Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

# Acquisition and analysis of movement data from industrial robots

Bc. Petr Cezner

Supervisor: Ing. Pavel Burget, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Cybernetics and Robotics
May 2019

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

| | | | |
|---|---|---|---|
| Příjmení: | **Cezner** | Jméno: **Petr** | Osobní číslo: **434918** |
| Fakulta/ústav: | **Fakulta elektrotechnická** | | |
| Zadávající katedra/ústav: | **Katedra řídicí techniky** | | |
| Studijní program: | **Kybernetika a robotika** | | |
| Studijní obor: | **Kybernetika a robotika** | | |

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Sběr a analýza pohybových dat z průmyslových robotů**

Název diplomové práce anglicky:

**Acquisition and analysis of movement data from industrial robots**

Pokyny pro vypracování:

1. Navrhněte způsob sběru dat z průmyslových robotů KUKA přes rozhraní PROFINET a pomocí Ethernet KRL. Prozkoumejte možnosti synchronizace času v robotech např. pomocí NTP a implementujte přidělování časových známek ke každému rámci odeslanému z robota. Porovnejte obě komunikační rozhraní co se týká průchodnosti, rozptylu a případně navrhněte další ukazatele.
2. Navrhněte a implementujte pohyby robota, aby bylo možné sledovat dlouhodobou stabilitu, resp. odchylky v chování na základě analýzy získaných časových řad. Použijte jednak metodu zkoumání statistických momentů (viz [1]), jednak metodu založenou na skrytých Markovských modelech (viz [2]). Navrhněte klíčové ukazatele úspěchu použití uvedených metod.
3. S využitím předchozího bodu navrhněte diskrétní Markovský model chování robota, kde jednotlivé operace budou modelovány jako stavy. Půjde o model vyšší úrovně, který bude doplňovat spojitý model, který bere v úvahu jednotlivé pohybové veličiny.
4. Takto identifikované operace ukládejte do paměti a prozkoumejte možnosti využití metod Process Mining k vybudování diskrétního modelu. Porovnejte tento přístup s přístupem z předchozího bodu.

Seznam doporučené literatury:

[1] Novák, O. Robot diagnostics based on monitoring of its kinematic variables. Master thesis. CTU in Prague, Cybernetics and Robotics. 2019.
[2] Ron, M. Burget, P. Stochastic modelling and identification of industrial robots. In: Proceedings of the IEEE International Conference on Automation Science and Engineering. Piscataway, NJ: IEEE, 2016. p. 342-347. ISSN 2161-8070. ISBN 978-1-5090-2409-4.

Jméno a pracoviště vedoucí(ho) diplomové práce:

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

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **14.02.2019** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce:
**do konce letního semestru 2019/2020**

| | | |
|---|---|---|
| Ing. Pavel Burget, Ph.D. | prof. Ing. Michael Šebek, DrSc. | prof. Ing. Pavel Ripka, CSc. |
| podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | podpis děkana(ky) |

## III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

| | |
|---|---|
| . | |
| Datum převzetí zadání | Podpis studenta |

## Acknowledgements

I would like to thank my supervisor Ing. Pavel Burget, Ph.D. for his leadership, help and wise advice during the writing of this thesis. Also, I would like to thank to Ing. Martin Ron for his conslutation regarding the Markov Model part of this thesis.

I would also like to thank my family, my girlfriend and all of my friends for their support during the writing of this work and during my studies.

## Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in preparation of university theses.

In Prague, 24. May 2019

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

## Abstract

With the onset of the Fourth Industrial Revolution, many industrial enterprises are looking for methods, which reduce equipment maintenance costs. In this work, methods using the Ethernet KRL and industrial network PROFINET for the data collection on the industrial manipulator are described and evaluated. From these two methods, the method using the industrial network PROFINET and IIoT device Revolution PI has been selected and subsequently used to collect data from a manipulator that has assembled the structure using the LEGO bricks.

For the classification of the movement data, the methods using the statistical moments and time-varying hidden Markov model were used and tested. Furthermore, the third method was developed, using the hidden Markov classifier. This classifier has shown that it has better capabilities than previously developed methods. Furthermore, the method that uses a combination of discrete Markov model of robot behavior and a weak classifier is introduced and implemented. Tests have proved that this method can increase the accuracy of its classification.

Process Mining methods were then tested on the identified robotic operations, and the cases of using the process mining method were discussed.

**Keywords:** Predictive Maintenance, Machine Learning, IIoT, .NET Core, Markov Model, Classification, Process Mining, Manipulator, KUKA, KRL Code

**Supervisor:** Ing. Pavel Burget, Ph.D.

## Abstrakt

S nástupem čtvrté průmyslové revoluce, mnoho průmyslových podniků hledá metody, které by jim snížily náklady na údržbu zařízení. V této práci, jsou popsány a zhodnoceny dvě metody, využívající EthernetKRL nebo průmyslové sítě PROFINET, k sběru dat z průmyslového manipulátoru. Z těchto dvou metod, metoda využívající průmyslovou síť PROFINET a IIoT zařízení Revolution PI byla vybrána a následně použita pro sběr dat z manipulátoru, který skládal strukturu pomocí kostiček stavebnice LEGO.

Pro klasifikaci byli použity a otestovány metody využívající statistických momentů a metoda založena na časově proměnných skrytých Markovských modelech.

Dále byla implementována třetí metoda, využívají ke klasifikaci dat skrytý Markovský klasifikátor. Tento klasifikátor ukázal, že má lepší klasifikační schopnosti, nežli metody uvedené výše.

Dále je zde představena a implementována metoda, která využívá kombinaci diskrétního Markovského modelu chování robota a slabého klasifikátoru. Testy bylo dokázáno, že tato metoda je schopná zvýšit přesnost klasifikace slabého klasifikátoru.

Na identifikovaných robotických operacích pak byly vyzkoušeny metody Process Miningu a bylo diskutováno jejich použití.

**Klíčová slova:** Prediktivní Údržba, Strojové Učení, IIoT, .NET Core, Markov Model, Klasifikace, Proces Mining, Manipulátor, KUKA, KRL Kód

**Překlad názvu:** Sběr a analýza pohybových dat z průmyslových robotů

# Contents

# Figures

# Tables

# Chapter 1

# Introduction

With the onset of the Industrial Revolution 4.0, many companies and manufacturers are trying to find ways how to minimize their maintenance cost. There are five types of maintenance [1]: corrective, preventive, predictive, zero hours, periodic, but one, the predictive maintenance (PdM), uses the information gained from the machine [2].

The predictive maintenance techniques use state-of-the-art Machine Learning algorithms and Artificial intelligence. There can be found many papers, which aim at creating PdM frameworks (e.g. in [3]), discuss PdM techniques (e.g. in [4]) or implement PdM [5]. The PdM techniques create the model of machine behavior, which is supplied by data (time series, logs, etc.) from sensor. Based on data, the model decides if the machine is in fault or is going to be in fault.

One of the possible ways of modeling machine behavior is the usage of the hidden Markov model [6, 7, 8]. The approach presented in this thesis is based on the discussion in paper [9].

In this thesis, the studied system is an industrial robotic manipular. Firstly two possible methods of data acquisition will be introduced, developed and tested. Secondly, two already done methods of robotic movement classification will be presented and compared. Then a new framework for time series classification is developed. This framework is based on a hidden Markov model and it is used for robotic movement data classification.

In the end, the possibility of using the process mining techniques will be presented, analyzed and discussed.

## 1.1 Application

The outcome of this diploma thesis is developed .NET Core application called Data Acquisition Analysis. This application associates several libraries, each of them fulfills one part of the assignments. In figure 1.1 it can be seen how the libraries are connected. The libraries can be easily added to the framework, which is presented in [10].

## 1.2 Structure Of The Thesis

The thesis is separated into five chapters, where each chapter fulfills one guideline from the Thesis assignment:

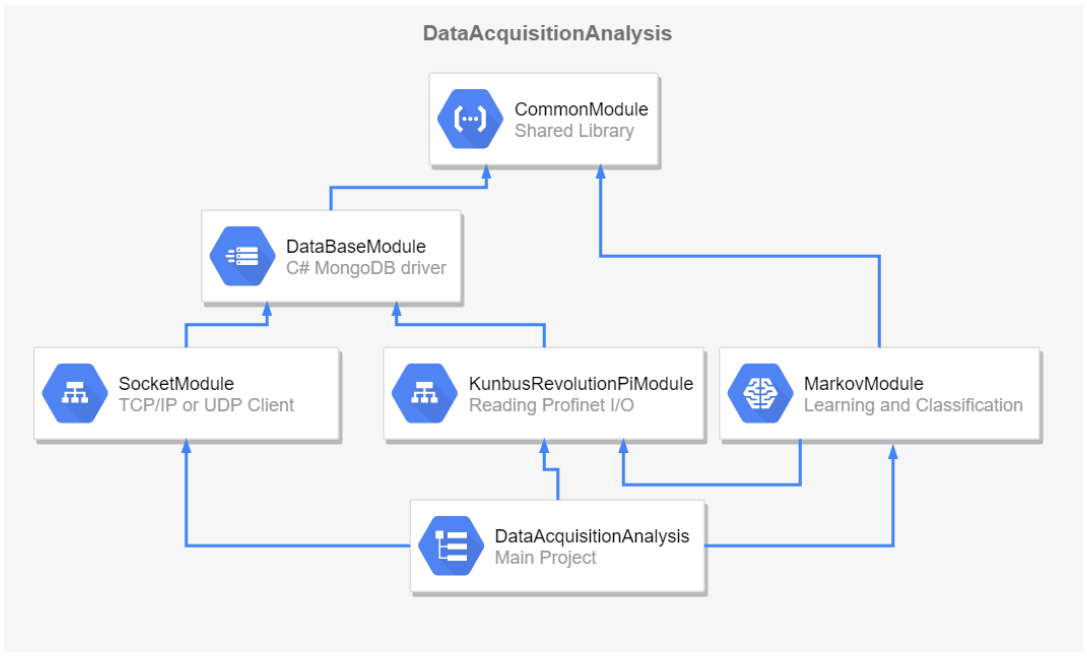


**Figure 1.1:** Application

1. *Data Acquisition* - Firstly the used robot on which the data acquisition will be performed is presented. Secondly, the new robotic assembly operation is created. Finally, the two acquisition methods are developed and compared.

2. *Comparation Of Allready Done Models For Robot Operation Identification* - Brief introduction of the done methods is followed by their comparison using measured data from the previous chapter.

3. *Markov Model Of Robot Behavior* - Creation of the Markov model of the robot behavior using the discrete-time Markov chain, which is completed by the continuous-time Hidden Markov Model Classifier. This model is then compared with models presented in the previous chapter.

4. *Process Mining* - Process Mining is introduced, and it is applied to the previously measured data.

5. *Conclusion*

# Chapter 2
# Data Acquisition

## 2.1 Workplace

The studied robots are from Testbed for Industry 4.0 at CIIRC CTU in Prague. The Testbed is a testing facility intended for small and middle size companies from the Czech Republic. Companies interested in automation and digital production can test there their new solutions according to the principles of Industry 4.0. Academic experts can help these companies check compatibility, functionality, and effectivity of their applications. If it is necessary, they can help companies with simulation and verification of their solutions and with the optimization of production processes.

## 2.2 Industrial Manipulators

In Testbed for Industry 4.0, two types of industrial manipulators are presented; cooperative and classical ones. The cooperative industrial manipulators are manipulators, which can't hurt the human operator. They are equipped with precise sensors which can detect collision with the operator and immediately stop their movement. The cooperative robots are in Testbed represented by KUKA LBR iiwa (see fig. 2.1). The classical industrial manipulators are not capable to cooperate with the human operator. Therefore they have to be closed behind the fence, or modern safety features have to be used (laser scanners, etc.). In this thesis, the focus on the classical manipulator will be taken.

### 2.2.1 Kuka KR Robots

Kuka company has two major series of robots, KR and LBR. LBR stands for Leichtbauroboter a.k.a. light construction robot. The flagship of this series is LBR iiwa who is a cooperative robot. KR stands for Kuka Robot. The main difference between KR and LBR series is that the LBR series robots are programmed in Java and KR is programmed in KRL (Kuka robotic language, see [11]).

Generally, every Kuka robot has three major parts. First one is the robot itself, the second is a robot controller, called KR C, and the last one is SmartPad.

The KR C is a specialized industrial PC in industrial case (called cabinet). The operating system in this PC is Windows, and on the basis of this system, the control

**Figure 2.1:** Kuka LBR iiwa 14

program is running. The cabinet has all necessary interfaces to communicate with the higher-level controller (PLC) and all robot peripheries, such as grippers and so on. The Kuka SmartPad is the main operator interface of the robot. It is a tablet-like device, on which the user can manipulate the robot and create basic programs.

### 2.2.2 Kuka KR10 1100 sixx

In this thesis, Kuka KR10 1100 sixx (Kuka KR10) is used as a studied system. The Kuka KR10 is industrial manipulator from Agilus family which is used for quick and precise manipulation of electronic devices manufacturing. The maximal payload of the robot is 10 kilograms, and its opearation space can be seen in picture 2.2.

#### Measured Data On Robot

The Kuka KR10 1100 Sixx has built-in internal sensors in each motor. The sensors measures motor variables, which can be accessed using the system program variables. All relevant motor variables are collected and stored. Namely: AXIS_ACT (actual robot position in joint space, measured in degrees), VEL_AXIS_ACT (actual velocity of each joint, measured in degrees per second), CURR_ACT (actual current in each motor in Ampers), MOT_TEMP (temperature in each motor, measured in Kelvins) and lastly TORQUE_AXIS_ACT (actual torque in $Nm$ which acts on the robot).

Along with these variables, the start and the end signals of each program were collected, because the data analysis presented in chapter 4 requires the beginning of the program and its end. Also, the program number is collected for learning the models, which will be presented later.

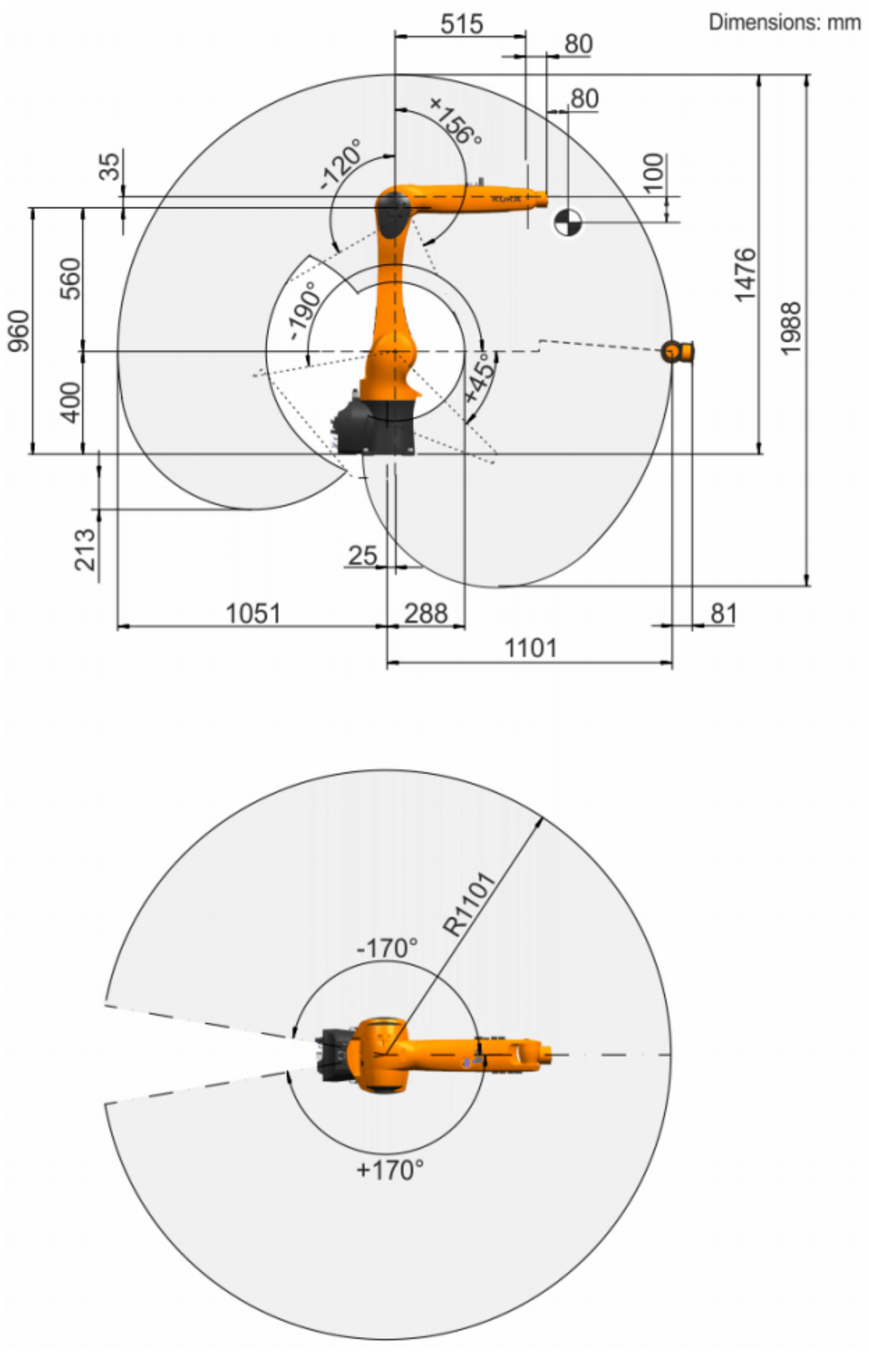


**Figure 2.2:** Kuka KR10 1100 sixx and its dimension and reachable area

## 2.3 MongoDB

MongoDB [12] is used in this project to store movement data of the robot. It is used for both data acquisition solutions (Ethernet KRL and PROFINET). MongoDB is a document-oriented "easy to use" database program, classified as a NoSQL database. MongoDB uses JSON-like documents, called BSON, for storing the data. BSON is a binary-encoded format of JSON, which provides additional data types, ordered fields and is efficient for encoding and decoding within different programing languages.

## 2.4 Data Acquisition Via Ethernet KRL

Ethernet KRL is an add-on technology package which allows data exchange between robotic manipulator and computer via TCP/IP protocol. There is a possibility of using another UDP protocol, but as it is said in [13], it is not recommended (connectionless network protocol, e.g. no packet loss detection).

However, the connection between the robot and the external PC is direct. Therefore the UDP packet lost should be minimal. The data acquisition is periodical on the application site with know period. So both protocol will be implemented, tested and their used will be discussed.

### 2.4.1 Implementation

In the picture 2.3, the architecture of the data acquisition via EthernetKRL is shown as well as used librabries from Data Acquisition Analysis application.

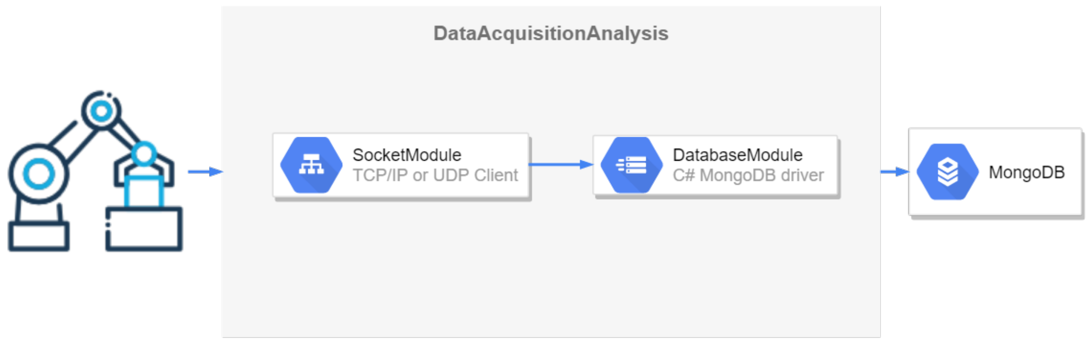


**Figure 2.3:** Data Acquisition with Ethernet KRL - process flow diagram

#### Robot Part

Kuka robots can be operated in two modes while using the Ethernet KRL, TCP/IP or UDP client or server. If the robot is in client mode, the connection to the server is possible only when the server is already running. If the server is not running the client cannot connect and the application failes. On the other hand, when the robot is in server mode, the application can run and await until the client connects.

The requirement is, that the acquisition would be a background task, which will not affect the execution of other programs. Therefore the variant of TCP/IP or UDP server was chosen.

The type of the server (the protocol) is selected in the configuration XML file of Ethernet KRL in the section called *Protocol*. For more information see [13].

The robot and external PC can exchange data in XML message or binary format. In this thesis, the XML message exchange between robot and PC is chosen. It is due to easy implementation on the robot site.

As in [10], the decision of using the Submit interpreter was made. The submit interpreter is a program, which runs aside (as a background task) the main robotic program, so it doesn't have any effect on it. This program runs in an infinite loop and shares resources with the main program on a lower priority. The Submit interpreter has some parts, which are necessary for running the main program. In Kuka KSS 8.5 it is possible to have more Submit interpreters (SPS.sub). Therefore the whole data acquisition via Ethernet KRL is implemented as new submit interpreter, so it doesn't have any effects on the main program.

As it was mentioned above, the server mode of Ethernet KRL was chosen. In this mode, when the program is started, the command EKI_Open() sets the server in the listening mode and awaits the client requests for connection. On Kuka, it is only possible to have one client connected to the server.

When all measured variables are serialized, the packet is sent to the external PC with command EKI_Send().

### PC Part

Data, which are sent from the robot, are processed on the PC with the MongoDB database. On this PC, Data Acquisition Analysis application in TCP/IP or UDP client mode is started. This mode can be initiated with the input argument: *socket*. In this mode, the client is started, and after a successful connection to the robot server, the data acquisition loop begins. Application parses data stream from the robot and saves each packet (one measurement) to MongoDB using the C# driver.

## 2.5 Data Acquisition Via PROFINET

In this section, the second method of data acquisition will be presented. This method uses an industrial network called PROFINET and has some common part with the solution shown in [10]. It can be seen in figure 2.4, which parts from Data Acquisition Analysis are used during the data acquisition via PROFINET.

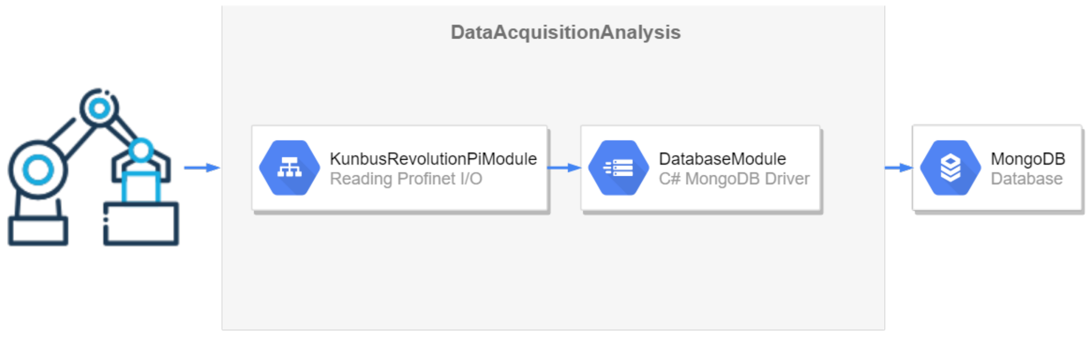


**Figure 2.4:** Data Acquisition with PROFINET - process flow diagram

### 2.5.1 What Is PROFINET?

The PROFINET is an industrial network standard which uses the link layer of Ethernet network as described in standard IEEE 802.3. It is capable of real-time, and non-real-time communication and data can be transferred both synchronously and asynchronously.

There are two main types of PROFINET devices. IO Controller and IO Device. IO Controller controls the automation task. IO Device is controlled and monitored by the IO Controller. Data are exchanged in periodical cycles in which IO Controller reads inputs or writes to the outputs, which are inputs of all its IO Devices. IO Controller has an image of IO Device inputs and outputs (I/O), which are mapped to user addresses space of the IO Controller. They are then available to the user in the applications (e.g. TIA Portal).

#### Kuka Robots

The Kuka robots are IO-Device for main hlligher-level or central controer. However, they can work as IO-Controller so they can control their peripherals such as grippers, clamps or unbus Revolution PI (see next section). The higher-level controller doesn't have IO Devices of Kuka robots in its hardware configuration.

### 2.5.2 Kunbus - Revolution PI

Revolution Pi by Kunbus is an industrial PC based on well-known Raspberry Pi. Specifically, the Revolution Pi Connect device is used. It is an open source IIoT (Industrial Internet of Things) gateway. This device is based on the Raspberry Pi Compute Module 3 contains a 1.2 GHz quad-core processor with 1 GB RAM and 4 GB eMMC flash memory. The used operating system is modified Raspbian with the real-time patch. This device support common IIoT protocols like MQTT or OPC UA [14].

To the Revolution Pi, additional modules can be connected. Modules are used, when it is necessary to integrate Revolution Pi to the industrial network, such as PROFINET, PROFIBUS, etc. Additional modules are connected to the Revolution Core via overhead PI bridge module. Data exchange between the module and RevPi

Core takes place every $5ms$. The mentioned frequency is sufficient for our application. With this module, Revolution Pi is seen as the next IO Device in the industrial network configuration.

In this thesis, the Revolution Pi is used as Edge PC, which is connected as IO Device to the PROFINET network of the facility as well as to the Internet (in our case to LAN of the laboratory). Edge PCs are commonly used for collecting the data and sending them to the cloud or database.

### 2.5.3 I/O Mapping

Before data acquisition, it is necessary to map inputs and outputs from robot to Revolution Pi. For this purpose, the Work Visual program was used. The Work Visual is a program by Kuka, which is used for setting up Kuka KR robots and it provides a programming interface for robots. It allows user debugging of the robot program and its deployment to the robot.

Kuka KR robots have 4064 bits of digital inputs and outputs. First 2032 bits are used for communication with a higher-level controller (PLC). Therefore the remaining bits can be used for our application. Revolution Pi Profinet module can map up to 512 Bytes for both inputs and outputs.

For every measured variable (see sub-section 2.2.2) 32-bit value has to be used. Beside of measured the variables, the time stamp from the robot has to be collected as well as the start and the end of the program and the program number. These variables have one Byte size. Overall 1032 bits are used on the robot site.

During the mapping, last 1032 bits from 4064 robot bits are used. On the Revolution Pi site first 1032 bits are used. The example of mapped I/O can be seen in figure 2.5.

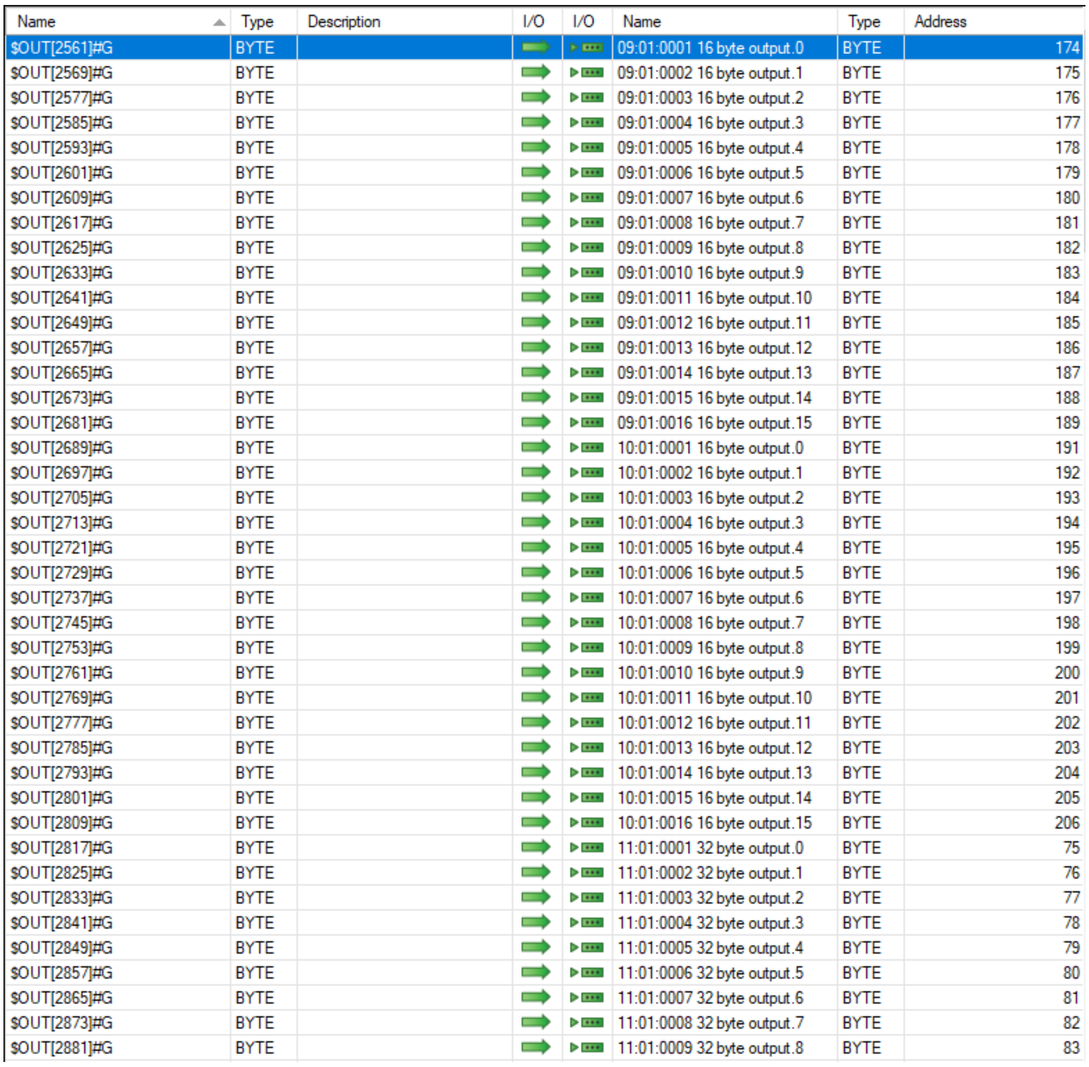
### 2.5.4 Data Acquisition Loop

#### Revolution Pi Part

C libraries, which allow reading and writing to Revolution Pi PROFINET inputs and outputs are written in C language. However, Data Acquisition Analysis is written in .NET Core using the C# language. Therefore the Shared library was used, which will be called by C# code during the runtime.

In C libraries there are many functions which can be used. However, three of them are called by C# code. First one piControlOpen() is called on the start of the data acquisition process. This function initializes the Pi Control Interface. Second one piControlClose(), is called when the program is terminated, and it is used for closing the Pi Control Interface. And the last function, piControlRead() is called during the run-time of the application. This function enables to read Revolution Pi inputs (robot outputs). The input parameters of this function are byte offset, length of reading bytes and pointer to the byte array, where data will be saved. This array is then converted to float or int.

The configuration of reading inputs on Revolution Pi is determined by a configuration JSON file. This file enables quick and easy setup of data acquisition. A user can specify which variable can be found on which Revolution Pi inputs (offset) and how



| Name | Type | Description | I/O | I/O | Name | Type | Address |
|---|---|---|---|---|---|---|---|
| $OUT[2561]#G | BYTE | | ➡ | ▶ | 09:01:0001 16 byte output.0 | BYTE | 174 |
| $OUT[2569]#G | BYTE | | ➡ | ▶ | 09:01:0002 16 byte output.1 | BYTE | 175 |
| $OUT[2577]#G | BYTE | | ➡ | ▶ | 09:01:0003 16 byte output.2 | BYTE | 176 |
| $OUT[2585]#G | BYTE | | ➡ | ▶ | 09:01:0004 16 byte output.3 | BYTE | 177 |
| $OUT[2593]#G | BYTE | | ➡ | ▶ | 09:01:0005 16 byte output.4 | BYTE | 178 |
| $OUT[2601]#G | BYTE | | ➡ | ▶ | 09:01:0006 16 byte output.5 | BYTE | 179 |
| $OUT[2609]#G | BYTE | | ➡ | ▶ | 09:01:0007 16 byte output.6 | BYTE | 180 |
| $OUT[2617]#G | BYTE | | ➡ | ▶ | 09:01:0008 16 byte output.7 | BYTE | 181 |
| $OUT[2625]#G | BYTE | | ➡ | ▶ | 09:01:0009 16 byte output.8 | BYTE | 182 |
| $OUT[2633]#G | BYTE | | ➡ | ▶ | 09:01:0010 16 byte output.9 | BYTE | 183 |
| $OUT[2641]#G | BYTE | | ➡ | ▶ | 09:01:0011 16 byte output.10 | BYTE | 184 |
| $OUT[2649]#G | BYTE | | ➡ | ▶ | 09:01:0012 16 byte output.11 | BYTE | 185 |
| $OUT[2657]#G | BYTE | | ➡ | ▶ | 09:01:0013 16 byte output.12 | BYTE | 186 |
| $OUT[2665]#G | BYTE | | ➡ | ▶ | 09:01:0014 16 byte output.13 | BYTE | 187 |
| $OUT[2673]#G | BYTE | | ➡ | ▶ | 09:01:0015 16 byte output.14 | BYTE | 188 |
| $OUT[2681]#G | BYTE | | ➡ | ▶ | 09:01:0016 16 byte output.15 | BYTE | 189 |
| $OUT[2689]#G | BYTE | | ➡ | ▶ | 10:01:0001 16 byte output.0 | BYTE | 191 |
| $OUT[2697]#G | BYTE | | ➡ | ▶ | 10:01:0002 16 byte output.1 | BYTE | 192 |
| $OUT[2705]#G | BYTE | | ➡ | ▶ | 10:01:0003 16 byte output.2 | BYTE | 193 |
| $OUT[2713]#G | BYTE | | ➡ | ▶ | 10:01:0004 16 byte output.3 | BYTE | 194 |
| $OUT[2721]#G | BYTE | | ➡ | ▶ | 10:01:0005 16 byte output.4 | BYTE | 195 |
| $OUT[2729]#G | BYTE | | ➡ | ▶ | 10:01:0006 16 byte output.5 | BYTE | 196 |
| $OUT[2737]#G | BYTE | | ➡ | ▶ | 10:01:0007 16 byte output.6 | BYTE | 197 |
| $OUT[2745]#G | BYTE | | ➡ | ▶ | 10:01:0008 16 byte output.7 | BYTE | 198 |
| $OUT[2753]#G | BYTE | | ➡ | ▶ | 10:01:0009 16 byte output.8 | BYTE | 199 |
| $OUT[2761]#G | BYTE | | ➡ | ▶ | 10:01:0010 16 byte output.9 | BYTE | 200 |
| $OUT[2769]#G | BYTE | | ➡ | ▶ | 10:01:0011 16 byte output.10 | BYTE | 201 |
| $OUT[2777]#G | BYTE | | ➡ | ▶ | 10:01:0012 16 byte output.11 | BYTE | 202 |
| $OUT[2785]#G | BYTE | | ➡ | ▶ | 10:01:0013 16 byte output.12 | BYTE | 203 |
| $OUT[2793]#G | BYTE | | ➡ | ▶ | 10:01:0014 16 byte output.13 | BYTE | 204 |
| $OUT[2801]#G | BYTE | | ➡ | ▶ | 10:01:0015 16 byte output.14 | BYTE | 205 |
| $OUT[2809]#G | BYTE | | ➡ | ▶ | 10:01:0016 16 byte output.15 | BYTE | 206 |
| $OUT[2817]#G | BYTE | | ➡ | ▶ | 11:01:0001 32 byte output.0 | BYTE | 75 |
| $OUT[2825]#G | BYTE | | ➡ | ▶ | 11:01:0002 32 byte output.1 | BYTE | 76 |
| $OUT[2833]#G | BYTE | | ➡ | ▶ | 11:01:0003 32 byte output.2 | BYTE | 77 |
| $OUT[2841]#G | BYTE | | ➡ | ▶ | 11:01:0004 32 byte output.3 | BYTE | 78 |
| $OUT[2849]#G | BYTE | | ➡ | ▶ | 11:01:0005 32 byte output.4 | BYTE | 79 |
| $OUT[2857]#G | BYTE | | ➡ | ▶ | 11:01:0006 32 byte output.5 | BYTE | 80 |
| $OUT[2865]#G | BYTE | | ➡ | ▶ | 11:01:0007 32 byte output.6 | BYTE | 81 |
| $OUT[2873]#G | BYTE | | ➡ | ▶ | 11:01:0008 32 byte output.7 | BYTE | 82 |
| $OUT[2881]#G | BYTE | | ➡ | ▶ | 11:01:0009 32 byte output.8 | BYTE | 83 |

**Figure 2.5:** I/O Mapping of the robots outputs and Revolution PI inputs

many bytes will be read. The advantage of this configuration file is that the user can easily set the same representation of data in Mongo DB as well as in the program.

The application allows user to specify, how often the Revolution Pi inputs will be read. The period of reading the inputs should be larger than the mentioned communication cycle between PROFINET module and RevPi Core ($5ms$). In subsection 2.7.1, this period is found.

During the acquisition loop, the data are sent to the MongoDB using the .NET Core MongoDB driver. Every sent data have two timestamps. One is the time when the data were written to robot outputs. The second one is the time when data were read on the Revolution Pi inputs.

### Robot Part

On the side of the robot, the similar interface presented as in [10] was used. Kuka robots allow using a simplified representation of I/O variables. For this purpose, the variable of the type SIGNAL has to be used. This type allows selecting a bit range of robot inputs/outputs and saves variables into these signals, without any conversion. The user has only been aware of the bit size of variables. For example, the range has to have 32 bits in length. As in [10], the *ciirc_io.dat* file was used for the declaration of the measured variables signals. Like in section 2.4, new Submit interpreter was created for this data acquisition. When the Submit interpreter is called, it writes desired variables into its signals. Immediately after that, data are propagated into robot digital outputs.

## 2.6 Synchronization Of Time

To determine the same time on each device involved in data acquisition, the computer on which MongoDB is installed was set up as an NTP server. NTP (Network Time Protocol) is the protocol for synchronization of clocks inside computers. This protocol enables all computers in the network to have the same accurate time. This protocol is using a UDP packet at port 123. The global time accuracy, i.e. NTP server synchronized to the exact clock is not needed. Nevertheless, it is enough that all devices have the same time on the scale of the local network, and the accuracy of the NTP server on the MongoDB PC is then sufficient.

The NTP server and client are by default disabled on Windows computers. NTP server and client are on Windows implemented by W32Time service. The service has to be enabled in Windows Services and configured into Automatic start. After that, a modification in registers is necessary. In register path:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services
\W32Time\TimeProviders\NtpServer,
```

the variable Enabled has to be changed from 0 to 1. After that, the Windows Time service's has to be updated. It is done by calling this command `w32tm/config/update` in a command prompt (cmd). We can check, that everything is set up correctly by typing `w32tm/query/configuration`. With this procedure, the NTP server on the database computer is enabled and running.

As it was mention in section 2.2, whole KRC runs on Windows machine. Therefore we can set up automatic time synchronization of the robot from the NTP server. Hence, the robot will be an NTP client. This can be done in Date and Time settings. There is an option of setting the NTP server IP as a reference. After setting up, the time on the robot is synchronized with time on database computer.

The NTP package has to be installed for the synchronization of Revolution Pi device with NTP server. After that, the `/etc/ntp.conf` file has to be edit. Default settings were deleted and the following line: `server 10.35.91.210 prefer` was inserted. After that, the NTP Deamon was started and time on Revolution Pi was synchronized with the database computer.

As it was said in previous sections 2.4 and 2.5, when each packet with data is sent from the robot, the current timestamp is added into that frame as additional information into data.

## 2.7 Comparation Of Acquisition Technique

Both presented acquisition methods were tested under the same environmental conditions and robot setup. During the acquisition, the robot performs a basic movement.

### 2.7.1 PROFINET

In the picture 2.6 the measurement of the robot trajectory can be seen, which was done by the Revolution Pi (see section 2.5). The method is capable of measuring the fast changes in the robot position. The only challenge is finding the optimal sampling period in the application. It is desirable not to have oversampled data.

The Shannon theorem was used to determine the optimal sampling period. The Shannon theorem says that the sampling period has to correspond to the sampling frequency $f_s > 2f_{max}[s^{-1}]$ of the maximal frequency of the system. The robot controller writes the robot measured values to the outputs every $12ms$, so the sample period in the application should be $6ms$ based on Shannon theorem. The values are held on the robot outputs for the period of $12ms$. Therefore the robot works as the zero-order hold model (ZOH). The data are oversampled, so the measurement has redundant samples.

Since the synchronization between the robot and Revolution Pi is not guaranteed, the PROFINET communication cycle should remain $6ms$. This will lead to oversampling data, so the preprocessing of measured data has to be used. The preprocessing is done by the Moving Average filter.

#### Moving Average Filter

Moving average is a method, which creates the mean over the sliding window of the length $k$ across nearby elements from the original data set. The mean is calculated as:

$$\mu_t = \frac{1}{k}\sum_{i=1}^{k} x_i, \tag{2.1}$$

where $x_i$ is an element from the original dataset. The result of the Moving average filter can be seen in figure 2.7.

#### Measuring of PROFINET Communication

Also beside the experiments, the measurement of the PROFINET communication was performed. For the measurement, the ProfiTap device, which can capture the PROFINET communication over the Ethernet cable, was used. The ProfiTap was connected right in front of the measured robot. This device was then capable of sniffing the PROFINET communication and of streaming it directly into the connected computer. On the computer, the Wireshark program was running. The Wireshark is a protocol analyzer and a packet sniffer.

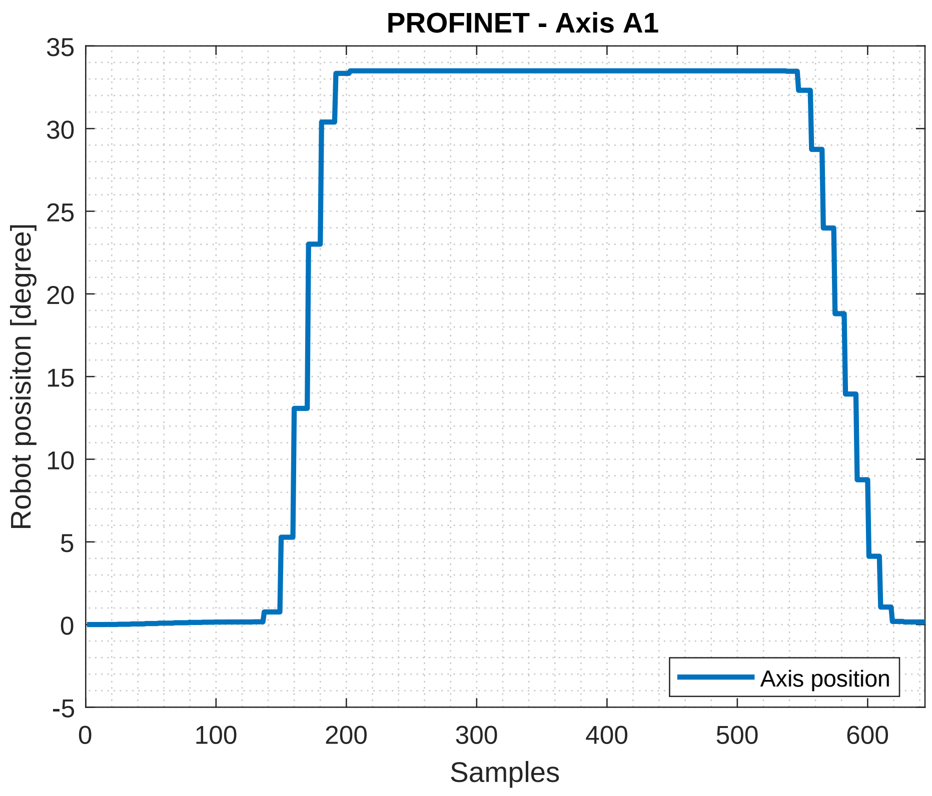


**Figure 2.6:** Mesurement on axis A1 by PROFINET

For the captured communication, the cycle communication of the PROFINET between the robot and Revolution Pi was $2.03ms$. However, it was measured that the value changes every $12ms$ (ZOH). Therefore, the reading of Revolution Pi inputs in the application was set to every $6ms$.

With found period, the measurement of the data was performed. The amount of the sent packet per second on Revolution Pi was equal to 166.67 (computed from 389 922 packets). The size of each packet $208B$ was sniffed with the Wireshark. With the size of the packet and the amount of packet sent per second, the throughput of the PROFINET network was calculated like $166.67 \times 208 = 34\ 666.67B/s = 277\ 333.34b/s$.

## Network Jitter

The network jitter is a variation in the time delay between the arrival of data packets to the destination PC. To calculate jitter, the unbiased sample variance is used. The original data sniffed by the ProfiTap will be used for the calculation of the PROFINET jitter. The resulting PROFINET jitter is equal to $0.013ms$.

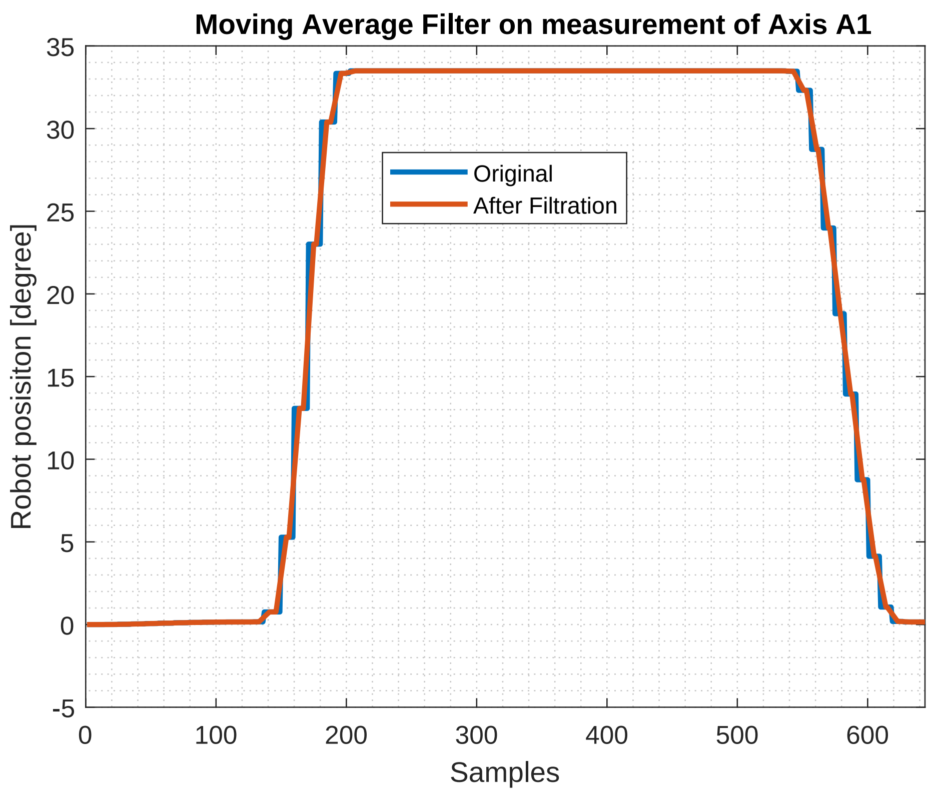


**Figure 2.7:** Application of Moving average filter on the data set.

### 2.7.2 Ethernet KRL - TCP/IP

The measurement via Ethernet KRL using the TCP/IP can be seen in the picture 2.8. The resulting trajectory of axis A1 is much more undersampled, then it was in the previous section.

Therefore, a deeper analysis was made. For the analysis of the Ethernet communication, the Wireshark application was used again.

The analysis consisted of three experiments during the amount of sent data was reduced to find out if the amount of data have any influence on the transmission speed.

First, the normal amount of data was sent from the robot to the database PC. In figure 2.9 the throughput analysis of the communication between the robot and database PC can be seen. The data are sent with an average period of 250.9$ms$ with network jitter 6.24$ms$. Then the experiments, during which the amount of sent data were reduce was performed.

Overall, three experiments were performed. In the first one, the measurement of the motor torques was not included in the data. In the second experiment, the amount of sent data was reduced by motor torques and temperatures. In the last experiment, only two motor variables (twelve values) were sent from the robot to the external PC. The results of the experiments can be seen in table 2.1, where the increase in the transmission speed is noticeable. However, the network jitter increased too. It has to

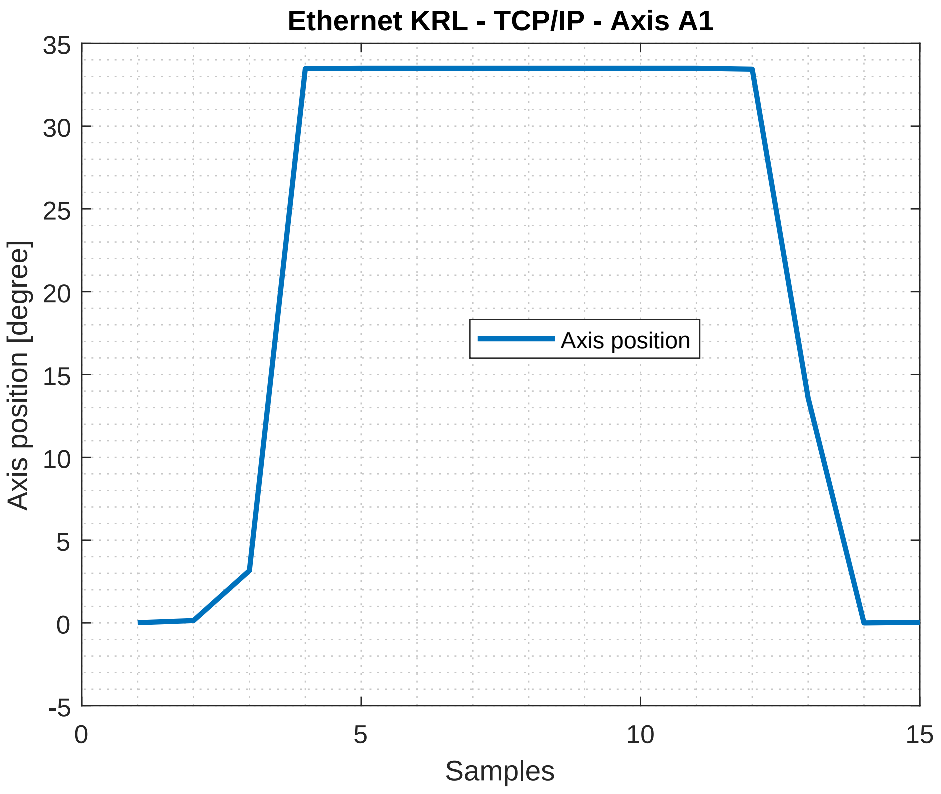


**Figure 2.8:** Mesurement on axis A1 by Ethernet KRL - TCP/IP protocol

be noted that the current network traffic in Testbed is low. If the traffic was higher, the slower transmissions speeds are expected.

Then the Round Trip Time of TCP/IP packet was measured. In the figure 2.10 the Round Trip Time (RTT) graph of the TCP/IP packet can be seen. The RTT is time how long it takes to send one packet plus how long it takes to an acknowledgment of that packet to be received. The average RTT time in all experiments is in the average $40.5ms$.

The throughput of the EthernetKRL using the TCP/IP communication with full measured data from the robot was 21 $613b/s$.

| **Number of sent variables** | **Avg. Period** $[ms]$ | **Network Jitter** $[ms]$ |
|---|---|---|
| 24 | 208.18 | 26.95 |
| 18 | 166.35 | 17.00 |
| 12 | 126.10 | 14.64 |

**Table 2.1:** TCP/IP communication experiment

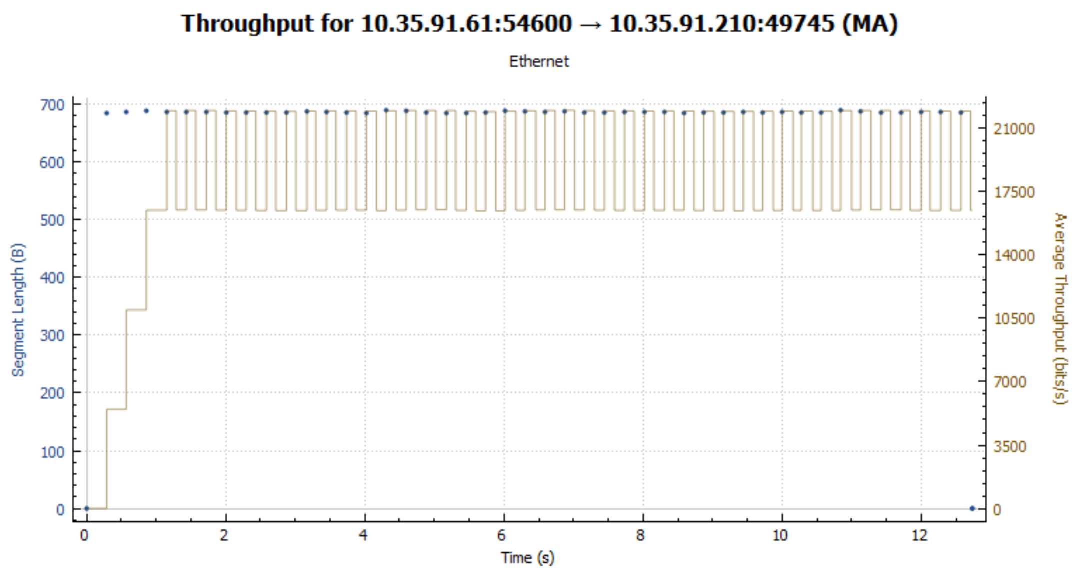


**Figure 2.9:** TCP/IP communication via Ethernet KRL - Experiment 1

## 2.7.3 Ethernet KRL - UDP

The measurement via Ethernet KRL using the UDP protocol can be seen in figure 2.11. Like in previous subsection 2.7.2, the measurement is undersampled in comparison with PROFINET data acquisition.

The Wireshark enables to show the time delay of current UDP packet from the previous packet. Using this information, the average period of the packet arrival was determined as $294.38ms$ with network jitter $6.44ms$. Therefore, the same experiments as in 2.7.2 were performed.

The results of the experiments can be seen in table 2.2. From the realized experiments, it can be concluded that the main delay of the communication between the robot and the external PC is on the robot side (serialization of the XML message, implementation of the communication, etc.). The measured throughput of the UDP communication with full measured data was $3.39 \times 726 = 2\,462.43B/s = 19\,699.49b/s$ with the period $294.83ms$.

| Number of sent variables | Avg. Period $[ms]$ | Network Jitter $[ms]$ |
|---|---|---|
| 24 | 245.93 | 4.76 |
| 18 | 200.55 | 4.68 |
| 12 | 167.83 | 4.65 |

**Table 2.2:** UDP communication experiment

## 2.7.4 Conclusion

The Ethernet KRL wasn't created for real-time communication as PROFINET. It was primarily designed for sending the data from/to the robot (like program numbers or

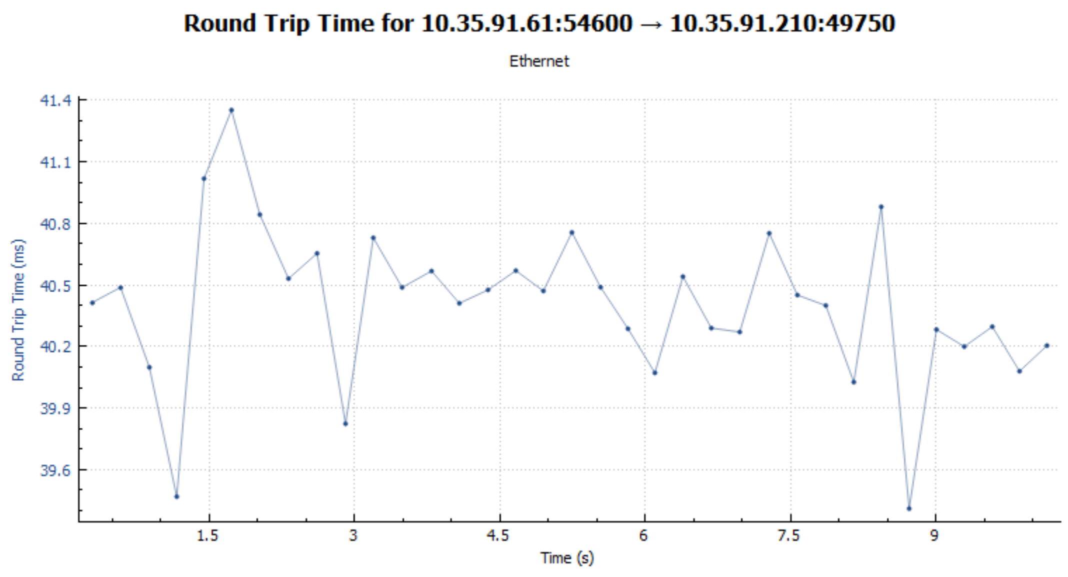


**Figure 2.10:** TCP/IP communication via Ethernet KRL - Round Trip Time (RTT)

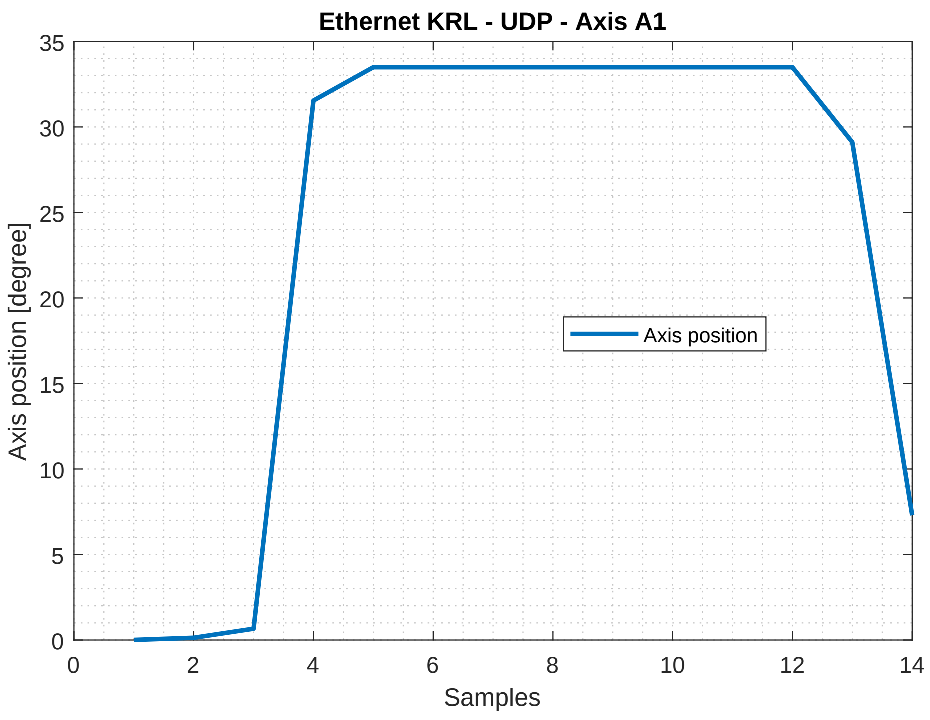


**Figure 2.11:** Mesurement on axis A1 by Ethernet KRL - UDP protocol

positions to which robot has reached). Also, the PROFINET has much better results than the Ethernet KRL in the measured throughput and jitter. Therefore, for further experiments and measurements, the PROFINET will be used.

If only available communication was Ethernet KRL, the most capable protocol for the sending of data is UDP. The TCP/IP has better results, but the network communication in Testbed is not overloaded, so the results don't reflect real-life industrial conditions. If network traffic was higher, the average period of TCP/IP would be bigger than in the case of UDP. Also, the UDP protocol communication doesn't overload the network traffic as TCP/IP (due to the acknowledgment of the message).

It has been shown that network jitter is not an essential parameter of the acquisition methods comparison. The network jitter is lower compared with the period of communication.

## 2.8 Robotic Program

For the testing of the robustness of the algorithms, a robotic assembly operation was created. The product which is assembled by the robot is a LEGO structure (see figure 2.13). The assembly contains ten pick operations and ten place operations which are very similar in the Cartesian positions. One operation is one robotic program, called by the higher controller during the assembly process. The bricks (see figure 2.12 for the assembly are stored in the warehouse (see the configuration in figure 2.14). These bricks have identical $Z$ Cartesian position. However, they have different $X$ and $Y$ positions. The distances between the bricks are the same as the width of the fingers of the gripper (minimal possible).

After the pick operation, the robot moves above the place area and place brick to desired positions, so the structure from the figure 2.13 is assembled. Besides the pick and place operations, there are two service operations for the tool changing, which are called in the begging and at the end of the assembly operation.

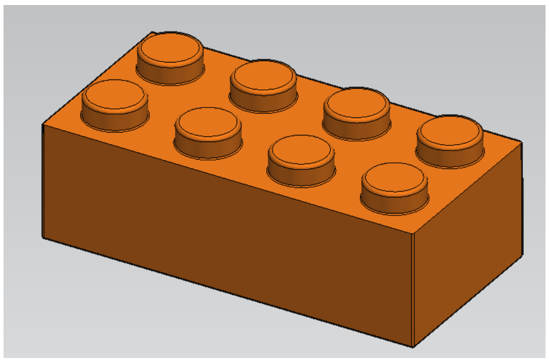


**Figure 2.12:** LEGO brick

To speed up the creation of the robotic program, Process Simulate was used. The Process Simulate is a verification and a simulation program by Siemens PLM company. This program is widely used in industry for offline programming of the robotic manipulators and simultaneously as a key part of the process called virtual committing. This process is used for testing of the PLCs programs with simulated robotic manipulators. During this process, it is possible to catch all bugs in the PLC program as well as in the robotic program.

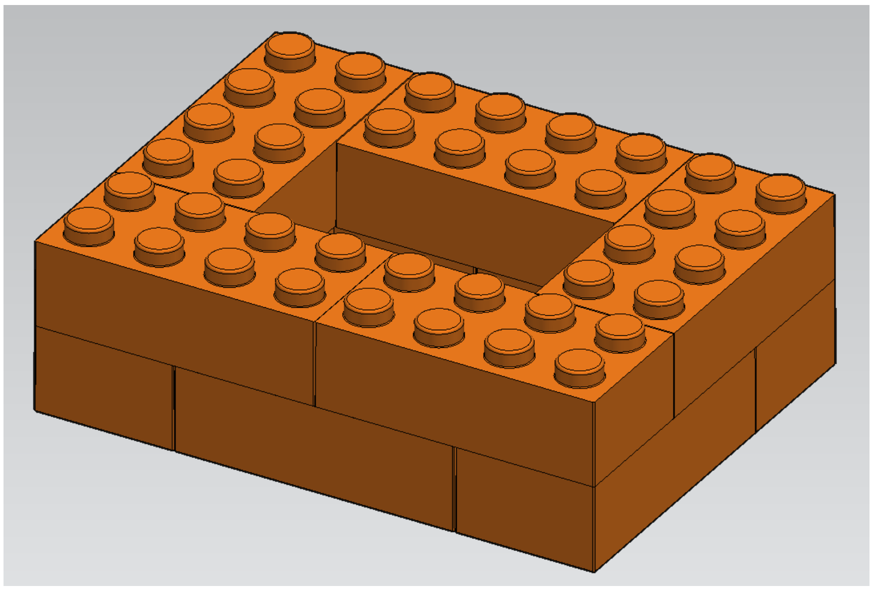


**Figure 2.13:** LEGO Assembly

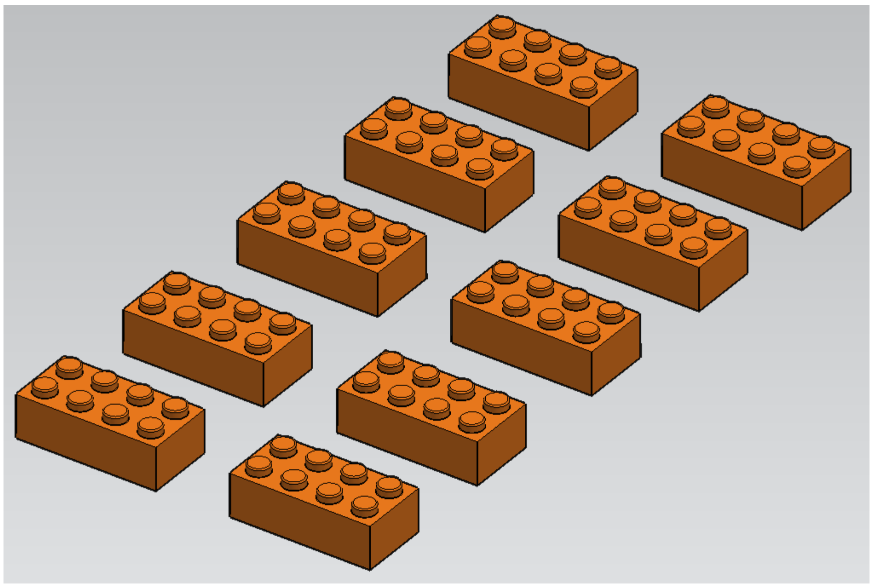


**Figure 2.14:** LEGO bricks before assembly

For the testing of the performance of the method, which will be presented lately, 63 assembly operations were measured on the robot. The measurement using the PROFINET was used for the data acquisition. Besides the normal behavior of the robot, the corrupted data was measured too.

To simulated the damaged robot, the $2.7Kg$ weight was mounted on the robot end effector. In the measured data (called corrupted data), the current should have a different process, and also the torque should be changed.

The corrupted data set was measured for testing how the methods are capable of detecting the fault on the robot.

## 2.9 Chapter Overview

In this chapter, the workplace was introduced as well as the studied robot. On the studied robot all relevant measured variables were introduced.

Then two methods of data acquisition were presented. The methods were implemented, and the pross and cons of both methods were shown. After the analysis, the method which is using PROFINET communication was chosen for its reliability and the fact that the PROFINET is real-time technology.

After the test of both methods, the assembly operation of the LEGO structure was created for the testing of a further method for operation classification and fault detection.

At the end of the chapter, the possible way for time synchronization was presented.

Chapter 3

# Comparation Of Allready Done Models For Robot Operation Identification

In this chapter, the methods for robot operation identification, which were created at CTU in Prague, are compared. Namely the method using the statistical moments and the method using stochastic modeling.

## 3.1 Statistical Moments

In [10], the method of using the statistical moments for robot operation identification was presented. Namely, it used first, second and third statistical moments:

- First moment - mean: $\mu = E(X)$,
- Second moment - variance: $\sigma^2 = E(X - \mu)^2$,
- Third moment - skewness: $skew = E(X - \mu)^3$,

where $X = \{x_1, x_2, \ldots, x_n\}$.

From the measured data, elements were picked, which were selected in [10] as the most important features. On these data, the statistical moments for each operation were computed. Then the data were sent to the Azure cloud for the training of the model using the framework created in [10]. In the table 3.1 results on the training set and the testing set using the cross-validation folds and K-means classification can be seen.

The cross-validation method is a method of splitting the data into the training and testing subsets. After splitting the set of data, the method trains the model on training subset and evaluates the model error on the testing subset. K-fold cross-validation splits data into k-folds (k is usually 5 or 10) and in each iteration uses $k - 1$ folds for testing and rest for evaluation of the accuracy. After the iterations, it aggregates the $k$ error measurements to get the final error estimate. The method presented in [10] uses three folds.

$K$-means is a clustering method which aims to split $n$ observation into $k$ clusters. Each observation belongs to a cluster with nearest means.

The method presented in [10] also uses the dimensionality reduction of the input data. Dimensionality reduction helps increase the speed of the classification with the reduction of the data dimension. Namely, it used the Feature Agglomeration algorithm.

As it is said in [10], this algorithm uses two inputs parameters, data to be reduced and the desired dimension.

| Folds Split | Full data accuracy | Accuracy after dimension reduction |
|---|---|---|
| 1 | 0.7232 | 0.5667 |
| 2 | 0.7144 | 0.5927 |
| 3 | 0.7329 | 0.5927 |

**Table 3.1:** k-Fold Cross-validation on measured data and the accuracy

As can be seen, the accuracy after the dimension reduction is not as good as before it. The characteristic of the robotic operation is a reason for the worse score. The pick and place operations, which are performed on the robot, are very similar. So when the dimension reduction is performed, some information about the operation could be lost. Also, the use of statistical moments could lead to the worse results. The whole operation is modeled only by one Gaussian, and when there are two very similar operations, their Gaussians could be similar. Therefore the $K$-means classification, which is used is not capable of determining the correct operation.

## 3.2 Stochastic Modeling Using Time-Varying Markov Model

The second method developed at the Department of Control Engineering at FEE at CTU is using the Time-Varying Markov Model for operation identification (classification) [9]. This method trains the time-varying Markov model using the frequency of occurrences of transitions as an approximate probability of transition during the training. As it is said in [9], this method uses the Gaussian distribution in the time-varying Markov model. The whole derivation can be found in [9].

The same measured data as in the previous case were used. Firstly the time-varying Markov model was trained. Then the model was tested on the test data set. The model has much better accuracy than the previous one. The accuracy was equal to 100%. The main reason for such accuracy is that the positional data were used. Positional data have enough information for the determination of operations. In [9], the accuracy is lower, due to the use of energy consumption data.

In figure 3.1 the evaluation of the log likelihoods of the selected classes can be seen. Each class belongs to one operation performed on the robot. It can be seen how the log likelihoods of each class are evolved with increasing time samples. Finally, the model classifies the input sequence as class 21, this classification is correct.

On the other hand, the classification of corrupted data has the misclassification ratio = 31%. In figure 3.2 the evaluation of log likelihood is ploted. The correct operation was operation 10, but the model classified it as operation 14.

The method models one time series by one Gaussian only and this can lead to such large misclassification. When the weight was mounted on the robot, some variables, such as speed, torque, and current changed in such a way, that the deviation of the nominal value leads to misclassification. However, if only a position data were used, there weren't any misclassifications. It has to be noted, that method presented in [9] is not made for fault detection. It is made for the classification of robotic operation during the normal run without any deviation.

The method, which will be presented in the next chapter, should have better results at corrupted data.

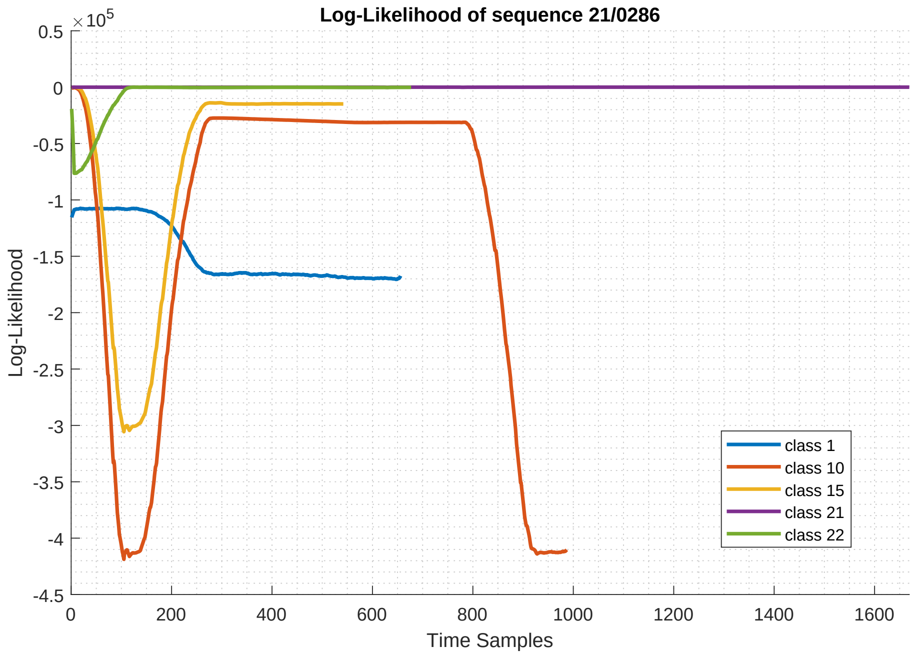


**Figure 3.1:** LogLikelihood of sequence - classification of operation 21

## 3.3 Chapter Overview

This chapter presented and briefly introduced two methods which were created in the past. Both methods were tested on the measured data of the robotic assembly operation shown in section 2.8.

The method presented in [10] could be used for the determination of the type of operation (pick or place). However, the method shown in [9] is ideal for the operation classification (resolution of the operation number).

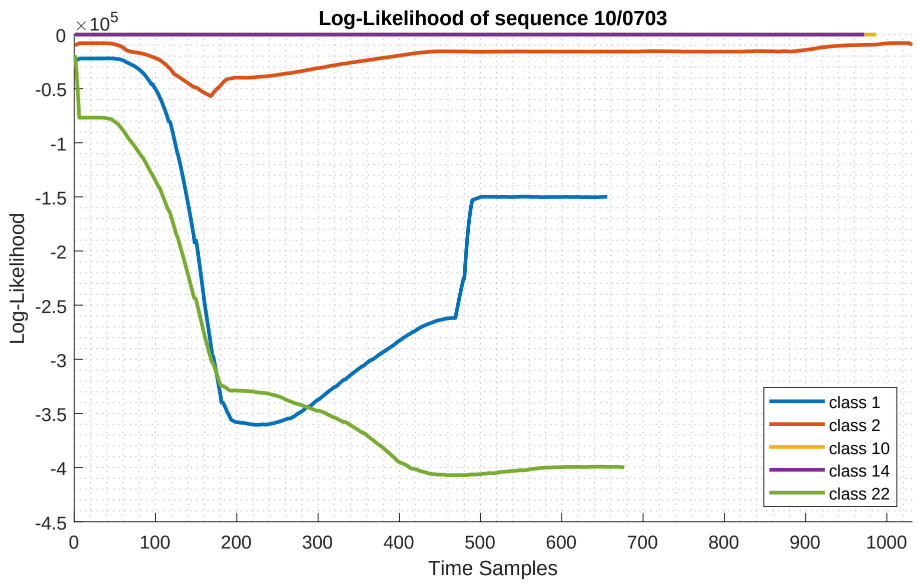


**Figure 3.2:** LogLikelihood of sequence - classification of operation 10

# Chapter 4

# Markov Model Of Robot Behavior

All parts, which will be presented in this chapter are implemented in the MarkovModule of Data Acquisition Analysis or DataProcessing python scripts (feature selections).

## 4.1 Discrete-Time Markov Chain

### 4.1.1 Markov Property

Let's have a stochastic system and its observation in time $0, 1, 2, \ldots, N$. Let's also have a state of the system in each observed time $X_n$. Suppose that the system is currently at time $n = 5$. Now we would like to predict the state of the system at time $n = 6$. Generally, the state $X_6$ depends on its previous states $X_0, X_1, \ldots, X_5$. Let's suppose that we have completed history $X_0, X_1, \ldots, X_5$. In this case, the future state depends only upon $X_5$. The knowledge of all states before $X_5$ is redundant if $X_5$ is known, all information is stored in state $X_5$. If a stochastic system has this property at any time $n$, it is said that the system has *Markov property*.

### 4.1.2 Markov Chain

As it is said in [15], a stochastic process on state space $S$ is said to be discrete-time Markov chain (DTMC), if for all $i$ and $j$ in $S$,

$$p(X_{n+1} = j | X_n = i, X_{n-1}, \ldots, X_0) = p(X_{n+1} = j | X_n = i). \tag{4.1}$$

A DTMC is said to be homogenous if, for all $n = 0, 1 \ldots,$

$$p(X_{n+1} = j | X_n = i) = P(X_1 = j | X_0 = i). \tag{4.2}$$

Equation 4.1 implies, when the present state of the system $X_n$ is given, the future state $X_{n+1}$ of the DTMC is independent of its past states $(X_0, X_1, \ldots, X_{n-1})$. The $P(X_{n+1} = j | X_n = i)$ is called a one-step transition probability of the DTMC at time $n$. The second equation says that if the probability of one-step transition on $i$ to $j$ is the same at all times $n$, then the DTMC is time invariant [15].

Let's use the following notation for one-step transition probability:

$$p_{i,j} = P(X_{n+1} = j | X_n = i), \; i, j = 1, 2, \ldots, N. \tag{4.3}$$

Note that there are $N^2$ one-step transition probabilities $p_{i,j}$. It is convenient to arrange them to $N \times N$ matrix form as shown below ([15]):

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,N} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,N} \\ p_{3,1} & p_{3,2} & p_{3,3} & \cdots & p_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{N,1} & p_{N,2} & p_{N,3} & \cdots & p_{N,N} \end{bmatrix} \tag{4.4}$$

The matrix P in the equation 4.4 is called the one-step transition probability matrix, or transition matrix [15]. Rows correspond to the starting state and columns to the ending states of the transition.

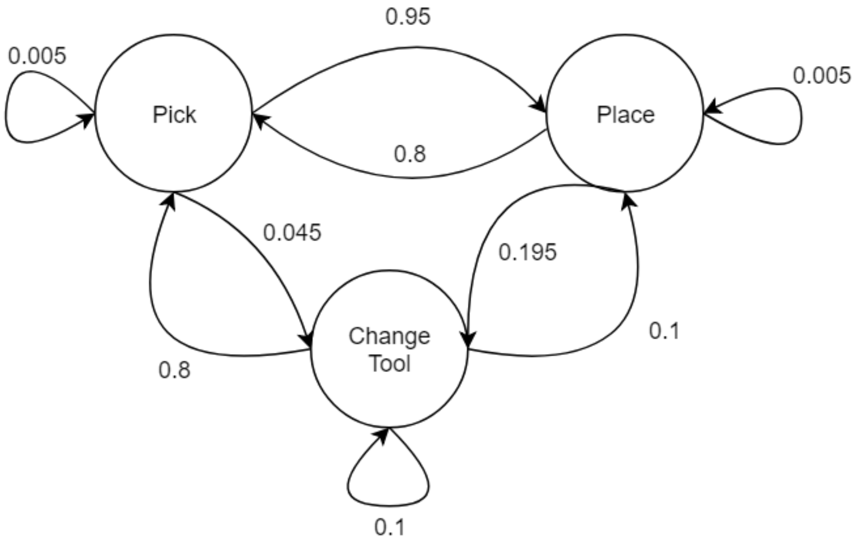


**Figure 4.1:** Possible transition diagram of three robot operations

The transition matrix in equation 4.4 can be represented graphically using the transition diagram. Transition diagram is a discrete graph with N nodes ([15]), where one node represents one state of DTMC. The transition probability $p_{i,j}$ is shown with the direct arc from node $i$ to node $j$. The probability is often written above this arc. The DTMC is possible to used as state-automata. Transition diagram can be seen in the figure 4.1.

## 4.2 Hidden Markov Model

A Markov chain presented in the previous section is used for computation of the probability of sequences, which has observable events. However, there are cases, which doesn't have events, which can be observed directly. These events are hidden. For example, the robotic manipulator is observed. For some reason, the signals with operations (pick, place, etc.) numbers are not measurable. However, signals for the robot gripper (Grip and Release commands, etc.) are measurable.

Hidden Markov model allows having both observed events (gripper commands) and hidden events (robot operations). In figure 4.2 possible transition diagram of the hidden Markov model are shown. Imagine that it is possible to observe the commands Grip and Release. Using these observations the current robotic operation could be estimated.

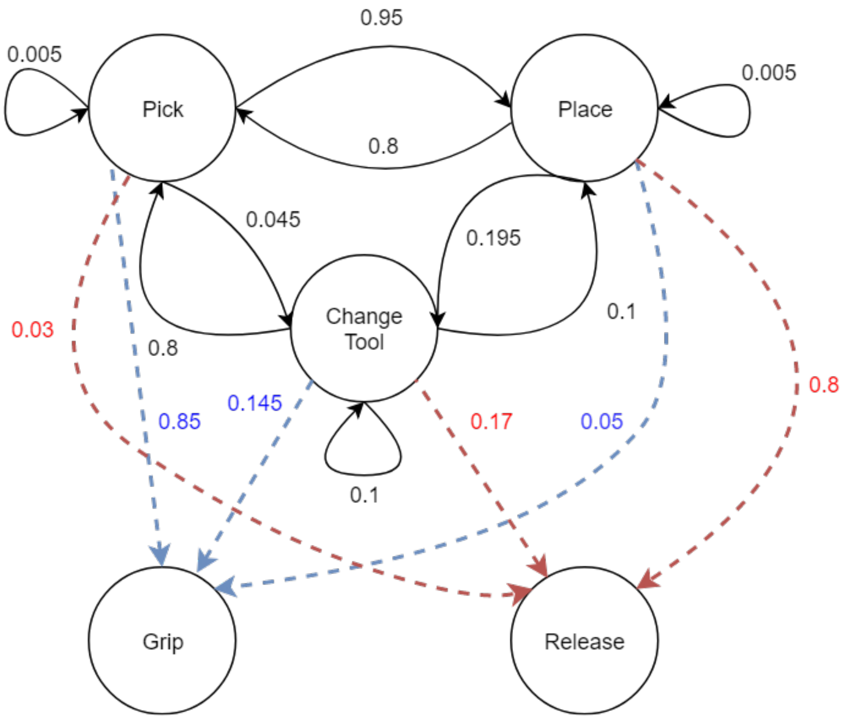


**Figure 4.2:** Possible transition diagram of three robot operations (hidden states) with observation

## 4.3 Continuous State Hidden Markov Model

The following equations and derivation are taken from [16]. The discrete-time Markov model (DT-HMM) is represented by:

- $\mathbf{Z}_N = \{z_1, z_2, \ldots, z_N\}$ - $M$-dimensional measurements made at times, $t_n$, $n = 1, 2, \ldots, N$. In case of this thesis, the dimension of measurements $M$ is $30 \times w$, where $w$ is a number of time series measured in one operation and 30 is number of measured motor variables on the robot.

- $\mathbf{X}_n = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ - $L$-dimensional unobservable states.

If the elements of the state vectors are assumed to be continuous, then the Hidden Markov Model could be continuous too (CS-HMM). The CS-HMMs represents the sequence of measurements $\mathbf{Z}_N$ as probabilistic functions of hidden states.

The DT-HMM has a prior distribution parametrized by discrete probabilities denoted as: $\pi_i$ for the initial probability of state $i$, $a_{ij}$ for the transition probability from the state $i$ to state $j$ and $b_t(j)$ for the observation likelihood.

The CS-HMMs has distribution governed by density functions $p(x_1;\theta_1)$, $p(\mathbf{x}_n|x_{n-1};\theta_X)$ and $p(z_n|\mathbf{x}_n;\theta_Z)$, whose parameters are $\theta_1$ for initial state, $\theta_X$ for state transitions and $\theta_Z$ for state observation (measurement). For notation, the following indices are defined globally:

- $n$ - $n$th element of an $N$-point sequence
- $k$ - $k$th member of a $K$-member set of training sequences
- $i$ - $i$th iteration of the Expectation Maximization (EM) algorithm (The EM algorithm is used for the estimation of HMM parameters.)

The $\mathbf{Z}_n = \{z_1, z_2, \ldots, z_n\}$ is the partial measurement sequence through time $t_n$ and $\mathbf{Z}_n^C = \{z_{n+1}, z_{n+2}, \ldots, z_N\}$ is the complement of $\mathbf{Z}_n$ in $\mathbf{Z}_N$.

The state evolution in CS-HMMs is characterized by joint likelihood, which looks like:

$$p(\mathbf{Z}_N, \mathbf{X}_N) = p(x_1)p(z_1|\mathbf{x}_1)\prod_{n=2}^{N} p(\mathbf{x}_n|\mathbf{x}_{n-1})p(z_n|\mathbf{x}_n) \tag{4.5}$$

where the explicit parametric dependence on the model parameters is dropped for convenience (see [16]).

If the CS-HMMs were used for the class assignments (labeling of the observation), it could be done using the likelihood of the measurement sequences (measured likelihood), which is obtained by marginalizing the equation 4.5 over all possible state sequences:

$$p(\mathbf{Z}_N) = \int p(\mathbf{Z}_N, \mathbf{X}_N)d\mathbf{X}_N, \tag{4.6}$$

where $\int d\mathbf{X}_N$ denotes the multiple integration $\int d\mathbf{x}_1, \int d\mathbf{x}_2, \ldots, \int d\mathbf{x}_N$. Each of that integral represents the $L$-dimensional integration over state space.

As is said in [16] because of the huge computational requirements of evaluation the discrete equivalent of 4.6, the [17] developed recursive functions to characterize and marginalize the joint probability for DT-HMMs [16]. The CS-HMMs has its counterparts (for discrete-time see [17], [18] or [19]):

- the forward density $\alpha(\mathbf{x}_n)$
- the backward density $\beta(\mathbf{x}_n)$
- the conditional state density $\gamma(\mathbf{x}_n)$
- the conditional joint density for time-adjacent states $\gamma(\mathbf{x}_n, \mathbf{x}_{n-1})$

All of these functions depend on the parameter set $\Theta = \{\theta_1, \theta_X, \theta_Z\}$.

### 4.3.1 Baum Density Definitions

The continuous-time counterparts of recursive functions presented in [17] are defined in the following subsections.

### Forward Density

The forward density is defined as:

$$\alpha(\mathbf{x}_n) = p(\mathbf{Z}_N, \mathbf{x}_n), \tag{4.7}$$

and it is fundamental for all HMM. The forward density is initialized at $n = 1$ like:

$$\alpha(\mathbf{x}_1) = p(\mathbf{z}_1|\mathbf{x}_1)p(\mathbf{x}_1). \tag{4.8}$$

Then the $\alpha(\mathbf{x}_n)$ is calculated recursively for $n = 2, \dots, n$ as:

$$\alpha(\mathbf{x}_n) = p(\mathbf{z}_n|\mathbf{x}_n) \int p(\mathbf{x}_n|\mathbf{x}_{n-1})\alpha(\mathbf{x}_{n-1})d\mathbf{x}_{n-1} \tag{4.9}$$

As is said in [16], it is convenient to define:

$$\begin{aligned} \delta(\mathbf{x}_n, \mathbf{x}_{n-1}) &= p(\mathbf{Z}_{n-1}, \mathbf{x}_n, \mathbf{x}_{n-1}) \\ &= p(\mathbf{x}_n|\mathbf{x}_{n-1})\alpha(\mathbf{x}_{n-1}), \end{aligned} \tag{4.10}$$

such that the recursion for $\alpha(\mathbf{x}_n)$ is proceed by the first computing and marginalizing $\delta(\mathbf{x}_n, \mathbf{x}_{n-1})$ and then multiplying by the output density [16].

So the equation 4.9 can be written as:

$$\alpha(\mathbf{x}_n) = p(\mathbf{z}_n|\mathbf{x}_n) \int \delta(\mathbf{x}_n, \mathbf{x}_{n-1})d\mathbf{x}_{n-1}. \tag{4.11}$$

### Backward Density

To compute the conditional densities $\gamma(\mathbf{n}_n)$ and $\gamma(\mathbf{n}_n, \mathbf{n}_{n-1})$ the backward functions are used. These are defined as:

$$\beta(\mathbf{x}_n) = p(\mathbf{Z}_n^C|\mathbf{x}_n) \tag{4.12}$$

The backward function is inicialized as $\beta(\mathbf{x}_N) = 1$ for the time $t_N$ sice $\mathbf{Z}_N^C$ is empty. With this inicialization, the recursive process can be used and the function has the following form:

$$\beta(\mathbf{x}_{n-1}) = \int p(\mathbf{x}_n|\mathbf{x}_{n-1})p(\mathbf{z}_n|\mathbf{x}_n)\beta(\mathbf{x}_n) \tag{4.13}$$

In [16] the following function is introduced:

$$\psi(\mathbf{x}_{n-1}) = p(\mathbf{Z}_{n-1}^C|\mathbf{x}_n) = p(\mathbf{z}_n|\mathbf{x}_n)\beta(\mathbf{x}_n), \tag{4.14}$$

which allows that the backward recursion is processed by computing $\psi(\mathbf{x}_n)$ first and then multiplied by the state-transition density and marginalized [16].

#### Conditional State Density and Conditional Joint State Density

The conditional state densities of CS-HMMs are used for characterization of individual states when conditioned on a given measurement sequence, which is important for state and model parameter estimation [16] and are defined as:

$$\gamma(\mathbf{x}_n) = p(\mathbf{x}_n|\mathbf{Z}_N) = \frac{\alpha(\mathbf{x}_n)\beta(\mathbf{x}_n)}{p(\mathbf{Z}_N)}. \tag{4.15}$$

The conditional joint state densities for time-adjacent states, which are important for model parameter estimation [16] are defined as:

$$\gamma(\mathbf{x}_n, \mathbf{x}_{n-1}) = p(\mathbf{x}_n, \mathbf{x}_{n-1}|\mathbf{Z}_N) = \frac{\psi(\mathbf{x}_n)\delta(\mathbf{x}_n, \mathbf{x}_{n-1})}{p(\mathbf{Z}_N)}. \tag{4.16}$$

### 4.3.2 Likelihood Evaluation and State Estimation

The measurement (observation) likelihood is given by the joint density of the measurement sequence $p(\mathbf{Z}_N, \mathbf{x}_n)$ and a single state vector like:

$$p(\mathbf{Z}_N) = \int p(\mathbf{Z}_N, \mathbf{x}_n) d\mathbf{x}_n \tag{4.17}$$

Equation 4.17 can be alternatively expressed as:

$$p(\mathbf{Z}_N) = \int p(\mathbf{Z}_N^C|\mathbf{Z}_N, \mathbf{x}_n)p(\mathbf{Z}_N, \mathbf{x}_n)d\mathbf{x}_n = \int \beta(\mathbf{x}_n)\alpha(\mathbf{x}_n)d\mathbf{x}_n, \tag{4.18}$$

where the following equality has to be noted:

$$p(\mathbf{Z}_N^C|\mathbf{Z}_N, \mathbf{x}_n) = p(\mathbf{Z}_N^C|\mathbf{x}_n). \tag{4.19}$$

The result of equation 4.19 shows that the definition for the conditional state density is appropriately normalized [16], so:

$$\int \gamma(\mathbf{x}_n) = 1 \text{ for all } n. \tag{4.20}$$

For the conditional joint state densities, a comparable argument holds by definition $\beta(\mathbf{x}_N) = 1$. From the equation at a time $t_N$, 4.20 can be obtained the simplest likelihood formula:

$$p(\mathbf{Z}_N) = \int \alpha(\mathbf{x}_n)d\mathbf{x}_n. \tag{4.21}$$

Maximum likelihood estimates for the hidden states are obtained by maximizing the state density $\gamma(\mathbf{x}_n)$ at each time step, giving the sequence of individually most likely states [16].

### 4.3.3 Parameter Estimation Of CS-HMMs

To estimate the Hidden Markov Model parameters, the EM algorithm can be used. The EM algorithm then distinguishes three types of data:

- incomplete data, which are the observed measurements,
- hidden data, which are the states,
- complete data, which are the concatenation of the observed and hidden data.

From equation 4.5 the likelihood of completed data or complete-data likelihood function (CDLF) can be obtained.

CS-HMMs has to be trained on multiple-independent measurement sequences because on the time-varying models the time averaging cannot be performed. The multisequence training set is denoted as:

$$\mathcal{Z} = \{\mathbf{Z}^1_{N_1}, \mathbf{Z}^2_{N_2}, \dots, \mathbf{Z}^K_{N_K}\}, \tag{4.22}$$

where $\mathbf{Z}^k_{N_k} = \{z^k_1, z^k_, \dots, z^k_{N_k}\}$is $k$th training sequence.

The EM is an iterative algorithm, which uses the observed data and the estimation from the previous iteration for selecting the estimates (the E-step) that maximize the conditional expectation of the logarithm of the CDLF in each iteration (the M-step). Letting $\mathcal{X} = \{\mathbf{X}^1_{N_1}, \mathbf{X}^2_{N_2}, \dots, \mathbf{X}^K_{N_K}\}$ denote the state sequences corresponding to the measurement training sequences, the CDLF for the training set is (by [16]):

$$\begin{aligned} p(\mathcal{Z}, \mathcal{X}) &= \prod_{k=1}^{K} p(\mathbf{Z}^k_{N_k}, \mathbf{X}^k_{N_k}) \\ &= \prod_{k=1}^{K} p(\mathbf{x}^k_1) p(\mathbf{z}^k_1|\mathbf{x}^k_1) \times \prod_{n=2}^{N_k} p(\mathbf{x}^k_n|(\mathbf{x}^k_{n-1}) p(\mathbf{z}^k_n|\mathbf{x}^k_n) \end{aligned} \tag{4.23}$$

Then the estimates from the $i$th iteration are:

$$\mathbf{\Theta}^{i+1} = \arg\max_{\Theta} Q(\mathbf{\Theta}, \mathbf{\Theta}^i), \tag{4.24}$$

where $Q$-function, is the conditional expectation([16]):

$$\begin{aligned} Q(\mathbf{\Theta}, \mathbf{\Theta}^i) &= \mathcal{E}_{\mathcal{X}|\mathcal{Z}_i; \mathbf{\Theta}^i} \log p(\mathcal{Z}, \mathcal{X}; \mathbf{\Theta}) \\ &\equiv \int p(\mathcal{X}|\mathcal{Z}; \mathbf{\Theta}^i) \log p(\mathcal{Z}, \mathcal{X}; \mathbf{\Theta}) \\ &= \prod_{l=1}^{K} \int p(\mathbf{X}^l_{N_l}|\mathbf{Z}^l_{N_l}; \mathbf{\Theta}^l_{N_l}) \log p(\mathcal{Z}, \mathcal{X}; \mathbf{\Theta}) d\mathbf{X}^l_{N_l}. \end{aligned} \tag{4.25}$$

As is said in [16], the E-step evaluates the $Q$-function at $\mathbf{\Theta}^i$ from the previous iteration, yielding a function that depends only on $\mathbf{\Theta}$. The M-step generates $\mathbf{\Theta}^{i+1}$ by optimizing the $Q$-function obtained from the $E$-step. The $M$-step optimized the model parameters. Therefore it has to specify the model densities. The $E$-step is specified in greater details using the dependency structure of HMM. By dropping the explicit dependence on the model parameters ([16]), the $Q$-function decomposes as:

$$Q = Q_1 + Q_X + Q_Z, \tag{4.26}$$

where each component corresponds to one of the three model densities. Using the maximization, the parameter updates for initial-state density are obtained:

$$\begin{aligned} Q_1 &= E_{\mathcal{X}|Z}\Big\{ \log \Big[ \prod_{k=1}^{K} p(\mathbf{x}_1^k) \Big] \Big\} \\ &= \sum_{k=1}^{K} \int \gamma(\mathbf{x}_1^k) \end{aligned} \tag{4.27}$$

The prameters in the state-transition density are updated by maximizing ([16]):

$$\begin{aligned} Q_X &= E_{\mathcal{X}|Z}\Big\{ \log \Big[ \prod_{k=1}^{K} \prod_{n=2}^{N_k} p(\mathbf{x}_n^k|\mathbf{x}_{n-1}^k) \Big] \Big\} \\ &= \sum_{n=2}^{N_{max}} \sum_{k=1}^{K} I_{nk}, \end{aligned} \tag{4.28}$$

where:

$$I_{nk} = \int d\mathbf{x}_n^k \int \gamma(\mathbf{x}_n^k, \mathbf{x}_{n-1}^k) \log p(\mathbf{x}_n^k|\mathbf{x}_{n-1}^k) d\mathbf{x}_{n-1}^k \tag{4.29}$$

The output-density parameters are updated using:

$$\begin{aligned} Q_Z &= E_{\mathcal{X}|Z}\Big\{ \log \Big[ \prod_{k=1}^{K} \prod_{n=1}^{N_k} p(\mathbf{z}_n^k|\mathbf{x}_{n-1}^k) \Big] \Big\} \\ &= \sum_{n=1}^{N_{max}} \sum_{k=1}^{K} \int \gamma(\mathbf{x}_n^k) \log p(\mathbf{z}_n^k|\mathbf{x}_n^k) d\mathbf{x}_n^k. \end{aligned} \tag{4.30}$$

By the substitution of the conditional state density $\gamma(\mathbf{x}_n^k) = p(\mathbf{x}_n^k|\mathbf{Z}_{N_k}^k; \mathbf{\Theta}^i)$ and the integrations of all states the expressions in 4.27, 4.28 and 4.30 are obtained by consideration the final form of 4.25.

As it is said in [16], if the continuous variables in 4.27, 4.28 and 4.30 are replaced with their discrete counterparts and the function components are maximized over those variables, the Baum–Welch re-estimation formulas [17] are obtained.

In the final application of CS-HMM, the Multivariable Normal Distribution was used as the CS-HMM distribution. The CS-HMMs was implemented using the Accord.NET C# machine learning library.

## 4.4 Hidden Markov Classifier

In previous sections, the continuous state Hidden Markov Model (CS-HMMs), which has a sequence observation as input and return the likelihood of sequence as output was presented. If there were more kinds of sequences (i.e., for operation one, operation two, etc.), then it is possible to have multiple Hidden Markov Models which will return likelihood of that, the sequence belongs to model.

The likelihoods can be then compared, and model with the highest likelihood can be then selected as the label of the input sequence $z_n$. Therefore, for each robotic operation, one CS-HMM was created. The resulting model (label) is chosen by using the maximum likelihood. However, choosing the maximum likelihood may not be optimal for all problems [19]. To create an optimal classifier, it is possible to multiply each model by some factor. Then the resulting classifier is called Bayesian Maximum a posteriori classifier (MAP).

The maximal likelihood (ML) is defined as:

$$\hat{y} = \underset{\lambda_j \in \Omega}{\arg\max}\; P(z_n|\lambda_j), \tag{4.31}$$

where $\lambda_j$ is $j$ Hidden Markov Model of class $j$ and $\Omega$ is set of all Hidden Markov Models.

On the other hand, the maximum a posterior decision rule attempts to select the class with maximum probability given the sequence [20]. Using the Bayesian rule:

$$P(\lambda_j|z_n) = \frac{p(z_n|\lambda_j)p(\lambda_j)}{\int p(z_n|\lambda_j)p(\lambda_j)d\lambda_j}, \tag{4.32}$$

then the maximum a posterior decision (MAP) is:

$$\hat{y} = \underset{\lambda_j \in \Omega}{\arg\max}\; P(\lambda_j|z_n). \tag{4.33}$$

Note that the ML and MAP are equivalent, if the prior probabilities of each model $p(\lambda_j)$ are equal.

## 4.5 Implementation

### 4.5.1 Discrete-Time Markov Chain Of Robot Behavior

As is mention in section 4.1, the Discrete-time Markov Chain can be used as a “state automata“. With this state automata, it is possible to create the model of robot behavior. Firstly, the states of the Discrete-time Markov Chain has to be determinated. One state will represent one robot operation. From the robot programs presented in section 2.8, the log (record) of the operation was saved. From the operation log, it is possible to create the transition matrix of the Markov Chain using the frequencies of the changes of the operations.

However, the operations are in the robot executed one by one. Therefore, the transition probability from the state $i = 1$ to the state $j = 2$ will be equal to 1. If the input sequence of Markov Chain with such transition matrix has an error, then the capability of resolving such error is limited. The model will then return an inaccurate likelihood.

Therefore it is possible to use the Laplace Smoothing. Laplace Smoothing is a technique used to smooth the categorical data. In general, it can be written as:

$$P(y = j) = \frac{\sum_{i=1}^{m} L\{y^i = j\} + 1}{m + k} \text{ if } y \in \{1, 2, \ldots, k\}, \tag{4.34}$$

where $L$ is the likelihood.

When counting (*Count()* function) of transition frequencies is used, the Laplace smoothing is rewritten as:

$$a_i(j) = \frac{\text{Count}(i \to j) + 1}{\text{Count}(i) + n}, \tag{4.35}$$

where $n$ is a number of added occurrences into $i$th row of transition matrix $a$.

### 4.5.2 Dimension Reduction

From the measured data, the features, which holds the most information about the robot operation were selected. The reason for the dimension reduction is that the lower dimension of input sequence into Hidden Markov Model should increase the speed of the classification (likelihood calculation).

For the selections of the features, the Random Forest Algorithm was used. Random Forest has several decision trees. Every node in the decision tree is a condition of a single feature. The node is designed to split the dataset into two datasets (decision).

Impurity is a measure of how the (locally) optimal condition is chosen. For the classification, the impurity can be an information gain (entropy) or the Gini impurity [21]. So when the tree is trained, it is possible to compute how much each feature decreases the weighted impurity in a tree. For the forest, the impurity can be than average across all trees, and the features can be ranked according to them [21].

The Random Forest was trained on the labeled dataset and ten most important features according to it can be seen in the figure 4.3.

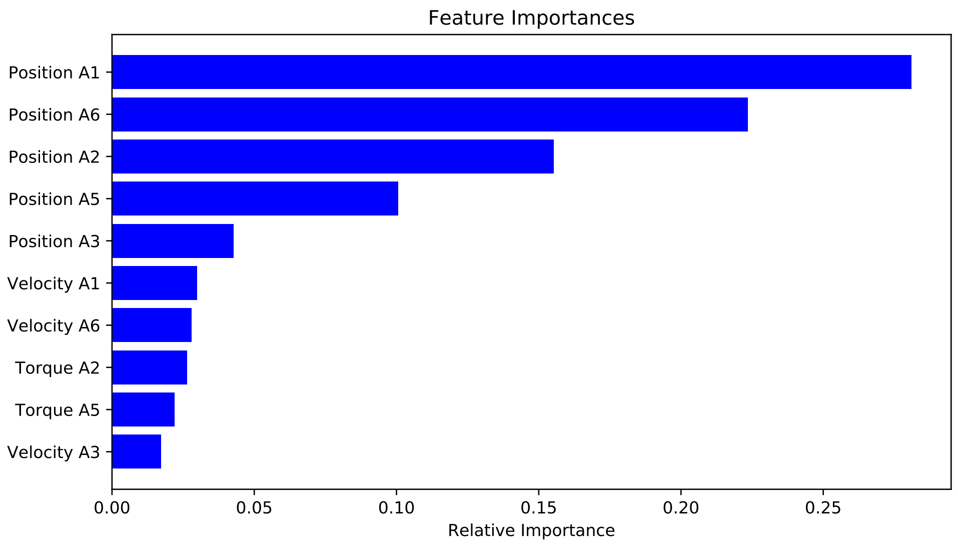


**Figure 4.3:** Feature importances and its relative importance

After determination of the essential features, from the input data set (sequences), these features were selected and used.

### 4.5.3 Hidden Markov Classifier For Robot Operation Classification - Continuous Model

The Hidden Markov Classifier, presented in section 4.4, is widely used for sequence classification. Therefore, it is a perfect choice for the robot operation classification.

From the measurement, the training data set was created. Each of the operation (time series batch) was labeled with class number. To determine the model performance, the input data sequence was split into two folds — one for the learning of the Hidden Markov Model Classifier, the second one for the performance check after training on the unknown data. The Accord.NET framework, which was used for training the Hidden Markov Classifier, train each model separately. The method of how each model was learned is described in the subsection 4.3.3.

For the training of the classifier, the computing cluster of *MetaCenterVO* [1] was used. Table 4.1 shows, how the number of the hidden states influence the classification of training data, test data and of the corrupted data.

On the corrupted data set, it was expected that the Markov Classifier which was learned on the train data set will have poor results. As can be seen in the table 4.1, the expectation was confirmed. The accuracy was computed using the confusion matrix from Accor.NET.

| Number of Hidden States | Training Data | Test Data | Corrupted data |
|---|---|---|---|
| 6 | 1.00 | 1.00 | 0.86 |
| 14 | 1.00 | 1.00 | 0.892 |
| 20 | 1.00 | 1.00 | 0.86 |
| 30 | 1.00 | 1.00 | 0.88 |

**Table 4.1:** The influence of the number of hidden states on the classification - the accuracy computed using the Accord.NET confusion matrix

As can be seen in the table 4.1, the models have 100% accuracy on the training and testing data. However, on the corrupted data, the accuracy is weak. Therefore, it was decided to modify the input sequence. Again the method of Random Forests was used to find out the features, which holds the most information about the state in the corrupted data set. The figure 4.4 shows these features.

The result from the feature selection on corrupted data set was then merged with the previously selected features. In the table 4.2, the final features can be seen.

Besides that, table 4.2 shows that the key features for identification of the robot operation are primary variables of the robot axis, which moves most during the assembly operation. Again, the performance check of the classification was made on the same data set as before. The table 4.3 shows that the new features help with the classification and can be seen that the accuracy is 100% in most cases. The main reason for such accuracy is that the positional data were used. Positional data have enough information for the distinguishment of operations. Such a high accuracy also

[1] Computational resources were supplied by the Ministry of Education, Youth and Sports of the Czech Republic under the Projects CESNET (Project No. LM2015042) and CERIT-Scientific Cloud (Project No. LM2015085) provided within the program Projects of Large Research, Development and Innovations Infrastructures. For futher informtion see *MetaCenterVO*.

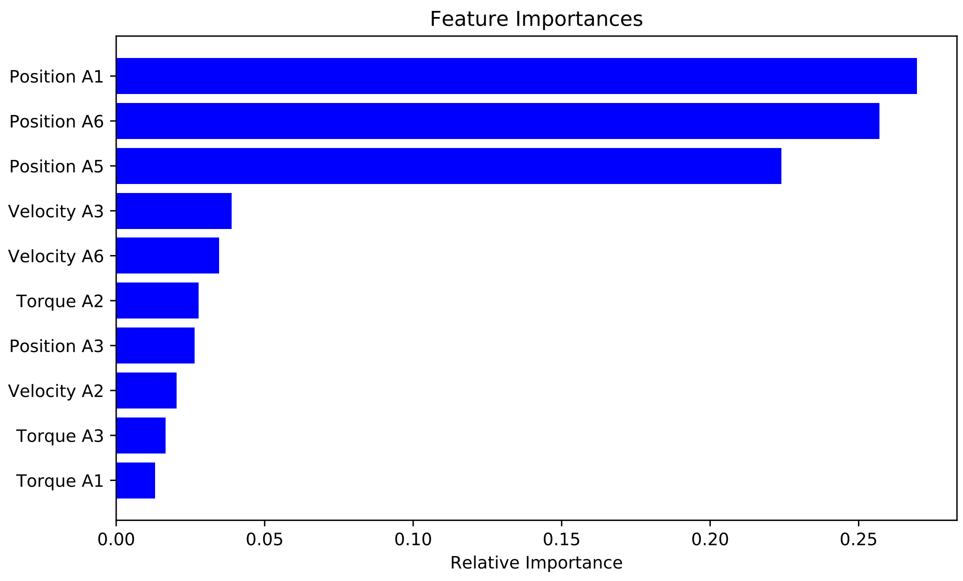


**Figure 4.4:** Feature importances and its relative importance from corrupted data set

show that the robotic controller is robust, and the robot has high repeatability. If there was significant payload (e.g., $9Kg$ payload) on the robot, the deviation should occur, and there will be some misclassification.

The classifier with 14 hidden states was chosen as the final classifier. This classifier has the best trade-off between the classification accuracy and the time of classification.

### 4.5.4 Fault Detection

The model, created in the previous section can classify the input sequence with an accuracy of 100%. However, it is not capable of detecting the fault on the robot. Therefore the threshold model [22] was created. The threshold model acts as a baseline for decision rejection. If none of the classifiers can produce a higher likelihood then the threshold model, the decision is rejected. The functionality of the threshold model is described in [22].

The Accord.NET framework has already implemented support for the threshold model creation during the learning process of the classifier. If that model works correctly, it is necessary to set up sensitivity property. Sensitivity property is governing the rejection by the threshold model in Accord.NET framework.

As it is discussed in [22], the sensitivity has a significant impact on the rejection. A few experiments with the different value of the sensitivity were performed. As the best trade-off between the rejection on the not-corrupted data set and on the corrupted data set, the final value $p(TM)$ = 1e-150 of the sensitivity was chosen, according to the notation presented in [22].

The final model with given sensitivity has been tested on the same data set as

| | Variable |
|---|---|
| 1 | Position Axis A1 |
| 2 | Position Axis A2 |
| 3 | Position Axis A3 |
| 4 | Position Axis A5 |
| 5 | Position Axis A6 |
| 6 | Torque Axis A1 |
| 7 | Torque Axis A2 |
| 8 | Torque Axis A3 |
| 9 | Torque Axis A5 |
| 10 | Velocity Axis A1 |
| 11 | Velocity Axis A2 |
| 12 | Velocity Axis A3 |
| 13 | Velocity Axis A6 |

**Table 4.2:** Final selected features

| Number of Hidden States | Training Data | Test Data | Corrupted data |
|---|---|---|---|
| 6 | 1.00 | 1.00 | 1.00 |
| 14 | 1.00 | 1.00 | 1.00 |
| 20 | 1.00 | 1.00 | 1.00 |
| 30 | 1.00 | 1.00 | 1.00 |

**Table 4.3:** The influence of the number of hidden states on the classification, with final features - the accuracy computed using the Accord.NET confusion matrix

models in previous section. The model accuracy on the not-corrupted data set has decreased from 100% to 91.6%. The remaining 8.4% of data were classified correctly, but they were also labeled as data with fault. The model is capable to detect the fault with accurcy of 60% on the corrupted data set. The classification of the robotic operation was still 100%.

### 4.5.5 Model Using DTMC and Markov Classifier

For the cases, when the probability from the Markov classifier is less than 100%, the following approach can be used. This approach used the Hidden Markov Classifier from the subsection 4.5.3 and the DTMC from the subsection 4.5.1.

Let's first define the folowing equations:

$$p_c(\mathcal{D}^t|c_t)p_d(c_t) = p(\mathcal{D}^t, c_t) \propto p_a(c_t|\mathcal{D}^t), \tag{4.36}$$

where $p_c(\mathcal{D}^t|c_t)$ is a class probability of the sequence from the Markov classifier and $p_d(c_t)$ is a probability computed like:

$$p_d(c_t) = \sum_{c_{t-1}} p(c_t|c_{t-1})p_a(c_{t-1}|\mathcal{D}^{t-1}), \tag{4.37}$$

where $p(c_t|c_{t-1})$ are transitions probabilities from DTMC transition matrix and $p_a(c_{t-1}|\mathcal{D}^{t-1})$ is a result of the equation 4.36 in previous time $t-1$. Equation 4.37

represents the current state (class) given by the new apriori probability from equation 4.36.

In the beginning, apirior probability $p_a$ is initialized using the uniform distribution (each state has the same probability).

For time $t-1$, let's first use the apriori probility $p_a(c_{t-1}|\mathcal{D}^{t-2})$ from time $t-2$ for computation of $p_d(c_{t-1})$ using the equation 4.37. The probility $p_d(c_{t-1})$ is then used in the 4.36 like:

$$p_a(c_{t-1}|\mathcal{D}^{t-1}) \propto p_c(\mathcal{D}^{t-1}|c_t)p_d(c_{t-1}), \tag{4.38}$$

where $p_c(\mathcal{D}^{t-1}|c_t)$ are likelihoods of input sequence given by the Markov classifier. From the result of equation 4.38 a current class (robot operation) using the maximum a posterior decision (MAP, see section 4.4) is computed.

For time $t$, the steps are the same. The result from equation 4.38 are used as apriori probability in equation 4.37.

For testing, a weak Markov classifier was created. This classifier was trained on the measured data as a classifier from subsection 4.5.3, but no feature selection was used. The apriori probability of this classifier is equal for each state. Therefore, the state of the input sequence is selected using the Maximal Likelihood (ML) decision (see section 4.5.3). Let's distinguish this classifier as a classifier, which is using the ML decision and submitted method as a classifier, which is using the MAP decision.

Only the position data was used as input sequence into both classifier. Therefore, the classification will be more inaccurate, and the classification of corrupted data is expected to be weaker.

| Method | Training Data | Test Data | Corrupted data |
|---|---|---|---|
| ML | 1.00 | 1.00 | 0.92 |
| MAP | 1.00 | 1.00 | 0.95 |

**Table 4.4:** Results of the experiments

Table 4.4 shows the results of the classification before and after using the MAP approach. The classification of the model which is using the MAP approach is in comparison better. Most of the misclassifications were given by the misclassification of the operation (trajectory) one. The operation one was mistakenly classified as the operation three. It is because the main impact of the decision is provided by the classifier (in our case weak HMM) than DTMC and it has a significant impact on the overall classification.

However, the MAP approach increased the accuracy of the classification and can be used as a supporter of a weak classifier (i.e., in combination with the methods presented in [9, 10]).

## 4.6 Comparison with Previous Method

### 4.6.1 Statistical Moments

In section 3.1 the method which uses the statistical moments is suitable to determine the type of the operation. But for the classification of the operations, which has similar movements, this method doesn't have such accuracy. The statistical moments are computed across the all operation, and if there are some deflections (such as the similar movement) the statistical moments can hide them (usage of only one Gaussian, etc.). On the other hand, the Hidden Markov Models apply multiple Gaussians. Aforementioned makes the method more suitable for the distinction of the operations.

### 4.6.2 Stochastic Modeling Using Time-Varying Markov Model

The method presented in section 3.2 has similar accuracy as the method created in section 4.5. Both methods use the Markov property for the estimation. The method from [9] models each sample in sequence with one Gaussian. The stochastic modeling lost its accuracy in the cases when the corrupted data were used. In submitted approach, the accuracy remains.

## 4.7 Chapter Overview

In this chapter more precise method of the classification (labeling) of the robot operation was developed. This method uses the continuous-state hidden Markov classifier for the classification of the time-series batch measured on the robot. Also, the application of the threshold model for the detection of the bias from normal was presented. This model can detect the fault of the robot.

The method was tested on the non-corrupted data set as well as on the corrupted data set. The results of the testing show that the method is well suitable for operation classification. It can precisely distinguish between similar operations.

Then the method, which combines discrete-time Markov chain and weak Markov classifier was presented. The test results show that the combination of the probabilities is capable of increasing the accuracy of the classification.

At the end of the chapter, the comparison with the already done methods was done.

# Chapter 5

# Process Mining

Process Mining is an analytical method which aims to discover, monitor and improve real processes using the knowledge from the event logs of the systems (log of the robotic operation, log of the computer program, etc.). It helps to eliminate the partial overview of how the process performed. The most frequent partial overviews are listed below(taken from [23]):

- **Subjectivity** - Everyone involved in the process have subjective pictures of it (based on its role). It is one of the reasons why the processes *As-is* in a classical workshop are difficult to be discovered.
- **Partial view** - Some processes aren't done by a single person (machine). Instead, they are done by teams of multiple people (machines), departments, or companies which work together on the final product, service.
- **Change** - The processes change all the time. If it was well documented at the beginning, it is likely that some changes were made during the lifecycle of the process. Even during the analysis, the process could change. It is hard to keep the process documentation well maintained.
- **Invisibility** - In these days it is much easier to lose the track of what is "going on" in the process. Some important information about the process may be stuck in the files of the system.

The process mining uses contextual relationships in the process for the diagnostic of the system. It detects or diagnoses the systems based on the facts mined from the process logs. It pairs the event data (observed behavior) and the process models (made by hand or automatically) [24], and through these pairs, it checks for the deviation in the behavior, predicts delays, support decision making, compliance and recommends a possible redesign of the process [24].

## 5.1 Process Mining Of The Robot Operations

The process mining uses the data (logs) that are already collected as a byproduct of the automation and digitalization of the business or industrial processes [24]. This statement also stands for the data collected from the robot or classified by the Markov classifier 4.4.

### 5.1.1 Data Preparations

According to the [24], the data has to have some minimal requirements. The minimum requirements are these three elements:

- **Case ID** - It specifies the instance of the process (scope of the process). It determines when the process starts and when it ends. In the case of this thesis, it reflects one assembled product (see section 2.8).
- **Activity** - It forms one step in the process. The activity names determinate the steps in the process map and their granularity [24].
- **Timestamp** - It is necessary to have at least one column which indicates when each of the activities took place [24]. If the log is not sequential, the timestamp is needed for determinating the order of the activities in the process. It has to be noted, that the timestamp has to be determined according to ISO 8601 time format.

The logs forming the operation have to be transformed, so they fulfill the above mentioned minimal requirements. In this thesis, the data used for the process mining, are raw data measured from the robot, which were used for the training the Hidden Markov Classifier. This data, have all the necessary properties for the process mining (labels, timestamps, etc.). However, they have to be transformed. According to the minimal requirements, the raw data were transformed into the following form:

- Case ID - assembled product,
- Activity - operation number,
- Start Time - the start time of operation (time of the first occurrence of the operation),
- Stop Time - the end time of operation (time of the last occurrence of the operation).

For the transformation of the log data, the *XesProcessFromCsv* python class in DataProcessing application was created. The data from the transformation are then stored in the .CSV file.

However, every tool for the process mining, which is currently on the market uses the .XES file format. This format is standardized by IEEE 1849-2016 standard [25]. XES is a shortcut for the extensible event stream, and it is an XML type tag-based language. As it is said in [25], its aim is to provide a unified and extensible methodology for capturing systems behaviors using event logs and event streams for designers of the information systems.

For the use of process mining tools, it wasn't necessary converted the .CSV logs into the .XES file. The ProM (used process mining tool) has built in .CSV to .XES converter.

### 5.1.2 ProM

ProM (Process Mining framework) is an open source framework for mining algorithms. It has a wide variety of process mining techniques in the form of plug-ins.

As it was mention in the section before, ProM has capabilities of importing the .CSV files and convert it into .XES file. After the importing, it is possible to use some of the already implemented methods for the analysis of the logs.

#### Creation Of Discrete Model

In the ProM, there is a possibility of creating the transition matrix, between the states based on the frequency of events transitions. This transition matrix is in ProM called Activity Matrix and can be used for creating of a discrete model. The resulting transition matrix can be seen in Appendix C. If the value is close to 1.0, then the transition between states is quite confident. If the transition between two states is equal to 0.0, then it cannot be determined. The values equal to $-1.0$ represents the transitions, which cannot happen.

The resulting matrix can be then used for the creation of a discrete model. However, it is necessary to filter the values, to achieve the sum of the values in a row is equal to 1. Therefore, the values were transformed from the range $[-1.0,\ 1.0]$ to the range of $[0.0,\ 1.0]$. After this transformation, each row has to be normalized.

The discrete model created in such a way has similar behavior as a model created in the previous chapter 4. However, the usage of Laplace smoothing in chapter 4 has a significant impact on model behavior.

In the cases of mistaken steps, it allows full recovery of the model, without any reset. However, if the model created by process mining has some inaccurate steps in the input, it kills the process of classification (possible multiplication with zero). Therefore, it is recommended to use Laplace smoothing on such a model, so it allows full recovery.

The power of process mining is that the user doesn't need to have full knowledge of the processes. Also, the information stored in Case ID allows other process mining techniques (algorithms) for model determination.

#### Creation Of Not-Ideal Process Model

Because the model mined from the logs is ideal, it was decided to create a simulated noise process. For the creation of log, the discrete-time Markov model, which has the same number of states but different transition matrix, was created. This Markov model than generates a new sequence of states occurrences, in the same amount as in the previous case. The previous information about Case ID and Time Stamps were added to this state occurrences. After that, the same process mining techniques for model creation were used. The mined Activity matrix of this process can be seen in Appendix C.

As was discussed in the previous section, the user doesn't have any further information for the creation of the model. The resulting discrete model has the same transition matrix as the Markov model, which generates the sequence.

### 5.1.3 Process Mining Techniques For Process Visualization

#### Miner with Inductive Visual Miner

Inductive process miner is miner technique developed by Sander Leemans (see [26]). As it is said in [27], the inductive process miner is a framework, which has directly-follows graphs as an input. The directly-follows graphs are obtained from the event logs in linear time. The acquisition uses highly-scalable techniques such as Map-Reduce (see [27]). The method used for building of the process model recursively use a divide-and-conquer strategy. It splits the directly-follows graph using the recursion on the sub-graphs until it finds a base case.

This framework was used on the event logs obtained by the measurement. The resulting process model can be seen in figure 5.1 visualized using the Folding Process Tree Visualizer. As can be seen, the process model of the robotic operation is an ideal case of the process. There are not any events (operation) which occurs irregularly. If the miner was used on the second log (one created by the modified Markov model from 5.1.2) the resulting process graph can be seen in figure 5.2. As can be seen from the figure, the mined process graph is more complexed than in previous case.

The process mining has been shown as a powerful tool for determinating the discrete processes when the operation (process) are complex, and the user doesn't have any inside look at process behavior. The process mining is a good tool for cases, where the documentation about the process is out of date or completely lost.

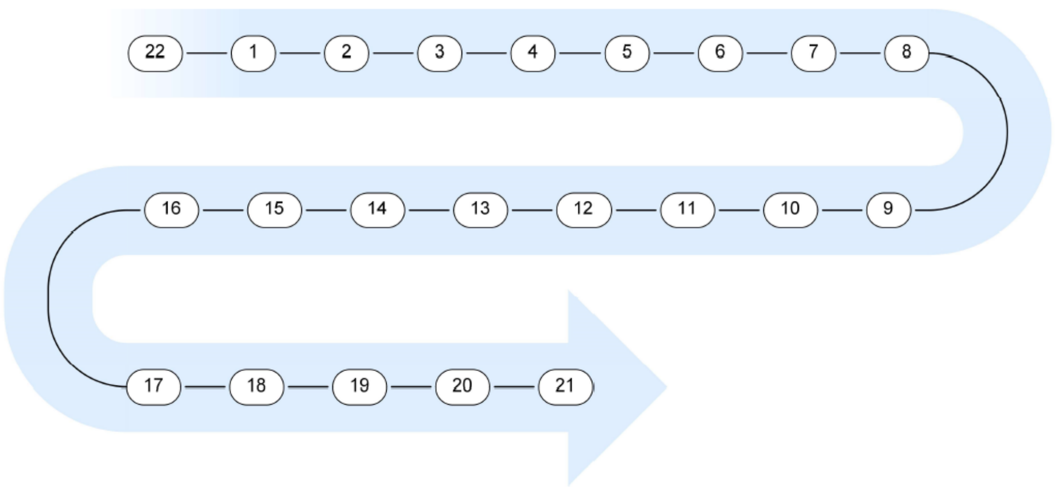


**Figure 5.1:** Miner With Inductive Visual Miner - final model(visualized using the Folding Process Tree Visualizer)

## 5.2 Chapter Overview

In this chapter, process mining was presented. The concept of it was described, and two methods were tested. One of them creates the Activity matrix and the second uses the Inductive process mining technique.

It was shown and discussed, that the process mining techniques help with the creation of the discrete models. However, it was recommended to use the Laplace smoothing to determine the better result in cases of misclassification.

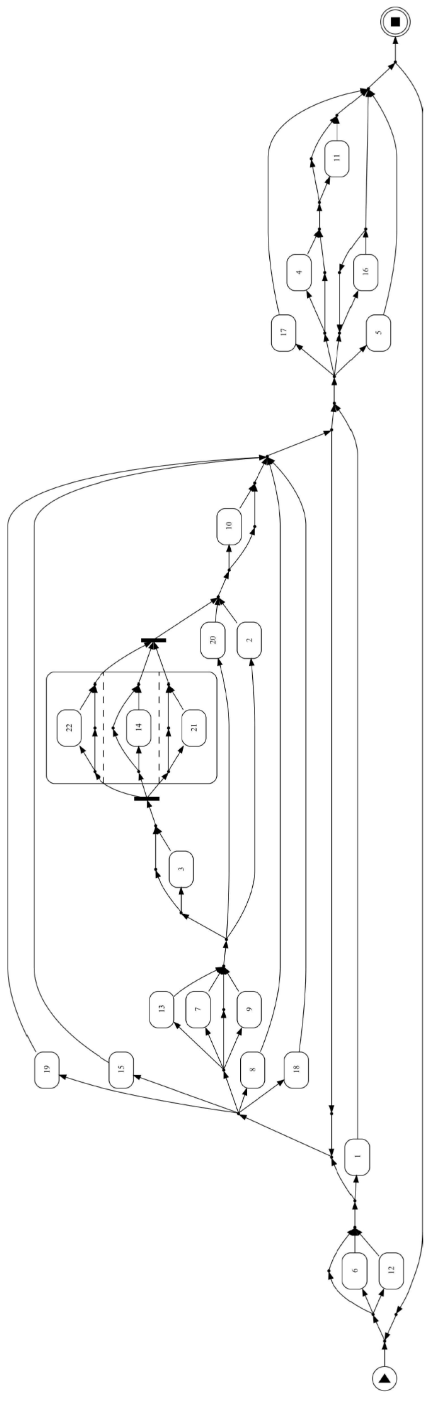


**Figure 5.2:** Mined process graph using the Inductive Process Miner - process generated by modified Marov model

# Chapter 6
# Conclusion

The object of the submitted thesis is an acquisition and analysis of the movement data from the industrial manipulator. First two methods of data acquisition were presented, implemented and tested.

The test showed that using the Ethernet KRL is not sufficient for the data acquisition due to its data acquistion period and latency. However, the second approach based on the PROFINET has an adequate period of the data transition. For the measurement via PROFINET, the IIoT (Industrial Internet of Things) device Revolution Pi was used. This device is capable of being simultaneously connected to the industrial network PROFINET and the Internet. Therefore, the final application for data acquisition and analysis was implemented at this device.

On the studied system, which was industrial manipulator KUKA KR10 Agilus sixx, a new assembly operation of a LEGO structure containing twenty-two robotic operations (programs) was created. The movement data measured on such operations was used for the tracking of stability and deviations. Two methods created in the past were then used for the analysis. First one is using the statistical moments and experiments showed that the method is suitable for classification of the robot operation type (pick and place operations, etc.). The second method is using time-varying Markov model. It has a higher distinguishing of the score for very similar operations than the first one. The experiments showed that this method has great results in operation type classification as well as in operation number.

In this thesis, a framework using the discrete-time Markov model of robot behavior was created. Firstly the fundamental theory of the Markov models was presented and then implemented. Then the continuous-state hidden Markov model classifier (CS-HMM) which is using the movement data as a complement to the Markov model of robot behavior was created. On the measured data set, the dimension reduction using the Random Forest was performed for increasing speed of the classification (by reduction of input data dimension).

The experiment showed that the CS-HMM classifier has 100% accuracy on the testing as well as the corrupted data sets. Such a high classification is possible due to the characteristic of the data. The position data are sufficient for the classification of the operation. Such a high accuracy also shows that the robotic controller is robust, and the robot has high repeatability.

However, the method could be deployed in a standalone way. Moreover, the possibility of using the method of threshold model with the combination of CS-HMM classifier for fault detection was demonstrated. The tests confirmed that this method could be utilized for the fault detection on the industrial manipulators. The application of the method is not limited by the industrial manipulators only. The model could be deployed on any type of device which is equipped with the sensors.

In cases, when the accuracy of the classifier is below some threshold, the method of combining the discrete-time Markov model with classifier was created. The experiments showed that the designed approach is capable of increasing the model accuracy.

In the end, the fundamentals of process mining were presented and tested on the logs of robot operations. It was discussed and proved that the process mining is a great tool for discrete model determination because it also filters out partial overviews.

In summary, the key contribution of this thesis is the creation of the framework for the time series classification. Due to the use of the hidden Markov model, this framework could be used on any device, which is equipped with sensors. Another contribution of this thesis is an in-depth analysis of time series acquisition techniques on Kuka robots and the analysis of using the process mining for the creation of a discrete model of the unknown system.

## 6.1 Future Work

The submitted approach presented in chapter 4 could be extended in the future. It could be added as one of the possible models for the analysis or classification in the framework presented in [10] and be deployed to Azure cloud. Additionally, the use of hidden Markov model for the fault detection could be enhanced by adding the data of real-life faults.

Furthermore, the measurement via Ethernet KRL should be performed in the network with higher traffic, when the communication between robot and external PC is influenced.

# Appendix A
# Bibliography

[1] Santiago García Garrido. Types of Maintenance. `https://www.mantenimientopetroquimica.com/en/typesofmaintenance.html`, May 2019.

[2] Predictive Maintenance Explained. `https://www.reliableplant.com/Read/12495/preventive-predictive-maintenance`, May 2019.

[3] M. L. Araiza. A formal framework for predictive maintenance. In *Proceedings AUTOTESTCON 2004.*, pages 489–495, September 2004.

[4] B. Lu, D. B. Durocher, and P. Stemper. Predictive maintenance techniques. *IEEE Industry Applications Magazine*, 15(6):52–60, November 2009.

[5] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi. Machine Learning for Predictive Maintenance: A Multiple Classifier Approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820, June 2015.

[6] J. Kinghorst, O. Geramifard, M. Luo, H. Chan, K. Yong, J. Folmer, M. Zou, and B. Vogel-Heuser. Hidden Markov model-based predictive maintenance in semiconductor manufacturing: A genetic algorithm approach. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1260–1267, August 2017.

[7] Z. Wu, H. Luo, Y. Yang, P. Lv, X. Zhu, Y. Ji, and B. Wu. K-PdM: KPI-Oriented Machinery Deterioration Estimation Framework for Predictive Maintenance Using Cluster-Based Hidden Markov Model. *IEEE Access*, 6:41676–41687, 2018.

[8] António Simões, Jose Viegas, José Farinha, and Inácio Fonseca. The State of the Art of Hidden Markov Models for Predictive Maintenance of Diesel Engines: HMM for predictive maintenance of diesel engines. *Quality and Reliability Engineering International*, January 2017.

[9] M. Ron and P. Burget. Stochastic modelling and identification of industrial robots. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 342–347, August 2016.

[10] Ondřej Novák. Robot diagnostics based on monitoring of its kinematic variables. Master's thesis, CTU in Praque, Prague, 2019.

[11] KUKA Roboter Group GmbH. *KUKA System Softeare 8.4 - Operating and Programming Instructions for System Integrators.* Augsburg, Germany, 1 edition, May 2015.

[12] MongoDB, Inc. .NET MongoDB Driver Documentation. `http://mongodb.github.io/mongo-csharp-driver/2.7/?jmp=docs`, May 2019.

[13] KUKA Roboter Group GmbH. KUKA.Ethernet KRL 2.1. `http://www.wtech.com.tw/public/download/manual/kuka/krc4/KST-Ethernet-KRL-21-En.pdf`, 2012.

[14] KUNBUS GmbH. Open Source IIoT Gateway RevPiConnect Flyer. `https://revolution.kunbus.com/wp-content/uploads/manuell/datenblatt/revpi_connect_flyer_en.pdf`, May 2019.

[15] V. G. Kulkarni. *Introduction to Modeling and Analysis of Stochastic Systems.* Springer Texts in Statistics. Springer-Verlag, New York, 2 edition, 2011.

[16] P. L. Ainsleigh, N. Kehtarnavaz, and R. L. Streit. Hidden Gauss-Markov models for signal classification. *IEEE Transactions on Signal Processing*, 50(6):1355–1367, June 2002.

[17] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171, February 1970.

[18] Petr Pošík. Hidden Markov Models. *Czech Technical University in Prague Faculty of Electrical Engineering Dept. of Cybernetics*, 2017.

[19] Christopher M. Bishop. *Pattern recognition and machine learning.* Information science and statistics. Springer, New York, 2006.

[20] César de Souza. Sequence Classifiers in C# - Part I: Hidden Markov Models - CodeProject. `https://www.codeproject.com/Articles/541428/Sequence-Classifiers-in-Csharp-Part-I-Hidden-Marko`, May 2019.

[21] Ando Saabas. Selecting good features – Part III. `https://blog.datadive.net/selecting-good-features-part-iii-random-forests`, May 2019.

[22] Lee Hyeon-Kyu and Jin H. Kim. An HMM-based threshold model approach for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):961–973, October 1999.

[23] Anne Rozinat and Christian Günther. What Is Process Mining? — Process Mining Book 2.1 documentation. `http://processminingbook.com/intro.html`, 2018.

[24] Pedro Robledo. Process Mining plays an essential role in Digital Transformation. *Medium*, September 2018.

[25] IEEE 1849-2016 - IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. `https://standards.ieee.org/standard/1849-2016.html`, May 2019.

[26] Sander Leemans. Inductive visual Miner - Sander Leemans. `http://leemans.ch/leemansCH/inductivevisualminer`, May 2019.

[27] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Scalable process discovery and conformance checking. *Software & Systems Modeling*, 17(2):599–631, May 2018.

# Appendix B

# Project Specification - English Version

1. Design a way of data acquisition from industrial robots KUKA using PROFINET and Ethernet KRL. Study the ways of time synchronization in the robots using e.g. NTP and implement timestamping for each frame sent from the robot. Compare both interfaces regarding throughput, jitter and eventually suggest other indicators.

2. Design and implement robot movements to be able to monitor long-term stability, resp. deviation in the robot behavior based analysis of the acquired time series. Use a method based on statistical moments (see [10]) and a method based on hidden Markov models (see [9]). Design key indicators to asses the sucess of both methods.

3. Using the previous item design a discrete Markov model of the robot behavior, where the robot operations will be modelled as the states of the model. It will be a higher-level model that will complement the continuous model, which considers individual movement quantities.

4. Store the identified operations in memory and explore the possibilities of process mining methods to build a discrete event model of the robot behavior. Compare this approach with the discrete Markov model.

# Appendix C

# Tables

| **From\To** | **1** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **2** | **20** | **21** | **22** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **10** | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **11** | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **12** | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **13** | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **14** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **15** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **16** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **17** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **18** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **19** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | -1.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **2** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | -1.0 | -1.0 | -1.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **20** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.95 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **21** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **22** | 0.95 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **3** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **4** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 |
| **5** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 | -1.0 |
| **6** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 | -1.0 |
| **7** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 | -1.0 |
| **8** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.97 |
| **9** | -1.0 | 0.97 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 |

**Table C.1:** Activity Matrix created by ProM based on logs from model preseneted in chapter 4

| **From\To** | **1** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **2** | **20** | **21** | **22** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0.0 | -1.0 | -1.0 | -1.0 | 0.8 | -1.0 | -1.0 | -1.0 | -0.51 | -1.0 | -1.0 | -0.53 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.67 | -1.0 | -1.0 | -1.0 |
| **10** | -1.0 | 0.0 | 0.875 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.67 | 0.84 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **11** | -1.0 | -1.0 | 0.0 | 0.857 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.84 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **12** | -1.0 | -1.0 | -1.0 | 0.0 | 0.75 | -1.0 | 0.67 | 0.84 | -1.0 | 0.67 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.5 | -1.0 |
| **13** | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.75 | -1.0 | -1.0 | 0.8 | -1.0 | -0.42 | -0.69 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.8 | -1.0 |
| **14** | 0.5 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.5 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -0.67 | -1.0 | -1.0 | -1.0 | 0.84 | -1.0 | 0.67 | -1.0 | -1.0 |
| **15** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.67 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.84 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -0.63 | 0.84 |
| **16** | 0.75 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.5 | -1.0 | -1.0 | -1.0 | 0.8 | -1.0 | -1.0 | -1.0 | 0.84 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **17** | -0.56 | -1.0 | -1.0 | 0.84 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | -0.59 | 0.75 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.84 | -1.0 | -1.0 |
| **18** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -0.48 | 0.0 | 0.84 | 0.8 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -0.68 | 0.84 |
| **19** | -1.0 | -1.0 | -1.0 | -1.0 | -0.53 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | -1.0 | 0.92 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.93 | -1.0 | -1.0 |
| **2** | -0.67 | -1.0 | -1.0 | -1.0 | -0.39 | -1.0 | 0.8 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | -1.0 | -1.0 | -1.0 | 0.9 | 0.8 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **20** | -1.0 | 0.84 | 0.8 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.8 | -1.0 | -1.0 | 0.0 | -0.49 | -1.0 | 0.75 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **21** | -1.0 | 0.8 | -1.0 | -1.0 | -1.0 | -0.43 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.8 | -0.81 | 0.0 | -0.33 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **22** | 0.67 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.75 | -1.0 | -0.63 | 0.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.86 | 0.75 | -1.0 |
| **3** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.67 | -1.0 | -1.0 | -1.0 | -1.0 | 0.67 | -1.0 | -1.0 | 0.75 | 0.67 | 0.0 | 0.75 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 |
| **4** | -1.0 | -1.0 | 0.5 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.92 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.86 | -1.0 | -1.0 | -1.0 | -1.0 |
| **5** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.5 | 0.8 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.84 | -1.0 | -1.0 | -1.0 |
| **6** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.67 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 | 0.86 | -1.0 | -1.0 |
| **7** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.89 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.9 | -1.0 | -1.0 | 0.84 | -1.0 | -1.0 | 0.0 | 0.875 | 0.8 |
| **8** | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -0.36 | -1.0 | -1.0 | -0.57 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.67 | -1.0 | -1.0 | -1.0 | 0.0 | 0.875 |
| **9** | 0.75 | 0.84 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.75 | -1.0 | -1.0 | -1.0 | 0.86 | -1.0 | -1.0 | 0.75 | -1.0 | -1.0 | -1.0 | -1.0 | -1.0 | 0.0 |

**Table C.2:** Activity Matrix created by ProM based on logs from a modified Markov model

# Appendix D
# Content of the CD

| | |
|---|---|
| `DataAcquisitionAnanlysis` | The folder with .NET Core application Data Acquisition Analysis. |
| `DataProcessing` | The folder with Python scripts. |
| `RoboticPrograms` | The folder with robotic programs. |
| `_petr_cezner_master.pdf` | The electronic version of this thesis. |