stored data are provided.

The Storage works as follows. After the initialization, the pipeline inputs and data-providing outputs are created. It must be emphasized here that the outputs are not a part of the pipeline. Therefore, it cannot be used to provide data to another pipeline modules. Inputs are then mapped to outputs in either 1:1 (`SingleStorage`) or N:1 (`MultiStorage`) scheme. Then, when a message is received by an input, it is, according to the mapping, stored in the buffer of the appropriate output.

Example To clarify this concept, let's use an example of a problem that this concept solves. Imagine a relational database with a given scheme that is used as a Storage. In the database, there is a table with the following fields (columns): measurement_id, device_id, position_x, position_y.

In this database, we want to store measurements of position (x, y) of two different devices. We use two separate `DataGenerator`s to get the data. Afterwards, we use two separate pipelines to somehow process the data. The processed data need to be stored in the table described above.

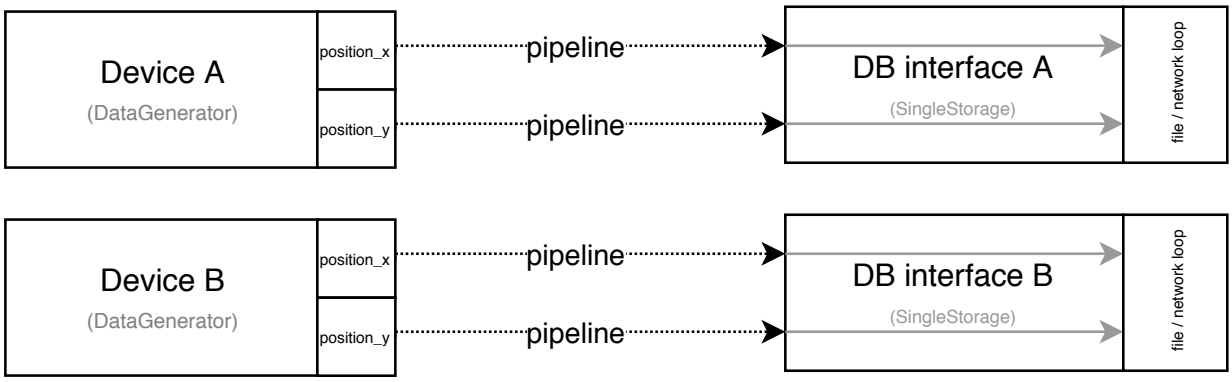


Figure 3.6: `SingleStorage` use example.

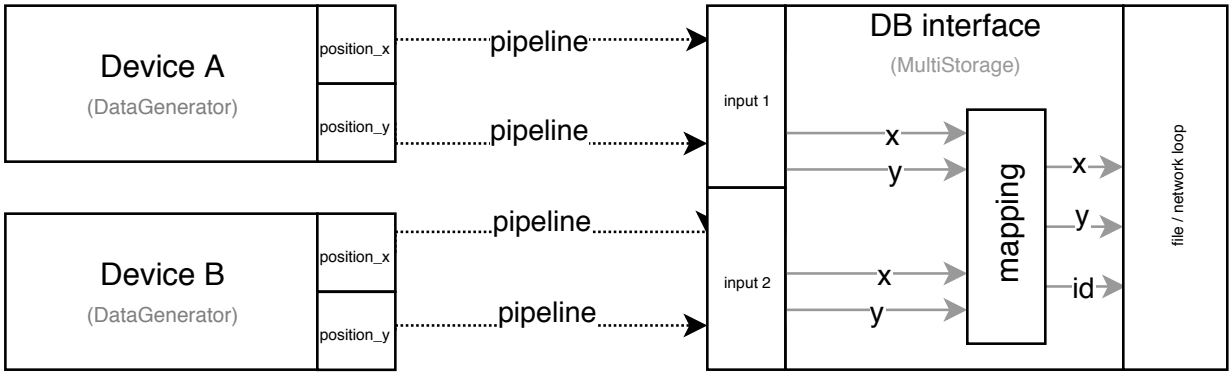


Figure 3.7: `MultiStorage` use example.

If we have had only had one device, we could have used a buffer that would be read in a loop, and its contents would be sent to the database. But we need to combine the data from two different branches of the pipeline (Figure 3.7), also we need to add fill in the device_id field and we also might need to change the structure of the data record. That is what is `MultiStorage` for, and `SingleStorage` is here to be used for the single-device scenario (Figure 3.6).

Storage, DatabaseBuffer and StorageRecord After the initialization of the object, inputs, outputs and the mapping, the module is ready to receive the data. When the storage receives the data on one of its inputs (`SinkModule`s), method `void OnNewData(SinkModule module)` is fired as `SinkModule`'s callback. The data are processed and stored in `DatabaseBuffer` where it is stored until `StorageRecord GetOutput()` is called by an external loop that is sending or saving the data to the storage itself. `DatabaseBuffer` is an extension of a queue of messages, `Queue<DataMessage>`, that also keeps its name and info structure.

The data of the output is stored in the `StorageRecord` object. The record keeps set of

the data from a single origin read at a single time frame. It can be looked upon as a row in a relational database. In addition to the origin, timestamp and the data, names of the respective fields are kept as well.

SingleStorage As said before, `SingleStorage` only maps its inputs to outputs in a 1:1 scheme (shown on 3.6). The mapping is therefore implemented as a dictionary with `SinkModuke` as a key and `DatabaseBuffer` as a value. The dictionary is filled by `void MapInputOutput(int input, int output)` that accepts the number of input and the output to tie. Both of these has to be already created.

Implementation of `void OnNewData(SinkModule module)` method is in this case about obtaining the target `DatabaseBuffer` from the dictionary and checking whether all the Buffers have any data stored by calling function `bool OutputDataReady()`. If there are data in all the buffers, the condition is fulfilled and the callback set in the initialization phase is invoked. After that, the data are ready to be accessed by calling `StorageRecord GetOutput()` either as a reaction to the callback or by a simple polling algorithm.

MultiStorage In the case of `MultiStorage`, the mapping gets more complicated. After inputs and outputs are created, it is necessary to group them. In the example shown in Figure 3.7, inputs would be grouped to sections "Input 1" and "Input 2".

Each input group has its `DatabaseBuffer` that is to be continuously filled with data. When an instance of data are added, the `void OnNewData(SinkModule module)` method is called, and it is checked whether any of the input groups has it data ready. It there is an such input, `StorageRecord` is created and prepared to be accessed via `StorageRecord GetOutput()`.

To simplify the work with the input groups, class `InputGroup` is internally used. Similarly to `Storage`, it has methods `bool OutputDataReady()` and `StorageRecord GetData()`. These methods are used to check and extract data from inputs bound with the `InputGroup`. In a pipeline building procedure a programmer should not interact with the class at all.

3.5.2.7 CloudConnector

We stated that `Storage` only serves as a local buffer. To transfer messages to a cloud, it is necessary to use `CloudConnector`. It is an abstract class that is intended to periodically collect messages from a `Storage`, convert them to the right format and send them to the cloud.

The Connector is first initialized by calling `bool Setup()` method, called from within the constructor. Then, method `void Run(bool blocking)` is to be run. This method should take care of collecting and sending the messages using, for example, a thread. To implement specific `CloudConnector`, programmers have to implement their own `Setup` and `Run` methods. The latter has also to implement a thread initialization and handling.

3.5.2.8 DataMessage and DataHandlerInfo

To pass data between individual modules of the pipeline a structure that encapsulates them was created to unify the interface and to couple the data with additional information. Functional requirements are: the data type of the data passed is arbitrary; the message can be empty/carry invalid data, so it is possible to send a message to synchronize two streams even if the data were filtered out; the message keeps its origin module; message also keeps a timestamp of the data.

Inputs and outputs of modules are initialized dynamically. This is due to reduced complexity and also to be able to support loading configuration from a configuration file in the future. Therefore it would be difficult to use Generics [31] to define its types and types of messages passed between them. Instead, `dynamic` [32] data type was used. Therefore, all the interfaces can just use `DataMessage` without specifying the type. Instead, the type (or typecast) must be specified within the function that manipulates the data.

Origin of the message is kept as a reference to `DataHandlerInfo`, the identification structure of each `Module` and `DataHandler`. As described earlier in the previous section, this structure is the only property defined in the `Module` and `DataHandler` interfaces that are used as base interfaces for `Sink/SourceModule`s and the basic pipeline modules. The handler contains an ID of the module within the parent handler, a reference to the `Module` and a reference to the `DataHandler`.

The DataMessage also has property `Properties` that keeps a message-related information used e.g. for routing.

3.5.2.9 PipelineBuilder

As described above, the pipeline is built by connecting `SourceModule`s to respective `SinkModule`s. Nevertheless, manually connecting every pair of modules could be very error prone, especially when building an extensive pipeline.

To overcome this problem and make the problem more natural for a human programmer, `PipelineBuilder` class was designed and implemented. It is a helper class which keeps reference to two `DataHandler`s passed to it in its constructor - `DataSink` and `DataSource` - on which more advanced design operations can be done. The Builder allows the programmer to use methods that connect `DataSink`s and `DataSource`s of the `DataHandler`s. One specific `DataSource` can be specified to connect with a `DataSink` as well as a range of these or a list of pairs.

To use the `PipelineBuilder`, it has to be constructed first. A programmer could either directly call class constructor, `PipelineBuilder(DataSource source, DataSink sink)`, that takes the two `DataHandler`s as its parameters, or to call method `RegisterSuccessor(DataSink successor)` of a `DataSource` object with a target `DataSink` as a parameter. This method calls the constructor of the Builder. With the `PipelineBuilder` initialized, a programmer can connect these two `DataHandler`s. To connect a different pair of Handlers, a new instance of `PipelineBuilder` has to be created. After the initialization, the Builder is then ready to be used.

To connect a Source with a Sink of a Handler, method `PipelineBuilder RegisterSourceSink(int sourceNumber, int sinkNumber, int sourceOffset = 0, int sinkOffset = 0)` can be used. Arguments of the functions are used to specify `Modules` to connect. Calling the function links `(sourceNumber + sourceOffset)`-th `DataSource` to the `(sinkNumber + sinkOffset)`-th `DataSink`. Offset arguments, which may seem redundant, are used to make the code more readable. Programmers can use it in cycles, where they would use

```
for(int i = 4; i < 10; i++)
{
        builder.RegisterSourceSink(i, i, 2, -1);
}
```

instead of

```
for(int i = 4; i < 10; i++)
{
        builder.RegisterSourceSink(2+i, i-1);
}
```

to connect `DataSources` $6 \ldots 12$ to `DataSinks` $3 \ldots 9$.

The same can be achieved by calling `PipelineBuilder RegisterSourceSinkRange(int sourcesFrom, int sinksFrom, int numElements)`. The method connects `numElements` consecutive `DataSources` to the same number of consecutive `DataSinks`. First Sink and Source in the sequence are specified by their numbers also passed as a parameter of this method. The same task as the one shown above using `RegisterSourceSink` can be achieved by calling `builder.RegisterSourceSinkRange(6, 3, 10)`.

If programmers wanted to register multiple connections, which are not in range, they could utilize the `PipelineBuilder RegisterSourceSinkCollection(ICollection<IOPair> pairs, int sourceOffset = 0, int sinkOffset = 0)` method, where a collection of `IOPairs` is passed. `IOPair` is a structure defined within the `PipelineBuilder` class, that keeps the Sink number and the Source number. Programmers could then use it as shown below.

```
builder.RegisterSourceSinkCollection(new List<IOPair>(){
new IOPair(){Source = 0, Sink = 1}, // register 0->1
new IOPair(){Source = 3, Sink = 4}, // register 3->4
new IOPair(){Source = 4, Sink = 5}, // register 4->5
new IOPair(){Source = 8, Sink = 2}  // register 8->2
});
```

As the functions return the same instance of the `PipelineBuilder`, they can be chained together, for example `builder.RegisterSourceSink(0, 0).RegisterSourceSink(5, 5).RegisterSourceSinkRange(8, 8, 10)`.

3.5.3 AzureModule

Azure IoT Hub (sec. 2.4.1) was selected to transfer the data to the cloud. Therefore it was necessary to implement its functionality in the control program.

To wrap the functionality, class `AzureConnector` extending class `CloudConnector` was implemented. As required by the extended class (sec. 3.5.2.7), `bool Setup()` and `void Run(bool blocking)` had to be implemented as well as `void TerminateGracefully()` and `void Terminate()` methods. Two latter methods are probably self explenatory and it is not necessary to describe them in a greater detail. Instead, we will focus on the two former methods. But before we dive in, the .NET library for Azure IoT Hub shall be mentioned.

The library called `Microsoft.Azure.Devices` is developed and published by Microsoft under MIT License. It allows a .NET application, among other functionality, to connect to the IoT Hub and serve as an IoT Hub Device [33]. The application is using version 1.6.0 of the library.

At the beginning, `AzureConnector` has to be initialized by supplying an instance of `AzureSetupProperties`. This `struct` is supplied as a part of `AzureModule` and it is used to pass parameters such as `Device name`, `IoT Hub Connection String`, `IoT Hub URI` or `Timeout` to the Connector. During the initialization, `Setup` method is run. In the chapter discussing `CloudConnector` class, it is mentioned that `Setup` method is responsible for initializing the data comitting mechanism. The method is determined, but not limited, to initialize the connection and, if required, prepare the thread that will gather the data and send it to the cloud. In the case of `AzureModule` it does both.

Within the `Setup` method, `AzureConnectorKeyLister` (supplied as a part of `AzureModule`) is initialized with the instance's `AzureSetupProperties`, which contains the connection details. Subsequently, the device is registered to the cloud and a connection key is received. If the device was already registered in the cloud, the registration is not repeated next time when the program starts. For the sake of simplicity, the key is not stored in the persistent memory of the device. The key is then used to authenticate the device. That yields `DeviceClient` class instance that allows the program to communicate with the IoT Hub. After the communication is initialized, `Thread` is initialized and set to run `void CommitLoop()`, but not started.

In the `Run` method, the thread is started. First, the connection to the device is open. Then the program enters an (in)finite loop that polls the input queue for data. When there are data, `StorageRecord`s are gradually dequeued from the input queue, serialized to JSON format and sent.

The input queue is filled by `void OnData(Storage s)` that is registered as a data-ready callback for a `Storage` preceding the pipeline.

3.5.4 ProfinetIOCModule and SimaticNetPNIOWrapper

As discussed previously, it was decided to use PROFINET for the data acquisition. For this purposes, we had two different drivers for PROFINET IO Controller available for Windows platform, both of them delivered by Siemens company. Both of these had the same C language API described in [34]. Therefore it was possible to switch between these two without any obstacles, if necessary.

3.5.4.1 PROFINET C library

Because the Edge device project was written in C# language, it was not possible to directly integrate C code into it. There are at least two ways to do such kind of integration. First of these is to use C++/CLI language to wrap the library and then call this wrapper directly from the C# code. The other method was to use dynamically loaded libraries (DLL) that would be called by the C# code. The latter method was used because its use is more widely documented.

After experimenting with both libraries, the first one called `PROFINET Driver for controllers` and the second one that is part of `SIMATIC NET PC Software V14` it was decided to use the latter one as it was compatible with the used project management tool, TIA Portal v14. The former library was only compatible with v13 at the time. Although a way of integrating the former library with the project by splitting the TIA Portal project to two different parts each in the distinct software version was proposed by Siemens Support and successfully tested in our setup, it was decided not to use it as it added undesirable complexity to the project itself. The former library also only supported x86 CPU architecture.

The library was delivered as a bundle of static Windows libraries for x86 and x86_64 platforms with a C header file and an example application, the integration of the library into a .NET program required to use DLL. The library also supports a wide variety of functions that were not planned to be used, for example, sending and receiving alarms, ProfiEnergy for reading power consumption data, and more. Therefore it was decided to write a C wrapping library that simplified the PROFINET library API and was also compiled as a DLL. This library is included in the solution as `SimaticNetPNIOWrapper` project.

Function headers with `pnw_` prefix that also has `__declspec(dllexport)` and `__stdcall` modificators are available in `SimaticNetPNIOWrapper.h` file. These are the functions that are to be called by the C# program. The first modificator, `__declspec (dllexport)`, specifies that the function is to be exported, i.e. it can be accessed by an external program when the library is compiled as DLL [35]. The second one, `__stdcall`, indicates the used call convention – stdcall [36].

In the program source file, `SimaticNetPNIOWrapper.c`, there is an implementation of these functions. All these functions return an error code that represents a value of `PNW_RETURN_VAL` enum returned as `int32_t` used to emphasize the type of the returned value is a 32-bit integer. This was crucial to be able to correctly integrate these functions

Function Name	Address	Relative Address	Ordinal
pnw_close	0x0000000180011159	0x00011159	1 (0x1)
pnw_deinit	0x00000001800112c1	0x000112c1	2 (0x2)
pnw_init	0x00000001800111e0	0x000111e0	3 (0x3)
pnw_last_pn_error	0x00000001800110dc	0x000110dc	4 (0x4)
pnw_open	0x00000001800111cc	0x000111cc	5 (0x5)
pnw_read	0x00000001800111ae	0x000111ae	6 (0x6)
pnw_set_alarm_cbk	0x0000000180011078	0x00011078	7 (0x7)
pnw_write	0x0000000180011127	0x00011127	8 (0x8)

Table 3.2: Table listing PROFINET wrapper functions exposed to DLL.

with the C# code. To use the library, first the `int32_t pnw_init()` function has to be called to initialize the PROFINET driver. It is desired to call the function only when the library is initialized.

Next, `pnw_open(uint32_t *handle, uint32_t cpId)` is to be called to run PROFINET controller identified by the CP ID passed in `uint32_t cpId` variable. The ID is given by the project setup in the TIA Portal (sec. 3.5.4.2). The variable, which is pointed to by a `handle` pointer, is used to store a number that identifies the particular instance of the PROFINET controller process. This number is further used to manipulate the controller (e.g., to close it or to read data from it).

If the initialization is successful, functions to read data, `int32_t pnw_read(uint32_t handle, uint32_t addr, uint8_t *inData, uint32_t inDataLen, uint32_t *readLen)`, and to write data, `int32_t pnw_write(uint32_t handle, uint32_t addr, const uint8_t *outData, uint32_t outDataLen)`, can be used. Variables `handle` and `addr` are used to identify the PROFINET controller instance and a module address to read or write. The other parameters specify a buffer to write the read data to, its length and the length of the data actually read, and in the case of the write access the buffer with the data to be written and its length. See commented code in the header file for further details.

PROFINET controller can also receive alarms. When such event happens, its contents are written to the standard output and, if set, an external function can be called. This is set by calling `void pnw_set_alarm_cbk(ALARM_CALLBACK alarm_cbk, uint32_t stdout)`. `ALARM_CALLBACK` and `ALARM_INDICATOR` are defined in the header. Exported functions are listed in table 3.2.

3.5.4.2 TIA Portal Project Integration

To include the IO Controller to the project in TIA Portal, `Simatic PC Station` component had to be added. Also, in the controller setup, an XDB file had to be generated in the `Simatic PC Station` → `XDB configuration` menu. This file contained a setup of the device and had to be imported to the driver setup.

On the PC running the PROFINET IO Controller driver, SIMATIC NET PC v14 had to be installed, and subsequently, the XDB file had to be imported via Station Configuration

Manager.

3.5.4.3 Integrating DLL into the C# code

Integration of this functionality is done in `ProfinetIOCModule` C# project, mostly in the `ProfinetIOCDataGenerator` class. This class defines the interface to the DLL functions and also does the data acquisition itself.

To import a function from DLL, `extern` keyword was used [37]. Programmer, among other parameters, can also specify the name of the function in DLL, if it differs from the name in C#, and the calling convention - stdcall in our case. For example, to import function `pnw_close` from our PROFINET driver wrapper, the statement below may be used.

```
[DllImport(@"ProfinetIOCModule/resources/SimaticNetPNIOWrapper.dll",
    EntryPoint = "pnw_close",CallingConvention =
       CallingConvention.StdCall)]
static extern ProfinetIOCStatus PnClose(UInt32 handle);
```

If the function imported is to return or be passed a pointer, the programmer also have to use `unsafe` keyword, for example `static extern unsafe ProfinetIOCStatus pnOpen(UInt32* handle, UInt32 cpId)`.

3.5.4.4 Pipeline Module

In addition to the DLL integration, `ProfinetIOCModule`'s `ProfinetIOCDataGenerator` also gathers data and provides it to the pipeline as a `DataGenerator`. This is done by `void GatherData()` function that is run as a thread that periodically gathers the data from the network and sends them as a `byte[]` to the pipeline.

`ProfinetIOCDataGenerator` has to be provided `ProfinetIOCConfig` and `ProfinetIOCInputPair[]` upon initialization. The first one keeps simple configuration parameters of the communication, such as period to read the data with, the second one defines a mapping between PROFINET address space and variables to be created. In the `ProfinetIOCInputPair` definition, the address of the data to be read and its length, also with the name of the module can be seen. `ProfinetIOCInputPairs` are mapped as `SourceModules` in a specified order, e.g., first element of the passed `ProfinetIOCInputPair[]` is mapped as the first `SourceModule`.

Module also provides two support `DataProcessors` - `ProfinetIOCFilteroutInvalid`, which filters out data that were read as invalid, and `ProfinetIOCIntegerConversionProcessor`, which is used to convert the array of bytes read by the `ProfinetIOCDataGenerator` to standard .NET integer of a given length.

3.5.5 RobotDiagnosticsModule

While modules previously mentioned can be used in a general project, this module contains a few `DataProcessors` implemented for this specific project. For example, it provides a functionality to segment the continuous measurement into individual operations or to calculate the necessary features from the data. As these `DataProcessors` are not dependent on each other, the functionality of each Processor will be discussed separately in a separate subsection.

3.5.5.1 RobotChangedDataProcessor

Because the period of data measurement cycle in the robot and the period of the PROFINET network differ, the same data are received by the Edge PC multiple times. Therefore it was necessary to distinguish the repeated instances of the data and to filter them out, which is done by `RobotChangedDataProcessor`.

This `DataProcessor` watches the cycle number (described in section 3.2.2), which is sent along with the measured variables by the robot. Whenever it changes, it lets all the remaining data on its inputs to go through. Otherwise, the input data are dropped and not forwarded to a successor in the pipeline.

The module also has a measure for detection of skipped cycles implemented. Based on current and maximum cycle number it predicts the next cycle number. On the next cycle change, this number is compared with the received one, and a warning message is displayed in the console if these two do not match.

3.5.5.2 RobotOperationAggregator

To be able to extract information for a given operation from the measurement, it is first necessary to segment the continuous time series to the time series of individual operations. To detect the moment, when the operation is changing to another one, the operation number (described in section 3.2.2) is used. Whenever its value changes, the time series can be cut and propagated further to the pipeline as an operation.

Internally, the Processor watches for a change in the operation number and simultaneously stores the incoming data into a buffer of type `RobotOperation<double>`. If the operation number changes, this buffer is sent as one message to the successor.

If a new measurement is received by the `RobotOperationAggregator`, method `Add(string name, DateTime acquisition, T sample)` of the buffer is called, where `name` is the name of the variable (e.g., "cur_A1" for the current of the first axis). This method adds values of all variables of this measurement to respective instances of `SeriesBuilder<DateTime, T>` which stores them. When it is necessary to access the series of the data, it can be accessed through methods `Series<DateTime, T> GetTimeseries(string name)` or `Dictionary<string, Series<DateTime, T>>.Enumerator GetEnumerator()` of the `RobotOperation<T>`. Classes `Series` and `SeriesBuilder` are part of the `Deedle` [38] library shared under BSD 2-Clause license.

3.5.5.3 StatisticalMomentsCalculator

When operations' time series are aggregated, features have to be extracted. The first three statistical moments of each time series - mean, variance and skewness - given by equations 3.1 [39] are extracted as described in section 3.2.2.

$$\mu = E(X), \qquad \sigma^2 = E(X-\mu)^2, \qquad skew = E(X-\mu)^3. \tag{3.1}$$

More on this topic These equations are already implemented in `Deedle` library (for sample mean) and can be accessed via `Series`'s methods `Mean()`, `Skewness()` and indirectly via `StdDev()` (this has to be squared to get the variance). Although according to the documentation this method should have also been available, due to a bug or a mistake, it couldn't be used as it was not accessible through the library API (*Deedle* v1.2.5).

These moments are calculated for each of the time series and subsequently are sent as a vector to a successor.

3.5.5.4 FeatureAgglomerationReductionTransformation

Dimensionality reduction is employed to reduce the amount of data sent to the cloud. In the case of Feature Agglomeration (see sec. 2.5.3) used in our project, the reduction is only matter of grouping multiple values together and calculating their average. Both the number of inputs and outputs of the module is exactly one. The module sinks and sources a `double[]` (array of real numbers).

Values that are to be grouped are specified in JSON configuration file in the format $[x_1, x_2, \ldots, x_n]$, where the value of x_i determines the group to which i-th input value belongs. Ordered list of values x_i specifies, which group is then written to which output value. This configuration is supplied to and interpreted by `FADROutputClusters` class object that provides grouped input numbers.

For example, array $[1, 2, 1, 3, 2]$ says, that 1st and 3rd input values are grouped and written as 1st output value; 2nd and 5th input values are grouped and written to 2nd output, and the 4th input value is alone and set as the 3rd output value.

3.5.5.5 DataTagger

This `DataProcessor` takes the input message and sets its "tag" property to a given string value. The message is then sent on the output of the module.

3.5.5.6 DataScaler

With transformation properties given by object of type `DataScalerProperties`, the module does standardization on `double` array that is carried by the input message according to equation

$$out = \frac{in - \mu}{\sigma}, \tag{3.2}$$

where μ is mean, σ is standard deviation of the data. Point-wise division of the vectors must be employed. This equation scales the input data to a data with zero mean and unit variance.

`DataScalerProperties` provides mean and standard deviation parameters to the scaler. These are loaded from a JSON encoded configuration file.

3.5.6 FactoryModule

This module is used to initialize the pipeline and is completely project dependent. It means, that for another project this module would be rewritten. Similarly, in case of implementing a configuration file based setup, this module would have to be replaced.

For this specific setup, it consists of a few static methods, `static void Main(string[] args)` among them, that create all the required objects. Method `static void RegisterRobot(string name, ProfinetIOCDataGenerator pngen, int inputOffset)`, for example, initializes all the objects of the pipeline for a specific robot and connects appropriate `DataSources` and `DataSinks` by using `PipelineBuilder` API. Method `static List<ProfinetIOCInputPair> GenerateRobotInputPairs(string name, UInt32 addressStart)` generates PROFINET mapping for the robots, i.e. a mapping between an PROFINET address (and data length) and a variable name used in the pipeline.

3.6 Cloud Setup

As described in section 3.4, used Azure cloud architecture consists of Azure IoT Hub, Azure Event Hubs, Azure Function, and Azure Table Storage services and Azure ML model. The first of these, IoT Hub, serves as an access point to the cloud for IoT Devices, while the others communicate within the cloud only.

When a message is received by the IoT Hub, it is routed according to its properties to one of the paths - the path for the measured data to be stored or the path for the reduced and scaled data to be processed by the model. In either way, the message is routed to an Event Hub and consequently passed to an Azure Function by which it is processed and stored. The architecture is depicted in Figure 3.8.

3.6.1 Teaching and Deploying the Model

First, the model was taught. The specific setup of the ML model will be elaborated in Chapter 4, here, the software implementation is focused. The model is taught in a Python 3 environment using open-source libraries such as Pandas and Scikit-learn.

The base file of the teaching algorithm is file `teach_test.py`, which is intended to be run for the teaching phase. In the file, the custom object of type `ModelTeacher` is initialized multiple times to perform K-folds cross-validation. Also, data scaling is done by sklearn's `StandardScaler` class via a helper function.

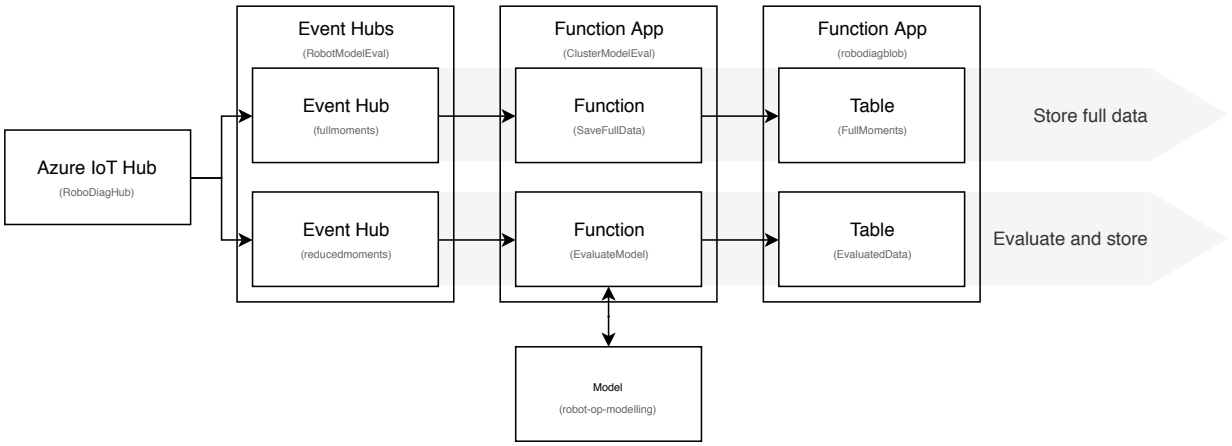


Figure 3.8: Architecture of the Azure solution.

In the `ModelTeacher` class, the data are fed to the model several times, every time with a different dimensionality reduction parameters. A decision is then made based on a given metrics, which of these reduction transformations gives the best results. Finally, the model with the best results is chosen.

The model, K-means, is represented by class `KmeansModel` that is extending class `ClusterModel`. The class has properties like `clusters_centers`, that gives center points of the clusters that the model represents, or `cluster_confidence`, returning a confidence, that the point lies within a cluster. Also, methods like `fit` or `predict` are here to teach the model and predict point's cluster. The `KmeansModel` internally uses sklearn's `KMeans` class.

Dimensionality reduction is done by sklearn's `FeatureAgglomeration`, that accepts a data set and a required number of dimensions to which the data set should be reduced. A naive approach was implemented to find the optimal reduction ratio using sklearn's `GridSearchCV`. It is a grid search algorithm that takes a list of values for each of the tuned parameters and goes over all possible combinations of these. The set of parameters which yields the best score is selected. A cross-validation can be also used for the selection.

For visualization, Matplotlib's `pyplot` library was used. To reduce the data set from its original space to 2D space, sklearn's PCA with the number of components equal to two was used.

3.6.1.1 Azure ML SDK Integration

The integration of the SDK is made by several function calls. First, an experiment is started by calling a helper function `run = init_azureml_experiment(...)` that calls SDK's

functions to open a workspace and initiate an experiment in the cloud. After that, the teaching procedure may begin normally. It is only necessary to interact with the SDK when the programmer wants to log data to the experiment log.

Experiment log can contain, for example, a single number or a series of numbers, a confusion matrix or a picture. Calling `run.log("accuracy", final_model.result.score_ksme)` would save the value of `final_model.result.score_ksme` as `accuracy` entry to the experiment log or `run.log_image("visualization", plot=plt)` would save a picture that is displayed on a pyplot's plot as a `visualization` entry.

At the end of the experiment, some files may be attached to the experiment; they can contain parameters or serialized models. A class called `DataSaver` has been made for this purpose. The class then calls for experiment's functions `upload_file(name=local_path, path_or_stream=local_path)` and `register_model(model_name=displayed_name, model_path=local_path)` to upload a file with the serialized model and, subsequently, register the file as a model.

3.6.1.2 Deploying the Model

When the model is saved and registered, the deployment to the web service could begin. The deployment was carried out using the SDK in three steps. In the first step, the web service was initialized. Here, parameters like size of RAM and description can be set. The second step involves the creation of the Docker image to be deployed. The image is created with parameters as `conda_file` that contains Python packages to be installed, `dependencies` list of other files to be included in the container or `execution_script` with a path to the scoring script to be run for each request. In the third step, the web service is deployed using the image configuration from the first step and the image from the second step. All of these steps are done via the SDK API (see `deploy.py` script). The service implements JSON REST API.

The scoring script (see `score.py`) comprises of two functions, `def init()` and `def run(raw_data)`. The former one is run only when the web service is started and is used for self-initialization. In this case, the model is loaded via the SDK and then deserialized. The latter function is run for every request to the web service API. It is passed a string that contains the payload sent by the web client. JSON formatted string is expected with property `data` containing an array of decimal numbers (see listing 3.1 for a sample request). The array is then passed to the model and the predicted operation (`label` field) with its confidence is returned. The is information is put into a Python dictionary and returned. The dictionary is converted to JSON in the background afterwards (listing 3.2).

```
{
  "data": [
    0.623309361769239,
    -0.085977239752716164,
    0.16520063120489359,
    0.033234002729132815,
    -0.26490188144599747,
    -1.6272617746185478,
    -0.135418783887069,
    1.3316906920817158
  ]
}
```

Listing 3.1: Example JSON request sent to the web service.

```
{
  "confidence": 0.00013167510709888308,
  "label": 4
}
```

Listing 3.2: Example JSON response received from the web service.

3.6.2 Interface Between the Edge PC and the Cloud

Transmission of the data to the cloud is done using the IoT Hub service. It is the interface between the Edge and Cloud worlds, so the setup had to be done on both sides. The Edge PC side is described in section 3.5.3, which explains the implementation of `AzureModule` within the C# framework. In this section, only the Cloud side of the solution is described.

3.6.2.1 Custom Endpoints

Every Azure IoT Hub can have up to 10 Custom Endpoints that can be connected to various services, for example, Event Hub or Service Bus. If a message reaches an Endpoint, it is passed to the service, that is connected to that Endpoint.

In the case of this project, we were interested in connecting the IoT Hub to Events Hub, as it is simple to connect the Events Hub to an Azure Function. Although it is possible to connect a Function to an IoT Hub directly, this has not been used, because message routing cannot be used in this scenario [40].

The setup was done in the Azure Portal under the service's configuration screen in `Message routing` → `Custom Endpoints` menu as it can be seen in Figure 3.9.

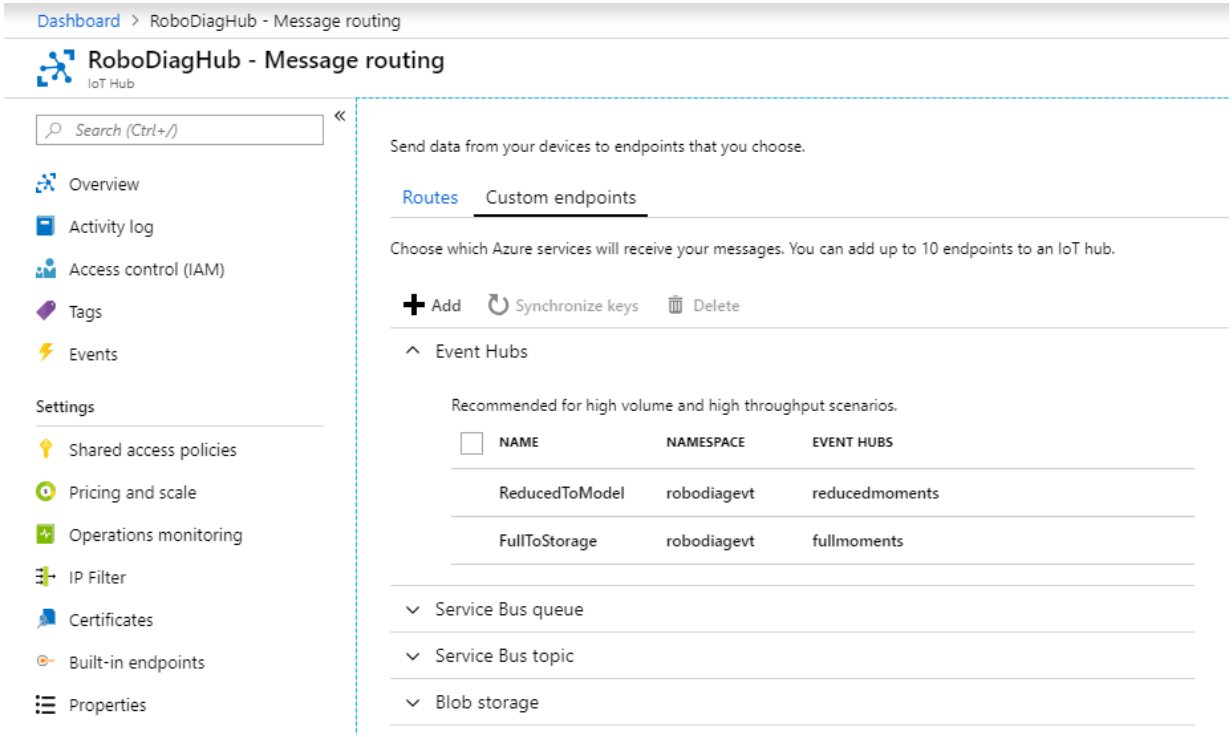


Figure 3.9: Custom Endpoints menu of the IoT Hub service.

3.6.2.2 Routing

The message sent by the Edge PC contains property `tag` that specifies the path the message should take. If the property is equal to `full-data`, the data are sent to the Endpoint that is connected to the data-saving-only part of the pipeline. If the property is equal to `reduced-data`, the message is routed through the prediction mechanism based on the ML model and saved afterwards. The routing setting, also with the selection queries, are depicted in Figure 3.10.

3.6.3 Interconnecting IoT Hub with the Model

To query the model with the data in the message, the Azure Function triggered to Event Hub has been used.

After the message is routed by the IoT Hub to one of its Event Hubs Endpoints, it is then passed to the respective Azure Function. In the case of the data that are to be processed by the model, the Function discussed is called `EvaluateModel` and its header can be seen in listing 3.3.

The statements delimited by brackets are called attributes [41] and in our case are used to add specifications of the input and output of the Function. The attribute beginning with the keyword `return` specifies that the value returned by the function is to be saved to a table

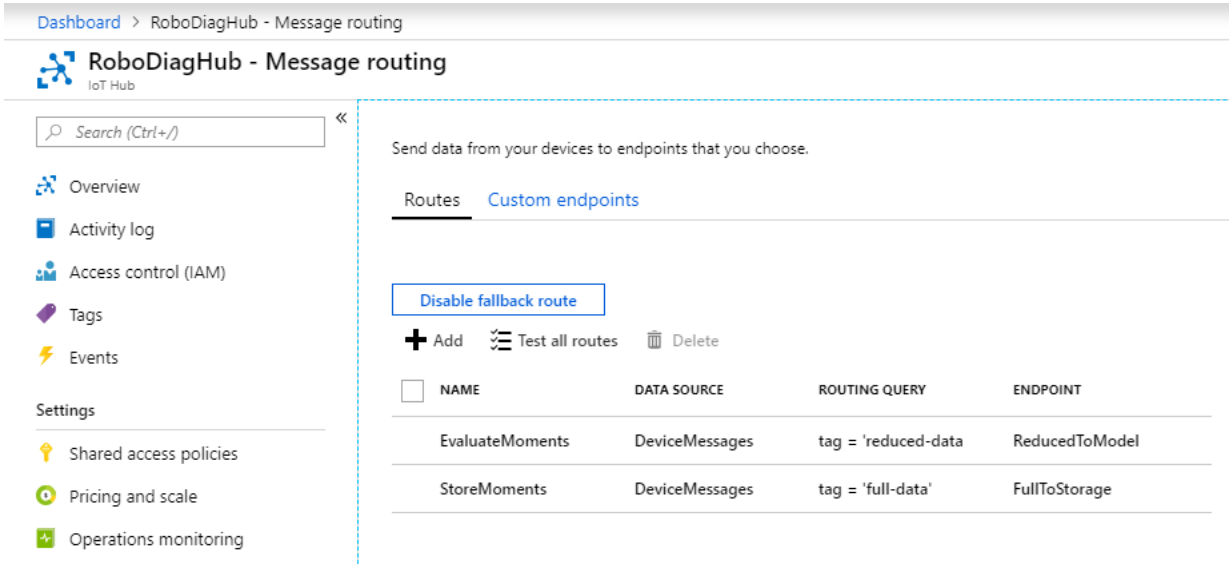


Figure 3.10: Routing menu of the IoT Hub service.

of Azure Table Storage called `EvaluatedData`. Connection to the table (incl. the name of the service and the access key) can be found under `AzureTablesDataStore` Application setting. Similarly, the attribute of `string message` argument of the function beginning with `EventHubTrigger` defines the connection to the Event Hub called `reducedmoments` by the Connection String stored as `EvaluateModelEvtHub`.

Application Settings can be found in the Azure Portal under the deployed `Function Apps` service in `Platform features` → `Application Settings` or in the Visual Studio IDE in project's `Publish` menu as `Manage Application Settings`. The Connection Strings to fill in are defined in the services connected to the Function (that is Event Hub and Table Storage) under `Shared access policies` menu of respective service.

```
[FunctionName("EvaluateModel")]
[return: Table(
  "EvaluatedData",
  Connection = "AzureTablesDataStore"
)]
public static EvaluatedDataRecord Run(
  [EventHubTrigger(
    "reducedmoments",
    Connection = "EvaluateModelEvtHub"
  )]string message,
  ILogger log
)
```

Listing 3.3: Header of the Azure Function for model evaluation.

In the Function body, several helper classes were implemented to parse the request and the response and build a web service request.

3.6.4 Storing the Data

`FullDataRecord` (for the store-only path) and `EvaluatedDataRecord` (for the model comprising path) classes were defined, because the Table Storage binding needs specific structure of the returned object to be able to save it. The object has to have `string PartitionKey`, `string RowKey` and `DateTimeOffset Timestamp` properties with a defined getter to be able to create a record in the database. The other properties of the object with the defined getter are saved to the record as well.

3.7 Chapter Summary

In this chapter, the interconnection of the robot and Edge PC using PLC has been described as well as the connection of the PC to the Azure cloud. The PLC is used to relay the data because, it is not possible to connect the robot and the computer directly. Furthermore, the implementation of the pipeline framework as well as the implementation of modules for this specific project was discussed. The last part of the chapter was dedicated to the description of the cloud environment and the implementation of the modules running within local model-teaching framework.

The whole pipeline, as well as the model teaching framework, works as expected. The data are collected from the robot, preprocessed locally and then sent to the cloud for further processing. As a future extension software for configuring the pipeline that would generate a configuration file with the description of the pipeline could be useful can be implemented as well as containerization (i.e. packing the respective programs into sandbox containers) and deployment of some Azure services, such as IoT Hub or Azure Functions to the Edge PC should be considered.

Chapter 4

Data and Model Evaluation

The previous chapter describes how the data were gathered and processed. In this chapter, we focus on the model. The goal is to find a model that would estimate the most likely robotic operation from the measured data also with some metric that would show, how likely the data instance belongs to that operation (e.g., likelihood).

4.1 Selecting and measuring the data

Originally, the collected data consisted of electric current, joint position, joint velocity and joint torque for each axis of KUKA Agilus robot. From these data, first four statistical moments (mean, variance, skewness, and kurtosis) and median were calculated.

In the first step, K-means was trained several times on (standardized) dataset, but with poor results. Average accuracy of these several runs was 62.1%, probably due to over-fitting.

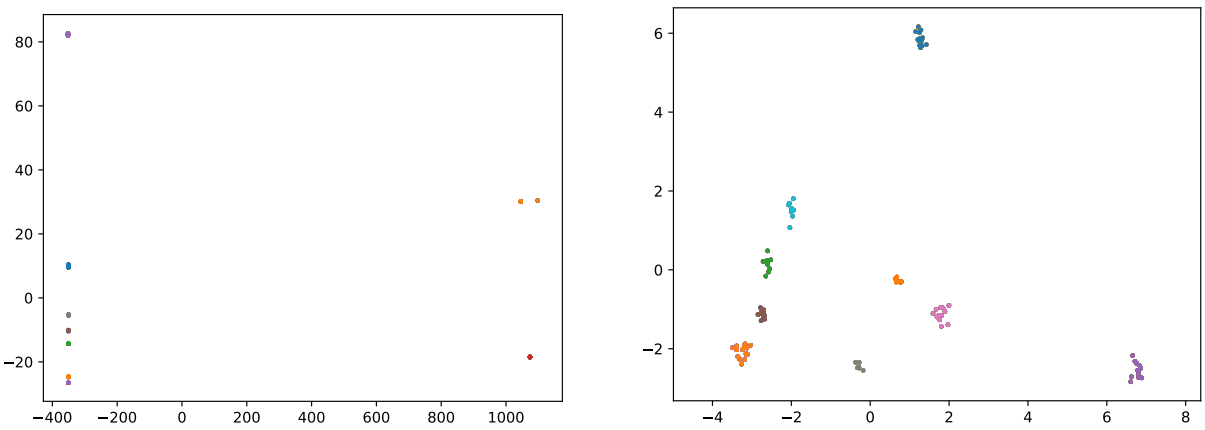
Correlation coefficients [42] were therefore calculated from the dataset consisting of approximately 400 measurements and it was found, that the joint torque highly correlates (over 98%) with the electric current; therefore it was removed. However, it did not improve the results significantly. Consequently, a few manual experiments were conducted, and it has been found that the model yields best results when only first three moments (mean, variance, skewness) of the electrical currents and velocities are taken into account. This procedure is called feature extraction.

With the K-means model this (standardized) dataset, the accuracy was over 99%. More about how the model was trained and how the accuracy was measured is be mentioned in section 4.3.

The feature extraction done on premise resulted in reduction of the data volume equal to

$$r = \frac{n_{axes} \times n_{moments} \times n_{timeseries}}{t_{operation} \times f_{sampling} \times n_{axes} \times n_{timeseries}}, \tag{4.1}$$

where n_{axes} is number of axes of the robot, $t_{operation}$ is a length of the inspected operation, $n_{moments}$ is number of statistical moments calculated, $f_{sampling}$ is sampling frequency of the



(a) Dataset before standardization

(b) Dataset after standardization

Figure 4.1: Comparison of the dataset before and after standardization to zero mean and unit variance.

measurement and $n_{timeseries}$ is a number of time-series measured for each axis. In our case $f_{sampling} = {}^{1}/_{0.012}\,Hz$, $n_{moments} = 3$ (for mean, variance and skewness) and $n_{timeseries} = 2$ (for current and velocity) and $n_{axes} = 6$.

For operations with length of one second, the reduction was approximately 96.4%, and for operation five seconds long, the reduction was 99.3%. The volume was further reduced by the dimensionality reduction.

4.2 Data standardization

Before diving into the modeling and reducing the data, it is usual to standardize the data [17]. In our case, standardization to the zero mean and unit variance was used by applying equation

$$\tilde{\boldsymbol{x}} = \frac{\boldsymbol{x} - \boldsymbol{\mu}}{\boldsymbol{\sigma}^2} \tag{4.2}$$

on data instances $\boldsymbol{x}$ (i.e. rows of the dataset) with mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ of the whole dataset. Obviously, the dataset is multidimensional, therefore $\boldsymbol{x}$, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are vectors. Point-wise division of two vectors must be used. Comparison of standardized and non-standardized dataset can be seen in Figure 4.1.

Because K-means creates (hyper)spherical regions around the clusters, the algorithm tends to perform better in the space, where the shape of the clusters is closer to a spherical one. It is, nevertheless, dependent on the dataset. In our case, it has been experimentally proven correct, as without the standardization K-means' accuracy was 75.2%, whereas, with the standardization, the accuracy reached up to 99.9%.

Over the whole chapter, the data we work with are considered to be standardized.

4.3 Teaching the model

As a first choice, K-means (described in section 2.5.1) model was used, as it is relatively simple to understand and it was already implemented in a well known Python library `sklearn`. This reduced a change of introducing an error, or bug, to the whole pipeline as the library is widely used[1]

Also, the learning algorithm is unsupervised, and therefore it does not need labeled data for training. Although we had labeled data, in future project this does not has to be true, the goal of this project therefore was to try the modeling with unsupervised data and use the labels just for the verification of the model.

The input data, statistical moments of time series measured from the robot, are standardized and then processed by a dimensionality reduction algorithm, Feature Agglomeration (described in section 2.5.3). The data reduced by the reduction algorithm are then used to train the K-means model.

4.3.1 Classification of the robotic operation

When the model is being trained, two important sets of parameters are calculated, as the model is not only used for classification of the robotic operations but also to determine a level of confidence, i.e. how likely is the prediction correct. The first one is done by the K-means algorithm, described in section 2.5.1. The latter one uses Mahalanobis distance (described in section 2.5.5) and its properties. To do the prediction, the algorithm needs two inputs, the data and number of operations present in the data. The data are measured from the robot and the number of operations can be observed from the control system or from the movements of the robot.

After the model is trained by K-means, an algorithm that generates function $f_c(\boldsymbol{x})$ of belonging of $\boldsymbol{x}$ to cluster c is run. The function is then equal to

$$f_c(\boldsymbol{x}) = 1 - F_{\chi^2_d}\left(D^2(C, \boldsymbol{x})\right), \tag{4.3}$$

where $F_{\chi^2_d}(x)$ is a Cumulative Distribution Function (CDF) of χ^2 (chi squared) distribution with d degrees of freedom (DOF) and $D(C, \boldsymbol{x})$ is Mahalanobis distance between point $\boldsymbol{x}$ and a distribution. The distribution describes the data belonging to cluster C.

By putting equations 2.4 and 4.3 together, we get a relation between parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ of a normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, point in a space $\boldsymbol{x}$ and a likelihood of $\boldsymbol{x}$ belonging to $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$,

$$f_c(\boldsymbol{x}) = 1 - F_{\chi^2_d}\left((\boldsymbol{x}_i - \boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu})^T\right), \tag{4.4}$$

with invertible covariance matrix $\boldsymbol{\Sigma}$.

[1] In Dec 2018 number of downloads of this library through PyPI, Python package repository, was over 1.8 million. This information was acquired using Python package **pypinfo 15.0.0**.

4.3.2 Dimensionality reduction

To reduce the dimension of the data, Feature Agglomeration algorithm was used. This algorithm requires two inputs, the dataset to be reduced and the expected number of dimensions, n_r. The second parameter was, however, not known. To overcome this, hyperparameter optimization[2] was used.

The expected number of dimensions is limited both from the bottom and from the top as a dataset cannot be reduced to a space with a lower dimension than one. At the same time, it is expected a *dimensionality reduction* algorithm *reduces* the dimension of a given dataset. Therefore the top boundary is lower than the original number of dimension n_o. Also, the number of dimensions n_r has to be a whole number. When the reduction is not feasible, because it would have caused a great loss of accuracy, the reduction does not have to be applied and $n_r = n_o$ can also hold.

The set of constrains, $0 < n_r \leq n_o$, $n_r \in \mathbb{N}$, gives us a finite set of numbers that can be inspected by a simple brute-force algorithm, if the space is small. In our case, when $n_o = 36$, this is feasible. To speed up the learning, at the expense of finding a slightly suboptimal solution, only even values of n_r are inspected,

$$n_r \in \left\{2k \,|\, k \in \mathbb{N}, 0 < k < \frac{n_o}{2}\right\} \cup \{n_o\}\,. \tag{4.5}$$

The model is taught with every value from this set. In our case, that means to teach the Feature Agglomeration algorithm on the input data with given n_r. K-means model has to be also retrained, because, when the reduction output dimension changes, K-means' input dimension changes consequently.

Sklearn's `GridSearchCV` algorithm is used for the brute force search. In addition to the search, it also does K-folds cross-validation for each value of the parameter and a mean of the scores is taken as the final score.

Variants of mixtures of Feature Agglomeration and K-means for inspected n_r are then scored by scoring function

$$S\left(X, M\right) = \begin{cases} -\infty & \text{if } s\left(X, M\right) < \lambda, \\ {}^{n_r}/_{n_o} & \text{otherwise,} \end{cases} \tag{4.6}$$

where $s\left(X, M\right)$ is an function that evaluates the model, while $s\left(X, M_1\right) > s\left(X, M_2\right)$ for better performing M_1 than M_2 by some metrics and λ is a threshold. Simply said, the scoring function S is equal to n_r/n_o if model M performs better than λ by metric s on data X and negative infinity otherwise. Parameter n_r giving the model with highest score is selected then.

4.3.3 Metrics used to evaluate the performance of the model

Two different metrics $s\left(X, M\right)$ were implemented and then compared.

[2] In machine learning, hyperparameter is a parameter of the algorithm, for example, the expected number of features in our case. Hyperparameter optimization is then a procedure that searches for an optimal value or a value close to the optimum.

The first of these, let's call it Supervised Metric, is provided with labels predicted by the model before any reduction algorithm has been applied to it. This set of labels is then compared with the set predicted by the reduced model, and a fraction of matching labels is then used as the score $s(X, M)$.

Comparison of the sets could not have been done directly by putting an equality sign between the elements of each pair of these two vectors, because the same clusters were labeled differently every time the algorithm has run. Because of this, a simple method to match these two sets was used.

First, the dataset is divided into groups each consisting only the measurements that have the same label. For each of this group, a mean, or centroid, is calculated. It is done twice, for the predicted labels and the original labels. That gives us two sets of centroids, Γ_p and Γ_o for centroids based on predicted and original labels respectively. Then, a distance between every pair of centroids $(\gamma_o, \gamma_p)\,;\, \gamma_o \in \Gamma_o,\, \gamma_p \in \Gamma_p$ is calculated.

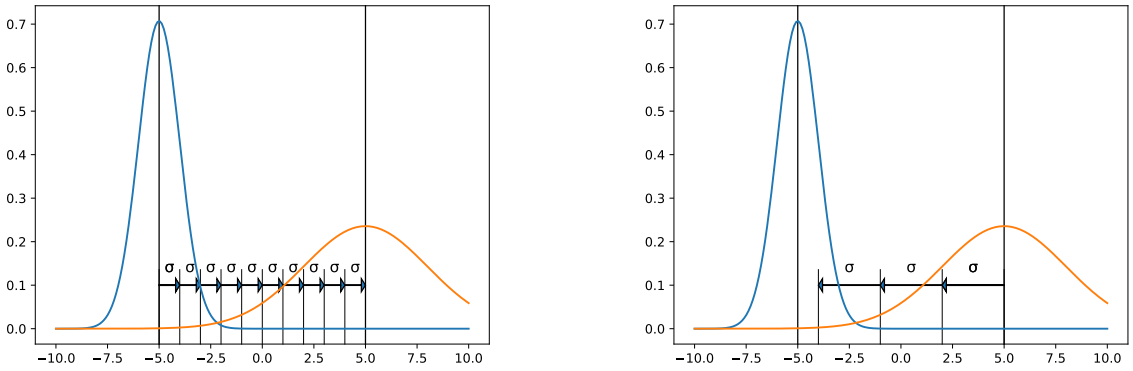
Afterward, for every γ_p the closest γ_o is found and a mapping from the label of γ_p to the label of γ_o is added to the mapping structure. This gives us a way to transform the label from the naming used by the reduced model to the naming that is used by the non-reduced model. The sets of labels that are using the same naming can be then directly compared by putting an equality sign between them.

The threshold parameter, λ, expresses an acceptable level of misclassification introduced by the reduction algorithm. If, for example, $\lambda = 0.05$, the reduced model can reduce the classification accuracy by at most 5% of the non-reduced model accuracy.

The second metric is based on Mahalanobis distance. The distance is calculated for each pair of clusters $c_i, c_j \in C;\, c_i \neq c_j$, where C is the set of all detected clusters for both directions. Therefore the distance is not only calculated from c_i to c_j, but also from c_j to c_i. This is done because the distance also depends on the covariance of the given cluster. Therefore two hypothetical clusters with distinct covariance matrices may have different Mahalanobis distance to point P even if their Euclidean distance from their centroids to that point is the same.

To calculate the distance from cluster c_i to cluster c_j, the Mahalanobis distance, taking the centroid and covariance of c_i, is calculated to the centroid of c_j and vice versa. Minimum of these two is taken as the distance because we are inspecting worst case situation (that is two clusters too close to each other). Parameter λ is, therefore, a minimal allowed Mahalanobis distance between two clusters.

To clarify once more, why we need to calculate the distance in both directions, 1D example is introduced. Mahalanobis distance in a 1D space is a distance between a point and a distribution's mean expressed as multiple of its standard deviations. To show this, we need to rewrite equation 2.4 to a form (equation 4.7) that holds for scalar mean μ and covariance Σ (therefore the covariance Σ is in fact variance σ^2) and then simplify it (equation 4.8). We consider x_i to be the centroid of the cluster to which we measure the



(a) The distance measured from the point of view of the blue distribution

(b) The distance measured from the point of view of the red distribution

Figure 4.2: Use of Mahalanobis distance for measuring a distance between two clusters.

distance.

$$d = \sqrt{(x_i - \mu)^2 \frac{1}{\sigma^2}} \tag{4.7}$$

$$d = \left| \frac{(x_i - \mu)}{\sigma} \right| \tag{4.8}$$

If we wanted to know a distance between 1D clusters represented by normal distributions with parameters $C_1 : \mu_1 = -5, \sigma_1 = 1$ and $C_2 : \mu_2 = 5, \sigma = 3$, we could do it by calculating Mahalanobis distance from cluster C_1 to C_2's mean and vice versa. Because σ's are different, we would have obtained two different Mahalanobis distances.

This can be seen in Figure 4.2, with C_1 depicted as the blue curve and C_2 depicted as the red curve, where the distance is illustrated by arrows pointing from the mean of the distribution to the target point. The length of the arrow is equal to the size of respective σ. From equation 4.8, C_1 to μ_2 is equal to $d(C_1, \mu_2) = 10$ and the distance C_2 to μ_1 is $d(C_2, \mu_1) \approx 3.33$. From these two distances, we would have chosen the worse one, i.e. the smaller one.

4.4 Validating the model

Model cross-validation is done by K-folds algorithm. The input dataset is divided into k parts, while one part serves as a test dataset and the other parts serve as the training dataset, this is done k times, while every time a different part is used as the test dataset.

In each iteration, the training dataset is reduced, the model trained and then verified by one of the metrics from section 4.3.3. Values of the metric are saved in each iteration and tested afterward for stability. Each value is then examined if it lays within ±10% region around the mean value. If this criterion is fulfilled, the model passes the stability test and the best behaving model among the tested is selected as the final model.

If the Supervised Metric is used, the cross-validation gives us an accuracy of all the tested models (as well as the mean of these) if the metric takes data labels as its input. The accuracy then expresses a fraction of correctly labeled measurements. Also, a confusion matrix can be obtained, that can be used for manual verification.

The metric based on Mahalanobis distance gives us the minimal distance between clusters, which is something that cannot be directly interpreted for evaluation. Nevertheless, the value can be used at least for model stability verification. An advantage of this method is that it does not need labeled data, whereas the Supervised Metric does.

For the whole procedure, the dataset was split into three parts - training, validation and testing dataset. The training and testing dataset is chosen by the K-folds algorithm. The validation dataset, that is used for optimizing the Dimensionality Reduction hyperparameter, is then taken from the training dataset by another, nested K-means algorithm that is called by the Grid Search algorithm.

4.5 Evaluation of the model

Two models have been taught and verified on two distinct datasets measured when two different robotic programs were run. Both models were validated by K-folds with $k = 3$ and run multiple times with shuffled dataset to obtain the statistics. The inner K-folds for tuning the target reduction dimension had its $k = 2$.

The first program consisted of a pick and place robotic operations that picked an object from one place and placed it to another. High-velocity point-to-point (PTP) movements were used to move the robot between the workspaces, while slow linear movements were used for the pick and place operation itself. Pick and place operations were programmed with the same z-axis offset and same velocities. The whole program was separated into eight distinct elementary robotic operations that were to be classified.

The second program contained similar operations, also pick and place, with both PTP and linear movements, but in this case, these two operations were not similar - different z-axis offsets and speeds were programmed for these operations. This program was separated into nine distinct robotic operations.

Due to the same z-axis speeds and offset is the similarity between some of the individual operations of the first program higher than that of the second program.

4.5.1 Robotic operation classification

From the two metrics, the Supervised Metric was selected, because it is easier to tune. Parameter λ has to be passed to the teaching algorithm and it has to be either guessed, calculated or optimized by some algorithm. In the case of the Supervised Metric, the parameter is much more intuitive to use. This metric relies on the correct classification of the unreduced data set, but this was working well on both of the evaluated data sets, therefore there was no need to use the second metric.

The model taught on approximately 350 measurements of the first program yielded accuracy 99.76% for the first dataset with a reduction optimum found at ten dimensions. Therefore the model was reduced from 36 dimensions to 10 dimensions (approx. 28% reduction). For the nonreduced dataset, the accuracy of the model was 100%.

The second model was taught on a dataset consisting of approximately 3600 measurements, and its statistic does not rapidly differ from the first dataset. The accuracy of the non-reduced model was also 100% and in the case of the reduced model 99.97%. The reduction optimum has been found in 4 dimensions. Hence the reduction ratio was approximately 89%.

Note, that the reduction ratio here is relative to the data volume of already reduced data by the feature extraction. Therefore, if we had 96.4% reduction of the approximately 1000 sample set of measurements to the 36-dimensional vector, this reduction from 36 dimensions to 4 dimensions (89%) gave us a total reduction of approximately 99.6% of the original volume.

4.5.2 Classification confidence

The model also produces confidence level that specifies, how likely is a point X classified as a member of cluster C belonging to this cluster. How this is calculated is explained in section 4.3.1.

The procedure has shown to work, but cannot be directly used for some thresholding method because values produced by this method ranged from 0.05 to 1.0 during the measurement for the correctly classified measurements. This happened, because the confidence level was expressed w.r.t. to the cluster itself, e.g., it was the likelihood of measuring data X if it belongs to cluster C, $\ell(C \mid X)$. Because of this, the confidence cannot be directly compared between clusters.

Also, when confidences of belonging to the remaining clusters were calculated, only 11% of measurements had exactly one non-zero value, while no measurement had more than one non-zero value. It was caused by numerical limitations, as it was either not possible to express the probability as `double` data type or the precision was lost during the computation.

4.5.3 Running the algorithm

Average runtime of the teaching algorithm, including all three K-folds iterations, is 15 seconds for the first dataset and 45 seconds for the second dataset on a computer equipped with Intel i74810MQ @2.80GHz CPU and 16GB of RAM. The algorithm runs as one process on one core of the CPU.

4.6 Chapter Summary

A model for identification of robotic operations was taught from currents and velocities of the robot's axes. From these time-series, mean, variance and skewness were extracted as features for the K-means algorithm. Data were then standardized, its dimension was reduced by Feature Agglomeration algorithm and, at last, K-means model was trained. This procedure has been validated by K-folds to verify its invariance on the measured data. In addition to this, calculation of the likelihood of the point belonging to a class has been implemented.

The reduction and classification algorithms behaved well with relatively high reduction ratio and classification accuracy. While the confidence function also behaved correctly, it cannot be directly used for comparison between the clusters, or a thresholding, because the values produced by the function first has to be expressed as posterior probability $p(C \mid X)$. Currently, only likelihood $\ell(C \mid X) = p(X \mid C)$ is calculated.

Further research is required to find the solution, for example, using a different algorithm for the classification, that can provide the posterior probability directly could lead to the solution.

Chapter 5

Conclusion

The goal of the project was to implement a set of programs to gather data from an industrial robot, pre-process it on-premise and then send it to a cloud for further processing. Microsoft Azure cloud was eventually selected, because of its rich IoT features.

The data, measurements of electric currents and velocities of the robot's axes, were gathered through PLC, in order not to break the original setup of the laboratory equipment. Next, Edge PC connected to the PLC via PROFINET did on-premise pre-processing of the data. The pre-processing comprised of extracting features, that is input data, for the machine learning algorithm and then reducing its dimension.

The extracted features were the first three statistical moments (mean, variance and skewness) of each measured time-series. Features was then reduced by Feature Agglomeration algorithm. From the original data, which consisted of the set of measurements of currents and velocities, the feature extraction and dimensionality reduction was able to reduce the volume of the data by more than 99%.

The reduced data were sent to Microsoft Azure Cloud via Azure IoT Hub service, processed by a classifier that classified the robotic operation represented by the data, and everything was saved to the Azure Storage Tables database. The list of classified operations can be used for a further diagnostics. Sequence of the operations or a trend of the confidence measure can be observed over time.

The classification is based on a K-means model, that is taught by the reduced features extracted from the measurements. The model is cross-validated by K-folds during the learning. In addition to the classification, the likelihood of point belonging to a cluster (i.e. likelihood of measurement being a representation of specific robotic operation) is calculated by a method based on Mahalanobis distance. Accuracy of the classifier was more than 99% with the tested datasets.

When the model is taught, also an optimal reduction ratio is found learning different models by trying various target dimensions to reduce to and searching for the best among these. When searching, the selection is also validated by K-folds.

Suggested next steps in the project would be creating a GUI and configuration file format for configuring the pipeline, also deployment of some Azure services to the Edge

PC should be considered. For the model, calculation of the probability of a point lying belonging to a cluster, $p(C \mid X)$ instead of likelihood should be introduced. Also, a search for other physical quantities that characterize the process should be carried out. Ultimately, it could be favorable, if the process would not require a change of the current industrial architecture and robotic program at all. For example, power measurement as used in [43, 44] could be employed.

Bibliography

[1] Maximilian Zarte; Agnes Pechmann. Concept for introducing the vision of industry 4.0 in a simulation game for non-IT students. July 2017, doi:https://doi.org/10.1109/INDIN.2017.8104825.

[2] Geissbauer, R.; Schrauf, S.; Koch, V.; et al. Industry 4.0 - Opportunities and Challenges of the Industrial Internet. Dec. 2014. Available from: `https://www.pwc.nl/en/assets/documents/pwc-industrie-4-0.pdf`

[3] Bassi, L. Industry 4.0: Hope, hype or revolution? Sept. 2017, doi:https://doi.org/10.1109/RTSI.2017.8065927.

[4] ElMaraghy, H. A. Flexible and reconfigurable manufacturing systems paradigms. *Int J Flex Manuf Syst*, 2005, doi:https://doi.org/10.1007/s10696-006-9028-7.

[5] Radziwon, A.; Bilberg, A.; Bogers, M.; et al. The Smart Factory: Exploring Adaptive and Flexible Manufacturing Solutions. 2014, doi:https://doi.org/10.1016/j.proeng.2014.03.108.

[6] KUKA Deutschland. KR C4 compact, Specification. 2018, accessed: 2018-09-14. Available from: `https://www.kuka.com/-/media/kuka-downloads/imported/48ec812b1b2947898ac2598aff70abc0/spez_kr_c4_compact_en.pdf`

[7] KUKA Roboter. KUKA System Software 8.3, Operating and Programming Instructions for System Integrators. 2015, accessed: 2018-09-14. Available from: `http://www.wtech.com.tw/public/download/manual/kuka/krc4/KUKAKSS-8.3-Programming-Manual-for-SI.pdf`

[8] KUKA Roboter GmbH. WorkVisual 4.0 For KUKA System Software 8.2, 8.3 and 8.4. May 2015, accessed: 2018-09-28. Available from: `http://www.wtech.com.tw/public/download/manual/kuka/KST_WorkVisual_40_en.pdf`

[9] KUKA Roboter. Quickguide KRL-Syntax. 2012, accessed: 2018-09-14. Available from: http://www.wtech.com.tw/public/download/manual/kuka/krc4/KUKAKRL-Syntax8.x.pdf

[10] Siemens. PROFINET IO, From PROFIBUS DP to PROFINET IO, Programming Manual. Feb. 2007, accessed: 2018-09-14. Available from: http://www.siemens.fi/pool/products/industry/iadt_is/tuotteet/automaatiotekniikka/teollinen_tiedonsiirto/profinet/man_pbdpio_to_pnio.pdf

[11] Siemens. PROFINET System Description, System Manual. June 2008, accessed: 2018-09-14. Available from: http://www.siemens.fi/pool/products/industry/iadt_is/tuotteet/automaatiotekniikka/teollinen_tiedonsiirto/profinet/man_pnsystem_description.pdf

[12] Siemens. Configuration manual "Shared Device". Nov. 2016, accessed: 2018-09-21. Available from: https://support.industry.siemens.com/cs/attachments/109741600/109741600_config_SharedDevice_en.pdf

[13] Siemens. I-Device Function in Standard PN Communication. Aug. 2015, accessed: 2018-09-21. Available from: https://cache.industry.siemens.com/dl/files/798/109478798/att_856108/v3/109478798_config_idevice_standard_DOCU_V1d0_en.pdf

[14] KUKA Roboter. KUKA.ProfiNet Controller/Device 3.1 For KUKA System Software 8.3. Sept. 2013, accessed: 2018-09-21. Available from: http://supportwop.com/IntegrationRobot/content/9-Peripherie_Entrees_Sorties/Profinet/8.3/KUKA_ProfiNet_31_en.pdf

[15] Microsoft Azure Documentation. Accessed: 2018-12-14. Available from: https://docs.microsoft.com/en-us/azure/

[16] Klein, S. *IoT Solutions in Microsoft's Azure IoT Suite: Data Acquisition and Analysis in the Real World*. apress, 2017, ISBN 978-1-4842-2143-3.

[17] Murphy, K. P. *Machine Learning A Probabilistic Perspective*. The MIT Press, 2012, ISBN 978-0-262-01802-9.

[18] Bifet, A.; Gavaldà, R.; Holmes, G.; et al. *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press, 2018, ISBN 978-0-262-03779-2.

[19] Jolliffe, I. T. *Principal Component Analysis, Second Edition*. Springer, 2002, ISBN 978-0-387-22440-4.

[20] Clustering - scikit-learn 0.20.1 documentation; Hierarchical clustering. Accessed: 2018-12-14. Available from: https://scikit-learn.org/0.20/modules/clustering.html#hierarchical-clustering

[21] Newman, M. E. J. *Networks: An Introduction*. Oxford University Press, 2010, ISBN 978–0–19–920665–0.

[22] Ward, J. H. Hierarchical Grouping to Optimize an Objective Function. 1963.

[23] Wong, T.-T. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. *Pattern Recognition*, volume 48, no. 9, 2015: pp. 2839 – 2846, ISSN 0031-3203, doi:https://doi.org/10.1016/j.patcog.2015.03.009.

[24] Maesschalck, R. D.; Jouan-Rimbaud, D.; Massart, D. The Mahalanobis distance. *Chemometrics and Intelligent Laboratory Systems*, volume 50, no. 1, 2000: pp. 1 – 18, ISSN 0169-7439, doi:https://doi.org/10.1016/S0169-7439(99)00047-7. Available from: http://www.sciencedirect.com/science/article/pii/S0169743999000477

[25] Manly, B. F. *Multivariate statistical methods : a primer*. Boca Raton, FL : Chapman & Hall/CRC Press, third edition, 2005, ISBN 1584884142.

[26] Kim, W.; Sung, M. Standalone OPC UA Wrapper for Industrial Monitoring and Control Systems. *IEEE Access*, volume 6, July 2018, doi:https://doi.org/10.1109/ACCESS.2018.2852792.

[27] KUKA. KUKA KR 10 R1100 sixx. Available from: https://www.kuka.com/-/media/kuka-downloads/imported/6b77eecacfe542d3b736af377562ecaa/0000210360_en.pdf

[28] KUKA Roboter GmbH. System Variables For KUKA System Software 8.1, 8.2 and 8.3. Aug. 2012, accessed: 2018-09-28. Available from: http://www.wtech.com.tw/public/download/manual/kuka/krc4/KUKASystemVariables8.18.28.3.pdf

[29] abstract (C# Reference). 2015, accessed: 2018-10-19. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract

[30] virtual (C# Reference). 2015, accessed: 2018-10-19. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual

[31] Generics (C# Programming Guide). Accessed: 2018-11-02. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/

[32] dynamic (C# Reference). Accessed: 2018-11-02. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/dynamic

[33] Microsoft Azure IoT SDK for .NET. Accessed: 2018-11-09. Available from: https://github.com/Azure/azure-iot-sdk-csharp

[34] SIMATIC NET, PROFINET IO-Base user programming interface, Programming Manual. Oct. 2015.

[35] Exporting from a DLL Using __declspec(dllexport). Accessed: 2018-11-16. Available from: https://msdn.microsoft.com/en-us/library/a90k134d.aspx

[36] __stdcall. Accessed: 2018-11-16. Available from: https://msdn.microsoft.com/en-us/library/zxk0tw93.aspx

[37] extern (C# Reference). Accessed: 2018-11-16. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/extern

[38] Deedle: Exploratory data library for .NET. Accessed: 2018-11-23. Available from: http://bluemountaincapital.github.io/Deedle/

[39] Casella, G.; l. Berger, R. *Statistical Inference*. Duxbury, second edition, 2002, ISBN 0-534-24312-6.

[40] Tutorial: Configure message routing with IoT Hub. Accessed: 2018-12-14. Available from: https://docs.microsoft.com/en-us/azure/iot-hub/tutorial-routing

[41] Attributes (C#). Accessed: 2018-12-20. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/attributes/

[42] Fisher, R. A. *Statistical Methods for Research Workers*. Springer New York, 1992, ISBN 978-1-4612-4380-9, doi:10.1007/978-1-4612-4380-9_6.

[43] Ron, M.; Burget, P. Stochastic modelling and identification of industrial robots. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, Aug 2016, ISSN 2161-8089, pp. 342–347, doi:10.1109/COASE.2016.7743426.

[44] Ron, M.; Burget, P. Density based clustering for detection of robotic operations. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, Aug 2017, ISSN 2161-8089, pp. 314–319, doi:10.1109/COASE.2017.8256122.

Appendix 1

Base structure of the attached disc

```
Root directory
├── IndustrialPart
│   ├── PLC
│   └── Robot
├── DataAnalysis
├── ModularDataHarvester
└── Thesis.pdf
```