THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Towards Automatic Generation of Formal Models for Highly Automated Manufacturing Systems

ASHFAQ FAROOQUI

Department of Electical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2018

Towards Automatic Generation of Formal Models for Highly Automated Manufacturing Systems

ASHFAQ FAROOQUI

Technical report number: R006/2018
ISSN 1403-266X

Department of Electical Engineering
Chalmers University of Technology
SE–412 96 Göteborg
Sweden
Telephone + 46 (0)31 – 772 1000

Typeset by the author using LaTeX.

Chalmers Reproservice
Göteborg, Sweden 2018

to those who seek

Abstract

The manufacturing industry is undergoing a digital revolution, often referred to as Industry 4.0. The aim of this revolution is to transform the factories into, so called, *smart factories*. These smart factories will be modular, decentralized, and interconnected, to achieve higher level automation and flexibility. Additionally, a smart factory will have a *digital twin*, a virtual replica that allows testing, monitoring, and visualization of the factory behavior. As these factories are aimed to be completely automated, ensuring correctness and safety of the control logic in each sub-system of the factory is of utmost importance.

The need for having digitalized tools that support operators and engineers was identified in a survey that was conducted to understand the problems faced during maintenance of manufacturing systems. To this end, this thesis provides an architecture that can be applied on old legacy systems as well as new state-of-the-art systems to collect data from the factory floor. The data obtained can be visualized in the form of Gantt charts to help operators keep track of the execution of the station. Furthermore, a model that captures the behavior of the system can be created by applying Process Mining algorithms to the collected data.

Model-based techniques have shown to be beneficial in developing control logic for highly automated and flexible manufacturing systems, as these techniques offer tools to test and formally verify the control logic to guarantee its correctness. These formal tools operate on such a model of the behavior of the system. However, manually constructing a model on which these tools can be applied is a tedious and error prone task, seldom deemed to be worth the effort. Thus, supporting engineers to build models will improve the adoption of formal tools within the manufacturing industry.

In order to obtain a formal model during the early development phase of the manufacturing system, this thesis studies the possibility to automatically infer a model of a system by interacting with its digital twin. The suggested L^+ algorithm, an extension of the well-known L^* algorithm, shows that it is possible to automatically build formal models in this way. Additionally, certain shortcomings are identified and need to be addressed before being able to these methods in a practical setting.

Keywords: Formal Methods, Industrial Automation, Automata Learning, Visualization, Operations

Acknowledgments

When I embarked on this long and lonely journey exploring the realms of academia as a PhD student, I had not anticipated the way it would transform me. Now, here I am at the half-way mark eager to move ahead; but stopping to reflect upon my metamorphosis so far, and those that have contributed to this metamorphosis. Like in all journeys, the people encountered have left an impact; some more than others, but each profound and beautiful in their own ways. And to each, I owe a debt of gratitude. This journey would certainly not have been possible without the guidance, support, and patience of many important people.

First of all, I would like to thank my supervisor Martin Fabian. Your ability to ask the most pressing questions and provide blunt and honest feedback has helped me mature as a researcher. The detailed comments on the nitty-gritties of writing have helped me develop my skills as a writer, and for that, I am ever grateful. You have been an impeccable example of what a supervisor must be, not just for the guidance and support you provide, but for teaching me the what, why, and how, of supervision.

I would also like to thank my co-supervisor Petter Falkman for, firstly, for offering me this opportunity to enter academia. Secondly, and more importantly, for helping me keep my work anchored by helping me find use cases for all my ideas.

Håkan Pettersson from Volvo Cars Corporation was a great host during my short stint at Volvo Torslanda. Thank you!

My gratitude to Fredrik and Kristoffer for the insightful discussions that have helped change my perspectives at times I needed to most. To all the present and former members of the “discrete klubb” a big thank you for all the interesting discussions and support in different forms.

Stephen King says “The scariest moment is always just before you start,” I am grateful to Raghav, Charul, Martin (Viktorsson), Per-Lage, and Patrik who not just inspired me to embark on this journey but whose support helped me get over the scariest times.

When one has not written a thesis, the customary acknowledgments to the author’s family seems meaningless, but when one has spent long weekends and evenings absent from family activities, it reaches its full meaning. This journey

would have been impossible without the constant support of loved ones. My family, though far away, are a source of strength and support. I am forever grateful to my parents for being ever so supportive in everything I have done. Tania, my wife, has been by my side through the ups and downs of this journey. More importantly, she has helped me learn the beauty of balance; the balance that has helped sustain this journey; the balance that has made this journey even more beautiful. Thank you!

Lastly, a big cheers to the Free Software communities, chiefly, Manjaro – my platform for a few years now; Emacs – for being everything but a good text editor; And, LaTeX– for making writing tedious but beautiful!

Ashfaq Farooqui
Göteborg, August 2018

This work has been supported by Vinnova FFI VIRTCOM (2014-01408), ITEA3 VINNOVA ENTOC (2016-02716), VINNOVA LISA 2 (2014-06258), and VR SyTeC (2016-06204).

List of Publications

This thesis is based on the following appended papers:

Paper 1 **Ashfaq Farooqui**, Patrik Bergagård, Petter Falkman, and Martin Fabian. Error Handling Within Highly Automated Automotive Industry: Current Practice and Research Needs. *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA),* 2016, Berlin, Germany.

Paper 2 **Ashfaq Farooqui**, Kristofer Bengtsson, Petter Falkman, and Martin Fabian. From Factory Floor to Process Models: A Data Gathering Approach to Generate, Transform, and Visualize Manufacturing Processes. *Submitted for possible journal publication.* 2018

Paper 3 **Ashfaq Farooqui**, Petter Falkman, and Martin Fabian. Towards Automatic Learning of Discrete-Event Models using Queries and Observations. *Submitted for possible journal publication,* 2018

In what follows, these papers will be referred to as Paper 1, Paper 2, and Paper 3, respectively. The individual contributions of each paper are outlined in Chapter 4.

Other publications

The following publications, authored by the author of this thesis, are relevant but not included in the thesis:

Ashfaq Farooqui, Kristofer Bengtsson, Petter Falkman, and Martin Fabian. Real-time Visualization of Robot Operation Sequences. *2018 IFAC Symposium on Information Control Problems in Manufacturing (INCOM),* 2018, Bergamo, Italy.

Ashfaq Farooqui, Petter Falkman, and Martin Fabian. Towards Automatic Learning of Discrete-Event Models from Simulations. *14th IEEE Conference on Automation Science and Engineering (CASE),* 2018, Munich, Germany.

Contents

Part I

Introductory Chapters

Chapter 1

Introduction

The complexity of the manufacturing industry constantly increases to keep up with advancements in technology, market trends, legislative requirements, and most of all high quality products. Industry 4.0 [1], also called the *fourth industrial revolution*, can be seen as a collection of various technologies – Internet of Everything (IoE), Cyber-physical Systems (CPS), and smart factories – to create the next generation of industrial systems [2]. From the design principles of Industry 4.0 provided by Hermann et al. [2], distributed modular systems are key to build these next generation factories. The different distributed modules in an Industry 4.0 setting are normally provided by different manufacturers and have different properties, but when put together need to work seamlessly. In general, the systems developed consist of several industrial robots supported by conveyors and fixtures, and are designed to be completely automated with minimal manual handling. The system design must not only take into account flexibility, efficiency, and development time, but also account for fault tolerant behavior.

Development of these complex automated manufacturing systems is a demanding task. In the automotive industry, the development process typically begins with the company describing the product and design requirements. Based on these requirements, suitable components are chosen and an overall layout is decided on, followed by physically building the system. In parallel, the *control system* is developed that is to control the physical system so as to manufacture the desired product. Physically building the manufacturing station is commonly known as *physical commissioning*. This includes installation of the physical system consisting of robots, conveyors, fixtures, sensors etc., and the control system that is responsible for control, supervision and coordination throughout the production. The requirements specified are not always accurate and often ambiguous. Hence, the resulting station undergoes several iterations before it can be used in production. These iterations can significantly delay the time to market and increase costs, thus they are undesirable.

In order to reduce time to market and save on costs in the long run, the use

of *Virtual Commissioning* (VC) [3] is rapidly increasing today. With VC, a simulation model of the station is first created using some simulation software, such as Process Simulate from Siemens [4] or Experior from Xcelgo [5]. Then, the different manufacturing scenarios are simulated to check if all requirements are fulfilled. Furthermore, the control system can also be connected to the simulation model and its logic can be tested. This is done to find and fix faults and bugs in the control logic. Additionally, by visually simulating the manufacturing system, errors due to collisions can be detected early on. Physical commissioning is then done only after the VC model with the control function is acceptable. By correcting most of the faults during the VC phase, unnecessary time need not be spent testing the physical system. Thereby, production can start earlier.

However, both physical and virtual commissioning require the control logic, which is there to ensure that the station works as intended and is safe to operate. Thus, its correctness is of utmost importance. Since the requirements typically keep changing during the development process, the control logic needs to be continuously updated and debugged, which makes its manual development a tedious and error prone task.

One approach to manage this type of development process, with ever changing requirements, is to rely on mathematically well-defined *formal methods* [6]. By applying formal methods, it is possible to analyze and understand the system in part and as a whole. Formal methods make use of computerized calculations to analyze the system using a formal model, usually in the form of a *discrete event system* (DES) [6], that models the system to be analyzed. Hence, engineers can focus on defining their system and then use computerized algorithms to analyze, verify, and validate the models.

While the use of formal methods eases the engineering task of building the control logic, the burden instead gets placed on building the models. As a model grows in size with incorporating more resources and their operation, the modeling burden grows exponentially. To alleviate this, the models can be built *modularly*, where the engineers build models of the respective resources and their operations, and compose them in a mathematically well defined way into a model of the overall system. In addition, modeling the requirements as DES, the control logic can even be automatically generated by *synthesis* [6], so as to be correct-by-construction, which further alleviates the risk of introducing errors.

The discussion provided above focuses on building models for the control logic of new manufacturing stations. Building new systems are not done very often, though, while changes to existing legacy systems are done daily in order to improve their behavior, such as cycle time adjustment, bug fixing, etc., or on a bigger scale to introduce new products or machinery. To be able to do this in a secure way avoiding disruptions of existing behavior and the production on the whole, manufacturers are interested in applying formal analysis and to test

changes before commissioning them to the physical system. However, a problem here is that, more often than not, virtually commissioned or formal models do not exist for legacy systems.

Manually creating the required models is hard and time consuming, requiring a high degree of knowledge in formal methods and automation system design. Incorrect or incomplete models are misleading and the use of formal methods may serve no purpose. To benefit from the use of formal methods, in terms of verification and synthesis of control logic, the task of building models needs to be done correctly. In order to do so, this thesis proposes a way to automatically create a model of the system using computerized tools.

1.1 Research Questions

This thesis aims to explore the following questions.

RQ1 *How does the manufacturing industry generally handle errors and perform maintenance? And what are the challenges faced?*

Error recovery techniques have been studied in academia [7, 8, 9, 10, 11], but these techniques have not really found their way into industrial practices. Identification of reasons for this disconnect might help bridging the gap between academia and industry.

RQ2 *How can operators be supported with tools and processes that will make it possible to make more data driven decisions?*

Maintenance of manufacturing systems is not an easy task. Decisions need to be made, tracked, and evaluated. Manually doing so is burdensome and often ineffective. Having digital tools to support operators can improve the quality of maintenance [12].

RQ3 *Is it feasible to automatically learn formal models of manufacturing systems? If so, what would be required to make it a reality?*

Model-based techniques have shown to be useful for a variety of reasons. The lack of usable models and the difficulty to create them manually is a deterrent to using model-based techniques to build and maintain manufacturing systems. Automatic creation of formal models has been studied [13, 14], but these algorithms are computationally heavy and have typically been designed for smaller applications. Recent advances in computer technology have led to computationally powerful computers, and availability of advanced simulation tools, which might hold the key to making it possible to learn a formal model of large manufacturing systems.

1.2 Objective and Contribution

The objective of this thesis is twofold. First, to highlight the problems faced by the automotive industry related to maintenance and error handling. Then, to propose methods, techniques, and tools to help during the commissioning and production phase so as to increase quality and efficiency. This is achieved by the following contributions:

- A survey to identify the problems faced by the industry and how these problems are currently solved.
- A survey of ongoing state-of-the-art projects that provide a glimpse into future manufacturing technologies.
- An approach to collect and transform data from the factory floor.
- Analysis and visualization of ongoing manufacturing processes in real-time using Gantt charts, and creation of a behavioral model using Process Mining.
- A study into methods that enable automatic generation of formal models from a simulation model of the system using the L^+ algorithm. The insights gained from this study will help identify avenues that will allow automatic creation of models for practical manufacturing systems.

The papers presented in this thesis build upon the ideas presented in this introduction. Paper 1 highlights the common problems faced by the automotive manufacturing industry with regard to error handling and presents work needed to address these problems. An outcome of this study was the need to have access to data from the factory floor to get a better understanding of the systems. Paper 2 presents a flexible and scalable architecture to collect and transform data from the factory floor. This architecture can be applied to existing and new manufacturing stations. The resulting data is then transformed into a more understandable abstraction in the form of operation descriptions. This data is then used to build models of existing manufacturing stations using process mining techniques to help improve and maintain the stations.

Paper 3 presents the possibility to build a formal model of a system from its simulation model. The functions of the simulation model are modeled as independent operations which can be executed from an external interface. The L^+ algorithm, an extension of the L^* algorithm [13], is interfaced to the simulation software, where it can observe the state of the simulation. Then, by posing queries the algorithm constructs a formal model capturing the behavior of the system.

1.3 Method

The outcome of this thesis is a set of activities and tools that can help build more reliable and efficient manufacturing systems. Most of the work involved implementing and testing the algorithms to demonstrate the applicability of the proposed methods. From a more theoretical perspective, the main research activity was to identify gaps in the field of automated modeling that will improve the practical applicability of these methods.

In general, the method followed was to first, by interacting with industry representatives find pressing problems that are faced by the industries; problems related to error handling and, more specifically, modeling of automated manufacturing systems, were considered. Then, solutions to these problems were proposed by suggesting a method – as a set of activities – to follow, along with accompanying algorithms. These algorithms were then implemented to demonstrate and evaluate the possibility of applying them to real-world scenarios. The evaluations showed that it is indeed possible to apply in practice the presented approach. However, there are also some parts missing and parts that need to be improved. These are summarized in Chapter 5.

1.4 Outline

This thesis is divided into two parts. This first part contains introductory chapters that aim to help the reader to better understand the ideas and concepts discussed in the included papers, and also to provide a direction of the research work. The second part contains the included papers.

This introduction presents a high-level process of building a manufacturing station and highlights the problems normally faced by engineers involved with building and maintaining these stations. Chapter 2 puts the thesis into perspective by positioning this work in the broader picture. To this end, existing industry practices and challenges are presented, followed by a new updated work-flow that aims to support engineers and operators during the development of a manufacturing system. Chapter 3 introduces the reader to the field of automatically inferring formal models. Chapter 4 provides a summary of the included papers. Finally, some concluding remarks in Chapter 5 sum up the work and provide a glimpse of future work.

Chapter 2

The Broader Picture

The aim of this chapter is to provide a context and help position the contribution of this thesis within the broader picture. In doing so, this chapter will give a general overview of existing methodologies followed during the development of a manufacturing system, and some of their challenges. Additionally, this chapter, will outline the new methodology that is proposed to help mitigate the challenges. The tools required to achieve the suggested methodologies in reality are also briefly introduced.

The traditional procedure followed during the development of a manufacturing system starts with a pre-study on the product that will be manufactured. The pre-study results in a set of requirements that need to be fulfilled by the manufacturing system; a bill of materials that contains the components, such as, robots, conveyors, fixtures, etc, that need to be procured; and the system layout representing the physical placements of those components. The bill of materials is procured and the physical building of the system is started, in smaller functional parts or as a whole. Based on the requirements, different tasks and actions needed to manufacture the product are planned. This is known as *process planning* [15, 16]. The end result of the process planning stage is the generation of the control logic. This control logic is responsible for controlling the resources in the manufacturing system so as to fulfill the objective of the manufacturing system in an efficient and safe manner. The control logic then needs to be tested on the physical system, which is usually done in two phases. In the first stage, components are tested individually or in groups. The second stage is performed after the system is physically commissioned. Here, testing is done until all requirements are satisfied, and the system behaves satisfactorily.

The consideration of control logic later in the development phase has several negative consequences resulting in high financial costs and loss of time. As seen in Paper 1, fixing software bugs might, as a side effect, introduce more bugs. Additionally, performing the tests on the physical system increases the risk of collisions and component failures. During the survey conducted in Pa-

per 1 it was also found that the initial testing done at the line manufacturers site was functional testing. Later, only when the complete system is physically commissioned, complete end-to-end testing is performed. Directly testing on the physical system might result in breakage and may require a fresh order of components. All these factors add to the unnecessary effort, cost, and time expended during the physical commissioning phase.

Development of manufacturing systems is just the first phase of its life-cycle. Building these stations is expensive, and the manufacturing company expects them to last for several years to even out the cost. Hence, these systems need to be well maintained to ensure good productivity during their life-cycle. Paper 1 presents the current industrial state of maintenance procedures. The concluding points in Paper 1 are to have a work-flow that incorporates ways to fix bugs in the software safely, and a possibility to digitalize the system to be able to use digital tools to monitor, visualize, and reason about it. Hence, the tools and processes created in the new work-flow need to not only account for a better manufacturing process but also need to cater to maintenance requirements.

2.1 The New Workflow

The Division of Systems and Control, at the Department of Electrical Engineering at Chalmers University of Technology, has been active in developing a work-flow and tools to support the development of manufacturing systems. A high level work-flow that is pursued at the department is presented in Figure 2.1. This new work-flow starts similar to the old work-flow by identifying the bill of materials. But the process planning starts already at the initial phase along with the bill of materials to identify the process, tasks and actions needed, resulting in the *bill of processes*. Based on the bill of processes, the components identified are digitally created in a 3D simulation environment. These digital replicas are placed according to the required system layout. At this point, potential collisions can be identified, and the layout can be updated to ensure safe operation. The engineers then take the bill of processes and convert this to actions to be performed in the simulation software. These simulated actions need to be transformed into control logic that can be run on a Programmable Logic Controller (PLC). The system is *virtually commissioned* by connecting the PLC to the simulation tool to verify functionality and test for errors. At this point, the bill of materials is finalized, and the components are procured. Following which the system is physically commissioned.

During the development phase, requirements are constantly changing due to new insights. Hence, the control logic needs to be constantly updated and tested to ensure that the requirements are met. One approach is to use mathematically well-defined formal methods to verify the behavior of the system and to generate

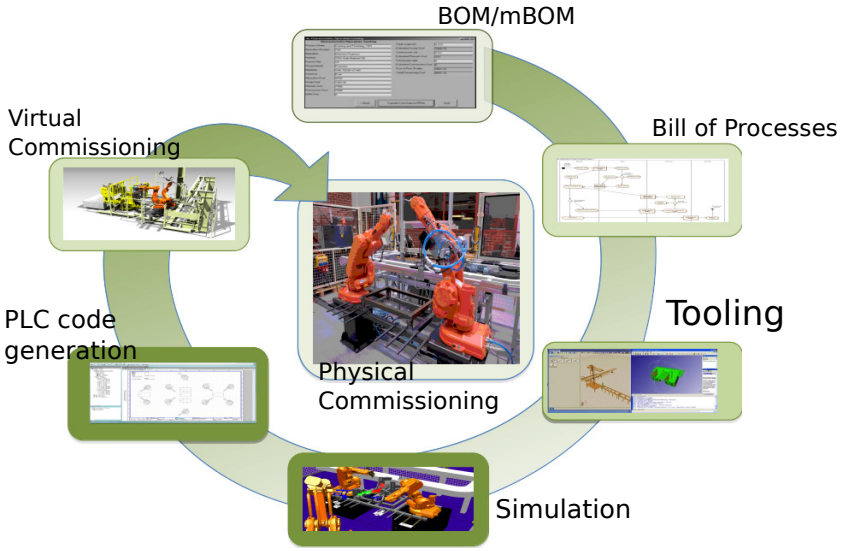


Figure 2.1: The new workflow

the control logic. With the use of formal methods, the focus is shifted from manually developing and testing the control logic, to developing a model that describes the behavior of the system.

To build a formal model the engineer needs to consider the level of abstraction needed to be described by the model. By modeling at a very low level the engineer risks making the model unmanageably large. On the other hand, by modeling at a high level, a number of details could be missed, and the model might not be usable. Hence, finding a level of abstraction that suits the system is key to developing a model. Once a suitable level of abstraction is decided, the engineer has to ensure that the model correctly captures the behavior. This is where this thesis contributes towards the development of manufacturing systems. It looks at possibilities to automatically infer a behavioral model of the system.

A by-product of the defined work-flow is the availability of a *digital twin* – a digital replica of the physical system. This digital twin behaves in the same way as its physical counterpart and can be used for monitoring, testing, and modifications. To enable the effective use of a digital twin, additional tools need to be created that can synchronize between the physical system and its digital twin. These tools need to capture and store operating data from the physical system for further processing.

The remainder of this chapter will introduce the different components needed to follow this thesis, specifically, virtual commissioning and formalisms for formal methods.

2.2 Virtual Commissioning

Simulation technology has come to a point where it is now possible to simulate systems at the sensor level. Software programs such as Process Simulate [4], Xcelgo Experior [5], and ABB Robot studio [17] are capable of simulating conveyors, fixtures, robots, and even humans, to a relatively high level of accuracy.

These simulation tools can also be controlled externally, for example, by using a PLC. Such a setup would constitute *virtual commissioning*, where a digital replica of the system is controlled by the PLC code that will eventually run on the shop floor controlling the physical system.

In order to be able to infer a model of the system, which is the aim here, the learning algorithm – the learner, introduced later in this thesis – requires an interface to the virtually commissioned system. This interface must allow the learning algorithm to execute actions which are at a pre-defined abstraction level, and allow the learning algorithm to observe the output of the virtual model. An example of such an interface is the OPC-UA [18] standard for communication. This standard is supported by many component vendors and can be used with the virtual and physical system. The OPC-UA, allows control and observation at the input/output level, and also facilitates observation of internal variables. Hence, it is an ideal candidate to use as an interface for the learning algorithm.

2.3 Modeling Operations

As discussed in Section 2.1, choosing a sufficiently moderate level of abstraction becomes important to create a model. The work in this thesis uses the abstraction of *operations* [19]. An operation is a task performed in a manufacturing system by one or more resources. Broadly, an operation can be defined as a set of actions that change the state of the system to accomplish an objective. The level of detail of an operation is not fixed, it could be used to define an update in one resource or can affect several resources. *Guards* are used to determine when a operation is allowed (or not allowed) to execute. These operations, along with their guards, are programmed such that they can be executed by the control system. A more formal definition is provided in Paper 3, Section 2.3.

2.4 Formal Methods

Formal methods are design techniques that use mathematical models to build software and hardware systems. These methods make use of mathematical proofs to ensure correctness. As systems become more complicated and safety becomes an important concern, a formal approach to the system design offers a certain

level of insurance.

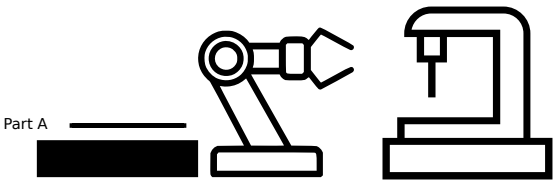
To apply formal methods, designers need to have access to a formal model that describes the behavior of the system. One of the many ways to create such a model is to make use of *finite-state machines* (a.k.a *finite-state automata* or simply *automata*) [20], which are commonly used to model discrete-event systems. The system is then abstracted into *events* and *states*, where the occurrence of an event moves the system from one state to another. The original state of the system, before the occurrence of any event is the *initial state*. A particular sequence of events can lead the system to some desired state, such a desired state is called an *accepted state* and the sequence of events an accepted sequence.

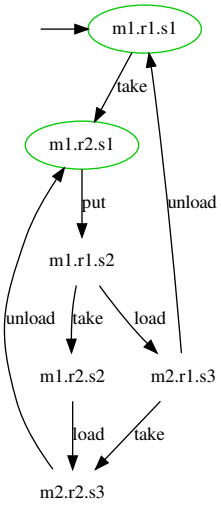
When talking about discrete-event systems as finite-state machines a sequence of events is referred to as a *word*. The set of words that are accepted by the finite-state machine is the accepted *language*. The automaton represents the *grammar* of the system; that is, when presented with a word, the grammar decides the validity of this word. Formal definitions of words and languages are provided in Paper 3, Section 2.

Chapter 3

Inference of Formal Models

Consider a small but realistic example consisting of a robot and a machine, as seen in Figure 3.1a. In this system, the robot takes the part (part A) from the pallet and puts it on the machine. The machine then loads the part, processes it, and unloads it. In the meantime, the robot can take the next part or do nothing. But the robot has to wait until the machine has unloaded its part before putting the next part on the machine. An automaton representing this setup is shown in Figure 3.1b, from which it is seen that the system is accepting as long as there is no part loaded in the machine.





(a) Representation of the physical system. (b) Representation of the system behavior.

Figure 3.1: A robot and a machine. The robot takes a part and puts it on the machine. The machine loads the part, and after processing unloads it.

Assume now that we do not have the model of this system, and want to obtain the model without manually building it. That is, we are provided access to the

actual physical system as in Figure 3.1a and would like to infer a model of it as seen in Figure 3.1b. There exist tools and techniques that would help to infer such models. Broadly, models can be inferred using *Grammar Inference* [21] or *Process Mining* [14] techniques.

The aim of this chapter is to provide a glimpse into existing ideas that deal with automatically inferring models. Both the fields of study, Grammatical Inference and Process Mining, are well established and quite vast. It is beyond the scope of this thesis to cover the details of the different algorithms. However, the general ideas employed in each of the fields will be discussed to equip the reader with the basics concepts. This will help interested readers to further delve into the details.

3.1 Grammar Inference

Grammar Inference (GI) has been studied both as a theoretical problem, where its goal is to uncover some hidden function, or as a practical problem of attempting to represent some knowledge as an automaton. GI finds its origin in various fields of study: computational linguistics, machine learning, formal learning theory, pattern recognition, and computational biology. Hence, it is also known by different names depending on the field: Automata Learning, Grammar Induction, Grammar Learning, etc. Though the different names can have different connotations, they all refer to similar ideas and processes.

The majority of GI algorithms work on generalizing some form of knowledge about the system to be learnt. The learning algorithm referred to as the *learner*, internally creates a representation of this knowledge. This representation is continuously refined and generalized to satisfy certain properties specific to that learner. When the learner is satisfied with the generalization reached, its internal representation can be converted into some meaningful representation. Of the many methods studied and developed under GI, several of them deal with learning finite-state machines. These methods can be classified as *Passive learning* or *Active learning*.

In-depth surveys of GI techniques are provided by [22, 23, 24].

3.1.1 Passive Learning

Passive Learning, sometimes referred to as Informed Learning, is a setting in which labeled data is provided to the learner, and the learner is tasked to find an automaton that generalizes this data. This data is usually an observation log from the system and consists of words. These words are labelled as accepted (or rejected) if they belong (or do not belong) to the accepted language.

Passive learning algorithms start by first generating a hypothesis from the available data. This is done by constructing a *prefix tree acceptor* (PTA), which is a tree-like automaton constructed by looking at the accepted words in the data. This PTA can be constructed in linear time and contains no loops or converging paths. In the example above, accepted words could be the set $\{\langle$take$\rangle$, $\langle$take, put, load, unload$\rangle$, $\langle$take, put, take, load, unload$\rangle\}$, and non-accepting words could be $\{\langle$take, put$\rangle$, $\langle$take, put, load$\rangle$, $\langle$take, put, take$\rangle\}$. The first step is to construct the PTA using the accepted data. The PTA obtained is a crude representation of the available data as seen in Figure 3.2.

The next step is to refine the PTA. The method of building the PTA and then refining it using non-accepting data samples is called *regular positive and negative inference* [25]. This method provides a basis for more advanced algorithms that focus on the refinement phase. A number of different strategies for PTA refinement have been presented in the literature [21, 26, 27, 28]; a detailed survey of those is beyond the scope of this thesis.

The most common refinement technique, though, is *state merging* [29]. The merging of states usually consists of three stages, the first is the search to identify the states to be merged; then the actual merging, where the merged states are represented by a single state yet retain the incoming and outgoing transitions; the final stage is called *promotion*, which keeps track of the states already tested and the next states to be evaluated. Several algorithms have been suggested to efficiently perform the search, merging and promotion. In [27] a general survey of state merging algorithms is presented where they are classified as *Exact Algorithms* or *Approximate Algorithms*.

Exact algorithms aim to create a model that exactly represents the input data. Some such algorithms are MMM [30], BICA [31], and EXBAR [32], all of which start with a basic trivial model as initial PTA. The algorithms then start looking for states to merge. On every merge done the algorithms search the input data to check if the samples are consistent with the automaton. Searching through the input data for inconsistencies is expensive. The algorithms mentioned above use different techniques and heuristics to perform this search efficiently.

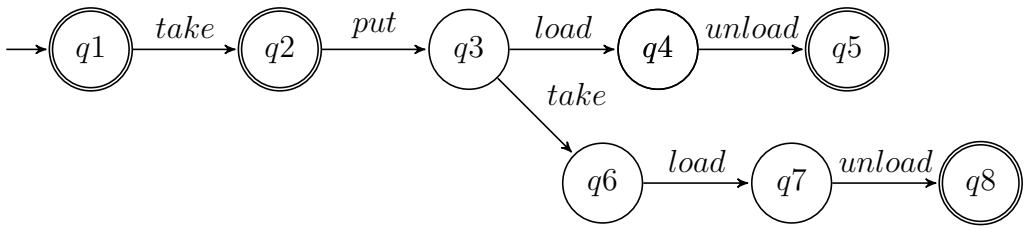


Figure 3.2: The PTA generated from the set of accepted words $\{\langle$take$\rangle$, $\langle$take, put, load, unload$\rangle$, $\langle$take, put, take, load, unload$\rangle\}$

Approximate algorithms, on the other hand, rely on heuristics to provide an automaton that approximately describes the input data. Examples of these algorithms are EDSM [33], SAGE [34], and ED-BEAM [32]. These algorithms work in similar ways to their exact counterparts, the difference lies in the way they perform the state merging. Additionally, these algorithms keep updating the PTA by backtracking and improving previous merges.

In order to obtain an accurate model of the system, the learner needs to have access to accepting and non-accepting data [35]. In a manufacturing context access to accepting data is usually never a problem. However, there does not seem to exist ways to obtain non-accepting data. Hence, the models identified using passive learning methods are limited to the fact that they represent only the observed behavior. One major problem when applying passive algorithms is the state-space explosion. It has been proved that learning from sampled data is NP-complete [35]. Thus, most of the algorithms are created and tested on problems with a significantly smaller state-space [27] compared to what is needed in a manufacturing setting.

Paper 3 uses ideas from passive learning in conjunction with active learning methods to obtain a more accurate model of the system.

3.1.2 Active Learning

Active Learning, or learning with queries, is a setting where it is possible to interact with an oracle. The algorithms available under active learning make the assumption that there is a *minimal adequate teacher*, which is an oracle that can answer queries. An active learner poses queries to the oracle and based on the responses constructs a model.

The analogy of a teacher and student fits well to explain the working of general active learning techniques. The student is the learner, and the teacher is the oracle. The student is given prior knowledge about the different events possible, in the example above it would be the set {take, put, load, unload}. And, the student is allowed to pose two types of queries to the teacher. The first type is about the membership of a given sequence of events; to this the teacher can respond positively if the sequence results in an accepted state, else the response is negative. The second type of query, called equivalence query, occurs when the student presents a hypothesis model to the teacher. If the teacher responds positively, that is, acknowledges that the model learnt by the student represents the actual system fairly accurately, then a model is found and the learning terminates. Else, in case the teacher finds the model incorrect, the student is presented with a counterexample – a sequence of events that is allowed in the hypothesis but not in reality, or vice versa. The student then updates its model to exclude or include the behavior of the counterexample, and continues asking queries. This

process iterates until a fairly accurate model is found.

For the sake of an example, lets say the student starts by asking if ⟨take⟩ is a member; the teacher responds positively. If the student then proposes a model with a single event, then since the model is invalid the teacher provides a counterexample by giving, say, the sequence ⟨take, put, load, unload⟩ The student will then take the presented counterexample into consideration, make new queries, and eventually present a new hypothesis. This process continues until the teacher decides that an acceptable model is found. The procedure highlighted above is intended only to provide an analogy into the actual learning process and to show how this type of learning is closer to how humans learn. The actual algorithms are far more complex. One of the most fundamental active learning algorithms is the L^* algorithm introduced by [13], a detailed explanation for which is provided in Paper 3, Section 4. This algorithm is known to run in polynomial time and guarantees termination [13].

The L^* algorithm has been a starting point for most of the research in the field of active learning [13]. From an algorithmic perspective, there have been only a handful of improvements and new approaches suggested. Schapire et. al. [36] improve the L^* algorithm by handling counterexamples that include a homing sequence when it is not possible to reset the target system. In the case when a robot is allowed to explore its surroundings to infer a map, it is tedious to always reset the robot to its initial position. Hence, Schapire et. al [36] present an algorithm that creates a homing sequence. Using this homing sequence allows the robot to infer its current state. Kearns and Vazirani [26] introduce the idea of discrimination trees to internally represent the knowledge of the system. The idea of discrimination trees is further explored by Malte et al. [37] who suggest the TTT [1] algorithm.

The L^* algorithm has been used successfully in practical settings to learn automata. Active automata learning has been applied to verify communication protocols using Mealy machines [38, 39]. By using a suitable abstraction interface, Arts [40] learn IO automata. Other techniques are directed towards learning models of software systems; Malte et al. [41] apply active automata learning towards learning models of software programs modeled as register automata, while Smeenk et al. [42] focus on learning embedded software programs.

The active learning algorithms assume the existence of an oracle. If a computerized oracle would exist, then there would be no problem to solve, since the model would be available in the oracle, and the task would then be to extract the model from the oracle. However, with the advancement of technology and easy availability of simulation software, it is possible to create a digital twin of the actual system to play the part of an oracle. These simulations are built up of sev-

[1] The name is derived from *Spanning Tree*, *Discrimination Tree*, and *Discriminator Trie*; the three concepts fundamental to the algorithm.

eral functions that can be executed using an external interface to the simulation software. For example, the event "take" would have a function that makes the robot pick the part from the pallet, or "put" would place the part on the machine, and so forth. The teacher is then no longer an oracle, but an interface to this simulation. It can then reply to queries by simulating the given sequence. The accepted states are then identified by observing the simulation, which could be achieved by reading the internal variables of the simulation and defining a predicate expression over these variables. This setup is further elaborated in Paper 3, Section 6.

However, in order to be able to apply these techniques to learn models there needs to be a way to find counterexamples. Queries for membership are fairly easy to handle, equivalence queries on the other hand are proved to be an NP-complete problem [43]. Hence, finding counterexamples is a bottleneck for active learning methods. In the case when the model of an existing system is to be created, which also has a virtually commissioned counterpart, Paper 3 integrates active and passive learning methods and presents the L^+ algorithm that uses observation data from the actual system to find counterexamples.

3.2 Process Mining

Process mining [14] is a field of study comprised of process discovery, conformance checking, and model enhancement. Process discovery, similar to passive learning, aims at creating a process model that imitates the observed process. The difference between process discovery and passive learning lies in their objectives. While process discovery takes a more pragmatic approach towards building and analyzing the underlying process in large organization, passive learning is a more theoretical method focusing on finding grammars that represent the data. What sets process discovery apart from passive learning is its integration with the other components of Process Mining, conformance checking and model enhancement, for investigating and maintaining the process models.

Process discovery techniques are used to understand task flows in large organizations where tasks can be performed for various resources or individuals. To do this, these techniques rely on process logs as input. The minimal requirement for the logs is a tuple containing `<caseId, event name, attributes>`. Here the `caseId` is a unique identifier that identifies the case being handled; in a manufacturing context it could be viewed as the product on which operations are run. The `event name` identifies the task executed. A number of other `attributes`, such as time stamps, the name of the resource that performs the operation, product variant, can be appended to improve the analysis. The resulting output from a discovery algorithm is a work-flow model that can be represented by formalisms such as Petri nets [44], Transition

Graphs [45], or Business Process Model and Notation diagrams [46].

The basic idea, in almost all Process Mining algorithms, is to construct a relationship table from the input logs. The relationship table contains information about the position of each event relative to the other events. In the example above, irrespective of the actual sequence of events, every product will log the sequence ⟨take, put, load, unload⟩. The relationship table will thus reflect a "directly follows" relation between the pairs ⟨take, put⟩, ⟨put, load⟩, and ⟨load, unload⟩. Thus, the resulting work-flow model is a straight sequence of these events. In a more complex setting, where there are different paths observed, the relationship table can additionally describe relations such as parallel or arbitrary.

The process outlined above closely resembles the alpha algorithm [47], the most basic algorithm in Process Mining. However, the alpha algorithm is not immune to noise in the logs, and also cannot be applied to systems that contain loops in their execution. Several advanced Process Mining algorithms have been presented that tackle these problems. The Heuristic Miner [48], for example, builds a relationship table that contains the number of times an event occurs before or after other events. Then, using heuristics to normalize this table, the algorithm constructs a model representing the different relationships. Another such algorithm, the Fuzzy Miner [49], aims to find closely related patterns and group them into clusters. In doing so, it becomes possible to not focus on the details, thus an abstract process is presented for possible further analysis.

A model defining relations between events is not the only form of output achievable from Process Mining. It is also possible to get a model from various perspectives and analyze different properties. For instance, by logging the execution time and executing resource for each event as additional attributes it is possible to analyze the system for bottlenecks and resource utilization. To this end, a model representing the relationship between the resources is first created. Then by aggregating the time spent at each resource, this model can include the time each resource is occupied.

Process mining has shown significant benefits in understanding underlying task flows, bottlenecks, resource utilization and many other factors within large corporations [50, 51], and has also proved beneficial in health-care [52, 53, 54] to learn and improve the underlying processes. Within the manufacturing domain, though, there has been only a handful of studies of applying Process Mining. Yang et. al. [55] and Viale et. al. [56] present a method to apply Process Mining on manufacturing data. While the former uses structured and unstructured data generated from a manufacturing system along with operators or workers to provide domain level knowledge, the latter works with definitions of the system provided by domain experts to find inconsistencies between the model and the actual process. Yahya [57] shares interesting insights into using process mining to understand manufacturing systems using artificially created logs. However,

there seem to be two main factors missing when aiming to apply Process Mining to manufacturing systems:

1. There is a lack of methods to capture and collect usable data from the factory floor.

2. There is a lack of a defined data structure/abstraction useful for generating models from factory data.

Paper 2 uses Process Mining to analyze manufacturing systems by collecting data from the factory floor. Section 2 of Paper 2 provides a general architecture that generates an event stream of program pointer data from the robots in a manufacturing system. Furthermore, the paper elaborates on transforming the generated event streams into the usable abstraction of *operations*. This abstracted data is used as input to Process Mining algorithms to obtain models and analyze the system. Section 4.3 of Paper 2 discusses visualization from an operation perspective, where the relations between different operations are studied to identify unique patterns that correspond to different products. Section 4.4 then shows the use of Process Mining to visualize how the product flows between the resources.

Chapter 4

Summary of Contributions

This chapter provides a brief summary of the papers that are included as part of this thesis.

Paper 1

> **Ashfaq Farooqui**, Patrik Bergagård, Petter Falkman, and Martin Fabian.
> Error Handling Within Highly Automated Automotive Industry: Current Practice and Research Needs. *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA),* 2016, Berlin, Germany.

This paper presents a study on error handling in the Swedish automotive manufacturing industry, specifically the body-in-white segment. To this end, a survey was conducted with several industry partners to get a glimpse into what were the most common errors encountered and the measures taken to avoid them. Additionally, the paper looks at ongoing research that is aimed towards making manufacturing highly automated. Based on the survey with industrial partners and ongoing research, the paper identifies future directions of work that will help industries handle error scenarios. Tool breakage, resource malfunctions, software bugs, and emergency stops were a few of the most common issues faced in the industry. To be equipped to handle these problems operators need to be well trained. Therefore additional effort is required to ensure training handles the standard scenarios. A common need towards alleviating the problems identified is the digitalization of the system and its processes.

Paper 2

Ashfaq Farooqui, Kristofer Bengtsson, Petter Falkman, and Martin Fabian.
From Factory Floor to Process Models: A Data Gathering Approach to Generate, Transform, and Visualize Manufacturing Processes. *Submitted for possible journal publication.* 2018

This paper provides a software architecture to collect data from robot manufacturing stations. The suggested architecture was designed to be applicable to existing and new manufacturing stations. The collected data is abstracted into the form of operations and can be visualized in real-time aiding the operators. To show the applicability of the data, the software architecture was applied and tested on manufacturing stations consisting of several robots. Then, process mining methods are used to process the obtained data to construct a general model that represents the activities of the station. Process mining algorithms are also used to identify the relationship between the resources in the station.

Paper 3

Ashfaq Farooqui, Petter Falkman, and Martin Fabian.
Towards Automatic Learning of Discrete-Event Models using Queries and Observations. *Submitted for possible journal publication,* 2018

This paper presents an approach to integrate active and passive learning by introducing the L^+ algorithm, an extension to the L^* algorithm. The L^* algorithm, presented in Paper 5, shows the possibility to learn formal models from a simulated environment. A major bottleneck to practically use the L^* on manufacturing stations relates to finding counterexamples. The approach presented in this paper, specifically the L^+ algorithm, uses previously collected sequential operation data to find counterexamples to a model created by the L^* algorithm.

Chapter 5

Concluding Remarks and Future Work

Building correct and error free control logic for manufacturing systems is a challenge. Mathematically well-defined formal methods provide the possibility to analyze and understand the system. These formal methods can ease the task of building control logic by focusing on building models that define the behavior of the system, and analysis can then be performed on these models to verify that given requirements are guaranteed to be fulfilled. Doing so allows the engineer to focus on the behavior and not the underlying details of the system. However, the models grow exponentially as the number of resources and operations performed by them increases, and this makes it hard to manually create models. Hence, it would be beneficial to be able to automatically build models that can then be used by engineers to perform analysis by applying formal methods.

This thesis focuses on automatically building models of manufacturing systems to help during the maintenance stages, as well as during the early virtual commissioning phase. To help operators doing maintenance, the current state of the manufacturing system needs to be tracked and visualized in a way understandable by the operators. To this end, an architecture to collect and transform execution data from robots in manufacturing stations is presented in Paper 2. This data is used to aid operators, by visualizing the execution of the system, to better understand and service the station in a timely manner. Furthermore, the logged data is also used to infer a model that describes the behavior of the system using Process Mining approaches. This model describes the relationship between the different operations in the system, and the general product flow between the resources.

The possibility to collect data and build a model in real-time creates many more avenues to support operators. Live data from the manufacturing system can be replayed and simulated in the model, in doing so any behavior not already present in the model can be captured and presented to the operator for inspection.

Similarly, the generated data can be used to continuously update the model, that is, build the model in real-time. This live model can be verified against the requirements of the manufacturing system to find deviations. Furthermore, these models can be used to perform predictive maintenance on the resources, predict delays, predict the throughput, thereby supporting the engineers and operators.

Another area that this thesis focuses on is learning of formal models from a digital twin. Engineers can couple the digital twin with active learning during the development process to analyze the effects of changing requirements. By defining executable operations – sets of actions – that perform a specific task, the learner is tasked with inferring a complete model that describes the behavior of the simulation model. This helps in building flexible manufacturing systems in which the engineers need not spend additional effort to manually build models for every product variant. Thus, the learner can infer models for the different products, using the virtual model, based on their requirements.

In conclusion, this thesis studied problems faced during error handling in the manufacturing industry. Certain key problems were identified, and the remainder of the thesis presented tools and techniques to tackle them. The presented approaches were tested and validated on small systems and toy examples to show a proof of concept. However, to be able to fully use these benefits in an industrial setting, the algorithms need to be improved and implemented.

Future work

To be able to fully realize the work presented in this thesis and apply it in real demonstrations, there are several challenges that need to be handled. The state-space explosion problem is one of the major challenges faced while trying to learn models automatically. An approach to alleviate this challenge is to use richer formalism to describe models; one such formalism is Extended Finite Automata [58], an extension of finite-state machines that is built up of bounded discrete variables and uses guards and actions to read and update variables while executing transitions. In order to automatically generate guards for an Extended Finite Automaton, it might be worth to investigate if Binary Decision Diagrams [59] can be used to encode state information in the observation tables. Then, by consolidating the state information in a smart way, to generate guards for each transition.

Concerning generation of data from the factory floor as presented in Paper 2, the current work is limited to gathering data from industrial robots. However, most of the logic in complex systems is contained within the PLC. Therefore, generation of execution data from PLCs into a reasonable abstraction is the next natural step.

In a more broader context, a major challenge is to reason about the quality

of the obtained models. Models are not perfect; they represent only a part of the reality. But to be able to reason and make decisions by using such models, there needs to be some metrics to help classify them. These metrics can provide the engineer with a degree of confidence when performing analysis on the model, and also help reason about the meaning of these results in relation to the physical system.

Bibliography

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, 2014. [Online]. Available: http://dx.doi.org/10.1007/s12599-014-0334-4

[2] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan 2016, pp. 3928–3937.

[3] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *Journal of Computational Design and Engineering*, vol. 1, 2014.

[4] "Process Simulate." [Online]. Available: https://www.plm.automation.siemens.com/global/en/products/tecnomatix/assembly-simulation.html

[5] "Xcelgo Experior." [Online]. Available: https://xcelgo.com/experior/

[6] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[7] P. Loborg, "Error recovery in automation an overview," in *AAAI-94 Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems, Stanford, Ca, USA*, 1994.

[8] P. Loborg and A. Törne, "Manufacturing control system principles supporting error recovery," in *Proceedings of the AAAI Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems, Palo Alto, CA, USA*, vol. 2123, 1994.

[9] Z. Gao, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques ;part i: Fault diagnosis with model-based and signal-based approaches," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3757–3767, June 2015.

[10] B. Vogel-Heuser, S. Rösch, J. Fischer, T. Simon, S. Ulewicz, and J. Folmer, "Fault handling in PLC-based industry 4.0 automated production systems

as a basis for restart and self-configuration and its evaluation," *Journal of Software Engineering and Applications*, vol. 9, no. 1, p. 1, 2016.

[11] P. Bergagård, M. Fabian, P. S, and K. Bengtsson, "Implementing restart in a manufacturing system using restart states."

[12] A. Farooqui, P. Bergagard, P. Falkman, and M. Fabian, "Error handling within highly automated automotive industry: Current practice and research needs," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 9 2016.

[13] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87 – 106, 1987.

[14] W. van der Aalst, *Process Mining*. Springer Nature, 2016.

[15] T.-C. Chang, *Expert process planning for manufacturing*. Addison-Wesley Longman, 1990.

[16] H. Marri, A. Gunasekaran, and R. Grieve, "Computer-aided process planning: a state of art," *The International Journal of Advanced Manufacturing Technology*, vol. 14, no. 4, pp. 261–268, 1998.

[17] "ABB Robot Studio." [Online]. Available: https://new.abb.com/products/robotics/sv/robotstudio

[18] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[19] K. Bengtsson, B. Lennartson, and C. Yuan, "The origin of operations: Interactions between the product and the manufacturing automation control system," *IFAC Proceedings Volumes*, vol. 42, 2009.

[20] J. E. Hopcroft, R. Motwani, Rotwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computability*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[21] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[22] ——, "A bibliographical study of grammatical inference," *Pattern Recognition*, vol. 38, no. 9, 2005.

[23] M. Bugalho and A. L. Oliveira, "Inference of regular languages using state merging algorithms with search," *Pattern Recogn.*, vol. 38, no. 9, 2005.

[24] R. Parekh and V. Honavar, "Grammar inference, automata induction, and language acquisition," *Handbook of natural language processing*, pp. 727–764, 2000.

[25] J. Oncina and P. Garcia, "Inferring regular languages in polynomial update time," in *Pattern Recognition and Image Analysis*, ser. Series in Machine Perception and Artificial Intelligence, N. P. de la Blanca, A. Sanfeliu, and E. Vidal, Eds., vol. 1. World Scientific, Singapore, 1992, pp. 49–61.

[26] M. J. Kearns and U. V. Vazirani, *An Introduction to Computational Learning Theory*. Cambridge, MA, USA: MIT Press, 1994.

[27] M. Bugalho and A. L. Oliveira, "Inference of regular languages using state merging algorithms with search," *Pattern Recogn.*, vol. 38, no. 9, Sep. 2005.

[28] E. Gold, "System identification via state characterization," *Automatica*, vol. 8, no. 5, pp. 621 – 636, 1972. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0005109872900337

[29] W. Wieczorek, *Grammatical Inference: Algorithms, Routines and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2016.

[30] A. L. Oliveira and S. Edwards, "Limits of exact algorithms for inference of minimum size finite state machines," in *Proceedings of the 7th International Workshop on Algorithmic Learning Theory*, ser. ALT '96, 1996.

[31] A. L. Oliveira and J. a. P. M. Silva, "Efficient algorithms for the inference of minimum size DFAs," *Mach. Learn.*, vol. 44, no. 1-2, pp. 93–119, Jul. 2001.

[32] K. J. Lang, "Faster algorithms for finding minimal consistent DFAs," Tech. Rep., 1999.

[33] S. M. Lucas and T. J. Reynolds, "Learning DFA: evolution versus evidence driven state merging," in *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, vol. 1, Dec 2003, pp. 351–358.

[34] H. Juillé and J. B. Pollack, "A stochastic search approach to grammar induction," in *Grammatical Inference*, 1998.

[35] E. M. Gold, "Language identification in the limit," *Information and Control*, vol. 10, no. 5, pp. 447 – 474, 1967. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019995867911655

[36] R. E. Schapire, *The Design and Analysis of Efficient Learning Algorithms*. Cambridge, MA, USA: MIT Press, 1992.

[37] M. Isberner, F. Howar, and B. Steffen, "The TTT algorithm: A redundancy-free approach to active automata learning," in *Runtime Verification*, B. Bonakdarpour and S. A. Smolka, Eds. Springer International Publishing, 2014, pp. 307–322.

[38] B. Steffen, F. Howar, and M. Merten, "Introduction to active automata learning from a practical perspective," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2011, pp. 256–296.

[39] B. Jonsson, *Learning of Automata Models Extended with Data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 327–349.

[40] F. Aarts and F. Vaandrager, "Learning I/O automata," in *CONCUR 2010 - Concurrency Theory*, P. Gastin and F. Laroussinie, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 71–85.

[41] M. Isberner, F. Howar, and B. Steffen, "Learning register automata: from languages to program structures," *Machine Learning*, vol. 96, no. 1, pp. 65–98, Jul 2014.

[42] W. Smeenk, J. Moerman, F. Vaandrager, and D. N. Jansen, "Applying automata learning to embedded control software," in *Formal Methods and Software Engineering*. Springer International Publishing, 2015.

[43] S. Goldman and M. Kearns, "On the complexity of teaching," *J. Comput. Syst. Sci.*, vol. 50, 1995.

[44] J. L. Peterson, "Petri nets," *ACM Comput. Surv.*, vol. 9, no. 3, pp. 223–252, Sep. 1977. [Online]. Available: http://doi.acm.org/10.1145/356698.356702

[45] M. Yoeli, "The cascade decomposition of sequential machines," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 4, pp. 587–592, Dec 1961.

[46] R. Dijkman, J. Hofstetter, and J. Koehler, *Business Process Model and Notation*. Springer, 2011.

[47] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, Sept 2004.

[48] A. Weijters, W. M. van Der Aalst, and A. A. De Medeiros, "Process mining with the heuristics miner-algorithm," *Technische Universiteit Eindhoven, Tech. Rep. WP*, vol. 166, pp. 1–34, 2006.

[49] C. W. Günther and W. M. Van Der Aalst, "Fuzzy mining–adaptive process simplification based on multi-perspective metrics," in *International Conference on Business Process Management*. Springer, 2007, pp. 328–343.

[50] W. van der Aalst *et al.*, "Business process mining: An industrial application," *Information Systems*, vol. 32, no. 5, pp. 713–732, 2007.

[51] W. M. P. van der Aalst, "Business process management: A comprehensive survey," *ISRN Software Engineering*, vol. 2013, pp. 1–37, 2013.

[52] R. S. Mans, M. H. Schonenberg, M. Song, W. M. P. van der Aalst, and P. J. M. Bakker, *Application of Process Mining in Healthcare - A Case Study in a Dutch Hospital*, ser. Biomedical Engineering Systems and Technologies. Springer Nature, 2008, pp. 425–438.

[53] A. Partington, M. Wynn, S. Suriadi, C. Ouyang, and J. Karnon, "Process mining for clinical processes," *ACM Transactions on Management Information Systems*, vol. 5, no. 4, pp. 1–18, 2015.

[54] E. Rojas, J. Munoz-Gama, M. Sepúlveda, and D. Capurro, "Process mining in healthcare: A literature review," *Journal of Biomedical Informatics*, vol. 61, pp. 224–236, 2016.

[55] H. Yang, M. Park, M. Cho, M. Song, and S. Kim, "A system architecture for manufacturing process analysis based on big data and process mining techniques," in *2014 IEEE International Conference on Big Data (Big Data)*, 10 2014.

[56] P. Viale, C. Frydman, and J. Pinaton, "New methodology for modeling large scale manufacturing process: Using process mining methods and experts' knowledge," in *2011 9th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, 12 2011.

[57] B. N. Yahya, "The development of manufacturing process analysis: Lesson learned from process mining," *Jurnal Teknik Industri*, vol. 16, no. 2, 2014.

[58] R. Malik, M. Fabian, and K. Åkesson, "Modelling large-scale discrete-event systems using modules, aliases, and extended finite-state automata," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 7000 – 7005, 2011, 18th IFAC World Congress.

[59] R. E. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys (CSUR)*, vol. 24, no. 3, pp. 293–318, 1992.

Part II

Included Papers

Paper 1

Error Handling Within Highly Automated Automotive Industry: Current Practice and Research Needs.

Ashfaq Farooqui, Patrik Bergagård, Petter Falkman, and Martin Fabian.

2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), 2016, Berlin, Germany.

Comment: The paper has been reformatted for readability, but is otherwise unchanged.

Error Handling Within Highly Automated Automotive Industry: Current Practice and Research Needs.

Ashfaq Farooqui, Patrik Bergagård, Petter Falkman, and Martin Fabian.

Abstract

Fault tolerant systems, commonly found in literature, are implemented in various computer applications. Some of these methods have been studied and developed to aid manufacturing systems; however, they have rarely been integrated into the manufacturing process. Broadly, the problem seems to be integration of error handling procedures towards the end of physically building the manufacturing line, lack of a defined workflow, untested program logic and inadequately equipped personnel to name a few. To this end, a survey was conducted within the Swedish automotive industry to get an understanding of current error handling procedures and its shortcomings, and are presented here. Based on this data, and looking at the trends within the manufacturing industry, this paper also identifies research topics aimed towards defining methods to create next generation fault tolerant manufacturing systems.

1 Introduction

Complexity of automotive manufacturing industry is constantly increasing to keep up with advancement in technology, market trends, legislative requirements, and most of all high quality products. To this end, the systems developed are highly automated with minimal manual handling; they must not only take into account flexibility, efficiency, and development time, but also account for fault tolerant behavior.

Development of automated manufacturing systems is a demanding task. One of the main hurdles is the inability to validate and verify all requirements before physically commissioning the system; this means the physical system needs to continuously be upgraded when problems are encountered in the design. As a consequence, human operators – maintaining these systems – need to be highly skilled, but cannot be given sufficient training until the physical system is commissioned.

Advancement in technology today promises tools and methods that may help overcome these hurdles and will eventually lead to improved designs and more robust solutions. Formal methods and virtual commissioning are good examples of this, that while not previously adapted due to lack of usable tools, are gaining acceptance within the automotive industry. Using these tools will allow industries to first verify and virtually validate a design before physically commissioning the system. Additionally, it will allow operators to be trained within the virtual space; thereby increasing efficiency.

According to Loborg [1], a *fault* is what causes a difference between the specification of a system and that which is observed, also known as an *error*. A *failure* is said to occur when an *error* results in a loss of service. Fault tolerant systems – systems that can handle an error without affecting the service delivered – have been studied in academia within the area of computer science. A number of these methods have been developed for manufacturing systems: Loborg [1] provides a survey of these methods; a case study analysis of eight industries using fault tolerant techniques is presented by Vogel-Heuser et al. [2]; Gao [3, 4] gives an elaborate overview on detection and diagnosis within fault tolerant systems. There is however, a lack of defined workflow that will help design, verify, and validate a manufacturing system before physical commissioning to ensure fault tolerance. *Error handling* on the other hand is the course of action employed, after occurrence of an error, to mitigate its effects and avoid a failure scenario. The present paper will use these definitions for the three terms – *fault*, *error*, and *failure* – while looking at current trends in the industry and providing possible research topics.

1.1 Contribution

This paper is part of ongoing work aimed towards defining a process to build fault tolerant manufacturing systems. It presents an overview of current error handling procedures employed in the industry today and its shortcomings – based on a survey conducted within the body-in-white segment of the Swedish automotive industry. Based on the problems identified from the survey, this paper will introduce future research areas that aims at supporting operators during error scenarios both for virtual and physical systems. The main idea with such techniques is to define a framework and workflow that will incorporate error handling into the initial preparation phase of the manufacturing system. Also, the paper suggests an additional step to the already existing process to ensure safety of the manufacturing line.

1.2 Outline

The paper is divided as follows: Section 2 provides an overview of current industrial setup and processes for error handling. This is followed by a brief outline of the survey results in Section 3. Section 4 provides some insights into future direction of the manufacturing industry which influence the proposed framework. Finally, Section 5 suggests possible research topics aimed towards creating fault tolerant systems.

2 Background

Within the body-in-white segment, a manufacturing *line* generally consists of a number of manufacturing *stations* – each responsible for a specific task, such as spot welding, stud welding, gluing etc. These tasks are generally performed by resources including, a certain number of robots needed for the actual process; further assisted by conveyors or Automatic Guided vehicles (AGV) for material handling. Tools, tool changers, and other task specific actuators are also present. The complete station in a broad perspective normally consists of a number of sub-stations: manual or automated feeding sub-stations; process sub-station, where the actual process takes place; a checking sub-station to ensure quality; and an unloading sub-station that feeds the product into the next station.

A manufacturing station can broadly be divided into: *physical system* consisting of resources and the product parts; and *control system* that defines the behavior of the station. This control system consists of a number of operations that must be executed in a defined manner by the physical system. For the control system to effectively control the physical system it is important that their states are always synchronized, here the *state* refers to a set of variables that capture the properties of a system or subsystem in a unique way.

2.1 Error handling process

Inspired by [1], [5] divides the process of error handling into the following phases:

- *Detection*: Where the actual state of a system is monitored and compared with its specifications in order to determine any discrepancies during execution.
- *Diagnosis*: Once an error has been detected, using available information to determine the fault that caused the specific error.
- *Correction*: Here the fault which caused an error is corrected, either by replacing or fixing the faulty part, usually by intervention of an operator.

- *Restart*: In order to continue execution safely and efficiently after the correction phase, the control system and physical system are resynchronized i.e making state of the controller and physical machinery to correspond, resulting in process restart.

During the restart phase, the state of the physical system is changed to a legal one by the operator. The challenge is then, to modify the controller state to correspond to the new physical state [6] so as to allow safe and efficient restart.

Though there have been a plethora of approaches towards fault tolerant systems within academia, there is no clarity on how many have been tested and implemented within the industry. The reason being lack of tools, processes and more often than not a disconnect between industry and academia.

3 Survey summary

A survey was conducted involving Volvo Cars, Volvo Group Trucks, Scania, National Electric Vehicle Sweden, and GKN Aerospace. The main intention was to identify and understand common errors faced in these industries, how they are dealt with, and measures taken to ensure fault tolerance.

3.1 Error scenarios

A set of commonly occurring errors that were identified are discussed below:

- Tool breakage is generally one of the most common fault that result in an error leading to loss of service; in extreme cases it might result in scrapping of the part. Detection and diagnosis in this case is rather quick, correction is fault dependent in most scenarios. The restart phase is handled in various ways depending on the task performed by the station. In some cases where cycle time is a few seconds (0-5s), re-running the complete cycle has no effect; in other cases when the process takes more than few minutes (15-45 min) re-running the complete process is redundant and might induce lags in the manufacturing line.

- Missing parts or buffer shortage are rarely an issue in the surveyed manufacturing stations as they are manually fed. In any case, if there is a missing part, production is paused and will be resumed when there is availability of the said part. Though there is no need for error handling procedure, it often results in a delay for the manufacturing line. Such delays tend to cause unsynchronized behavior in the subsequent stations possibly resulting in an unintended sequence of operations.

- Resource malfunction, i.e when any one of the resources in the station breaks down; this can result in a complete halt of the station or a temporary break. If the resource is replaceable it is replaced and production continues. On the other hand, if the resource cannot be replaced, for example a robot, then production is delayed until it has been repaired. The restart procedure in this case remains similar to that of tool breakage; the specifics are left to the operators discretion.

- Software bugs are not so uncommon in a manufacturing station and take up considerable amount of time to diagnose and implement permanent solutions. Since changes need to be made directly to the system, each bug fix may, as a side effect, inject new bugs. Apart from this, detection and diagnosis of software bugs requires highly trained and experienced personnel.

- Power outage is not a very common problem in industry today, there are backup systems to keep the production going. However, there are instances of power outage, e.g due to lightning. The problem lies not in the loss of power but rather its effect, as the *manufacturing stations* in the *line* could be unsynchronized with its respective control system; this can complicate the recovery process. In re-starting the line, there lies a major risk of damage to the various stations or a product part. The operator is then responsible to ensure the different systems are in the correct and safe state in-order to restart. This process is cumbersome, manually exhaustive and results in loss of production time.

- Preemptive emergency stops, as the name suggests, halts the manufacturing station in order to prevent the occurrence of an error. This is usually done on recognition of a fault by the operator as a safety measure. The effects are similar to that of power outage, and the operator is faced with the same challenges to ensure the safety of the complete line. Apart from this, there is no data from the manufacturing line to help further diagnose and study the fault.

- Unintended sequence of operations was not reported as a commonly occurring problem in the survey, mainly because most manufacturing stations have a fixed sequence of operations predefined in the control system. Hence, the manufacturing station can perform a single process. By allowing the control system to dynamically create the sequence of operations the station will then allow processing a range of product parts. In the future, with highly automated systems in which dynamic sequences are the norm, the control system must be verified to avoid occurrence of unintended sequences.

All the above errors implicitly require the operators to be highly skilled to handle any error scenario. This further opens up possibilities for human error due to miscommunication, lack of training and documentation, or misjudgment of the situation.

3.2 Measures to avoid error handing scenarios

Given the process of handling errors, having skilled operators is key to high productivity and low downtime. Apart from the training every operator is provided, line builders provide instruction manual or remote/on-site assistance to help support operators. In cases where there are multiple operators working within a single station, an internal operator manual and a logbook are maintained to support knowledge transfer and allow for better judgment.

During the preparation phase, before physical commissioning of the manufacturing station, simulations are run to make sure no geometric reachability issues arise. Apart from that, no other simulations are run to verify any other aspects of the system. Physical subsystems are generally validated along with function blocks at the line-builders site; the next stage of testing, today, is only when the station is setup. After which the logic and station are iteratively modified and validated. In this procedure, error scenarios are considered at a late stage of commissioning, generally after the physical station is commissioned. This leaves very little room for modifications and to incorporate error handling.

4 Future trends within manufacturing

Proposing effective ideas as solutions to the scenarios discussed in Section 3 must also take into account technological heading of the industry. This section introduces research questions, pursued by both academia and industry, that either require additional consideration to make them fault tolerant or will provide a framework to help realize the solutions.

Industry 4.0 [7], also called as the fourth industrial revolution, can be seen as a collection of various technologies – Internet of Everything(IoE), Cyber-physical Systems (CPS), and smart factories – to create the next generation of industrial systems [8]. Enabled by interaction between products, machines, and people, future industrial systems will be able to make smart decisions. From the design principles of Industry 4.0 provided by Hermann et al [8], distributed modular systems are key to build these next generation factories. Various projects directed towards the future of industrial systems have been initiated in many countries; *Factories of the Future* [9] is one such long term project running within the EU.

Automated robot systems are finding space within ship construction [10] and within aircraft construction [11]; the unique feature here is that the product stays in one place, while the robots move around depending on task and requirement to perform their respective operations. These types of systems increase complexity to keep the physical and control system synchronized, hence a robust control strategy and decision making algorithms are needed. Furthermore, verification of restart methods for multi-robot distributed systems needs to be well defined.

Apart from distributed systems, flexible or reconfigurable system design is another key factor in the Industry 4.0 vision. These systems will allow for dynamically changing configuration based on product requirement. One such project is *Factory in a day* [12] where a new manufacturing line can be setup in a short time. Or, it can be used to temporarily extend an existing manufacturing line to cater to market needs. While the project is aimed at Small and Medium scale Enterprises (SMEs), it has created interest within a larger community. In the light of fault tolerance, the reconfigured systems must be compatible with the existing error handling work-flows, and must also be verified for fault tolerance before changing the physical system.

Manufacturing lines produce large amounts of raw low-level data during each operation cycle. This data is then refined to provide meaningful information regarding the manufacturing line, which can further facilitate real-time decision making. To this end, Theorin et al. [13] provide a *Line Information System Architecture (LISA)* – an event-based service-oriented architecture which is both flexible and scalable. Manufacturing lines capable of utilizing this can provide much needed help in building future fault tolerant systems and improved error handling procedures.

Integrated Virtual Preparation and Commissioning (IVPC), introduced in [14], provides a framework to iteratively design and develop a manufacturing system within a virtual environment. In this method, the control system is implemented employing formal tools and validated, both against visual inspection and computations by formal methods, using a virtual model before actual commissioning of the station. Apart from providing an agile process to construct the control system, such a framework also allows for hardware-in-the-loop testing and virtual training for operator personnel.

5 Research needs

In Section 3 various error scenarios that effect the manufacturing station were presented. After an error-causing fault has been corrected, the physical and control system are unsynchronized; synchronizing the two is part of the restart phase referred to as resynchronization. Unlike Loborg [6] who suggests changing the internal controller state to correspond to that of the physical system, Bergagård et

al. [5] suggest that the control system is changed to a state from where it is correct to restart the system, and that the state of the physical system is changed accordingly. This method has been validated in a windscreen mounting station [15] with positive results. This solution holds for error scenarios like tool breakage, machine malfunction etc, but does not address safety of the complete manufacturing line on restart after a power outage or an emergency stop. Hence, an additional phase after the restart phase is suggested for further study, henceforth called *assurance phase*.

The *assurance phase* is made possible by using raw low-level data collected during the operation cycle, similar to the LISA project discussed earlier in Section 4. Using the current and last known sensor values to determine the state of the system and comparing this with the intended state, safety measures can be computed. Then, the control system will guide an operator to re-start the manufacturing line safely.

Another issue touched upon in Section 3 is the use of a logbook by operator personnel. In many cases this book is not well updated or could be misinterpreted leading to misjudgment and unwanted decisions. Maintaining logs of raw sensor data, a *digital logbook* in this case can come of use, instead of using manually maintained logbooks. The functionality of such a *digital logbook* can be extended to: help diagnose errors; facilitate training of operator personnel using the virtual environment to recreate various scenarios; and use in restart situations for safety assurance as already discussed.

To be able to achieve both, *assurance phase* and *digital logbook*, functionalities a common format for the logged data needs to be defined. Algorithms to efficiently, process, analyze, visualize, and store such large amounts of data need to be created. Additionally, the defined format must support interoperability between the physical system and its virtual counterpart.

Another area of study within error handling is distributed systems. With distributed systems decision making is decentralized, for example in ship and aircraft construction robots discussed in Section 4. As there is no centralized controller keeping track of individual subsystems, there is a need for formal tools and processes that will enable modeling and verification of such decentralized systems, specifically to ensure fault tolerance. Once a technique to model and verify decentralized systems is in place, the actual restart of such manufacturing system is of interest. A study of well defined workflow and efficient algorithms that can guide the operator to safely restart one or more resources will provide much needed foundation.

6 Conclusion

In conclusion, a survey conducted within Swedish automotive industry exhibited the need for a framework and work-flow to handle the following problems:

- Restart after power outage and emergency stops.
- Software bugs after commissioning.
- Training and support to personnel.
- Knowledge transfer and maintenance.

Based on future trends within the industry, this paper presented a need for further study on:

- An additional *assurance phase* after resynchronization.
- Use of logged data– *digital logbook* – from the manufacturing line to perform *assurance*; maintain knowledge; and, train personnel using the virtual world.
- Data format and algorithms to realize an *assurance phase* and a *digital logbook* leading to a fault tolerant system.
- Modeling and verification of restart within distributed and decentralized systems using formal methods.
- Validating restart of one or more distributed and decentralized system resource during active process.

7 Bibliography

[1] P. Loborg, "Error recovery in automation an overview," in *AAAI-94 Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems, Stanford, Ca, USA*, 1994.

[2] B. Vogel-Heuser, S. Rösch, J. Fischer, T. Simon, S. Ulewicz, and J. Folmer, "Fault handling in PLC-based industry 4.0 automated production systems as a basis for restart and self-configuration and its evaluation," *Journal of Software Engineering and Applications*, vol. 9, no. 1, p. 1, 2016.

[3] Z. Gao, S. X. Ding, and C. Cecati, "A survey of fault diagnosis and fault-tolerant techniques;part ii: Fault diagnosis with knowledge-based and hybrid/active approaches," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3768–3774, June 2015.

[4] Z. Gao, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques ;part i: Fault diagnosis with model-based and signal-based approaches," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3757–3767, June 2015.

[5] P. Bergagård and M. Fabian, "Calculating restart states for systems modeled by operations using supervisory control theory," *Machines*, vol. 1, no. 3, pp. 116–141, 2013.

[6] P. Loborg and A. Törne, "Manufacturing control system principles supporting error recovery," in *Proceedings of the AAAI Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems, Palo Alto, CA, USA*, vol. 2123, 1994.

[7] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, 2014. [Online]. Available: http://dx.doi.org/10.1007/s12599-014-0334-4

[8] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan 2016, pp. 3928–3937.

[9] "Factories of the future." [Online]. Available: http://www.effra.eu/

[10] "Carlos." [Online]. Available: www.carlosproject.eu

[11] "Cablebot." [Online]. Available: www.cablebot.eu

[12] "Factory-in-a-day." [Online]. Available: www.factory-in-a-day.eu/

[13] A. Theorin, K. Bengtsson, J. Provost, M. Lieder, C. Johnsson, T. Lundholm, and B. Lennartson, "An event-driven manufacturing information system architecture," in *IFAC/IEEE Symposium on Information Control Problems in Manufacturing, INCOM*, 2015, pp. 547–554.

[14] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Integrated virtual preparation and commissioning: supporting formal methods during automation systems development," 2016.

[15] P. Bergagård, P. Falkman, and M. Fabian, "Modeling and automatic calculation of restart states for an industrial windscreen mounting station," *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 1030–1036, 2015.

Paper 2

From Factory Floor to Process Models: A Data Gathering Approach to Generate, Transform, and Visualize Manufacturing Processes.

Ashfaq Farooqui, Kristofer Bengtsson, Petter Falkman, and Martin Fabian.

Submitted for possible journal publication. 2018

Comment: The paper has been reformatted for readability, but is otherwise unchanged.

From Factory Floor to Process Models: A Data Gathering Approach to Generate, Transform, and Visualize Manufacturing Processes.

Ashfaq Farooqui, Kristofer Bengtsson, Petter Falkman, and Martin Fabian.

Abstract

Obtaining data from automotive manufacturing stations will help operators and engineers better understand the stations as thereby improving its performance. The data needs to be extracted from tow sources, the PLC's and the robots. In practice, methods to extract usable data from robots are rather scarce. In this work, we provide an approach to capture data from robots, which can be applied to both legacy and current state-of-the-art manufacturing systems. The described approach is developed in Sequence Planner – a tool for modeling and analyzing production systems – and is currently implemented at an automotive company as a pilot project to visualize and examine the ongoing process. By exploiting the robot code structure, robot actions are converted to event streams that are transformed into an abstraction called operations. We then demonstrate the applicability of the resulting data, abstracted as operations, by visualizing the ongoing process in real-time as Gantt charts, that support operators performing maintenance. And, the data is also analyzed off-line using process mining techniques to create a general model that describes the underlying behaviour of the manufacturing station. Such models are used to then derive insights about the relationship between different operations, and also between resources.

1 Introduction

The complexity of the automotive manufacturing constantly increases to keep up with advancements in technology, market trends, legislative requirements, and most of all high quality products. While new techniques and processes are being researched under the umbrella of Industry 4.0 [1], it is not easy to implement these techniques on existing systems. The increase in complexity can be due to increased interaction between different sub-systems, added functionality to handle variants by the station, addition of logic to ensure safety, and optimization

techniques employed to improve factors such as process time, energy, quality, etc.

Additionally, once a manufacturing line is commissioned it is expected to last for several years to even out investment and start generating returns. More often than not, the obstacle of higher costs hinders the adoption of newer technologies. Hence, the gap between existing technologies on the factory floor and available technologies rarely diminishes. The requirement, to keep existing and newer systems compatible, adds to the complexity of manufacturing systems.

An overall system view that helps operators and engineers better understand the system and its dependencies, is rare. This makes it more difficult to debug problems and improve performance in a reliable and verifiable manner.

Model-based techniques, that offer design, validation, verification, and testing, are being actively adopted within the manufacturing industry [2, 3] to formally ensure correctness of complex systems. The last few years have seen a drastic advancement in model-based algorithms within, both, the more theoretical formal methods and application oriented computer science domains, that are practical to use on regular manufacturing systems. These techniques are usually coupled with virtual technologies such as simulations [4] and virtual reality [5] in the early phase of manufacturing or the virtual commissioning phase [6], thereby leading to shorter physical commissioning effort. However, creating formal models is a challenging task that requires skill, in-depth knowledge of the system, and creativity. Incorrect or incomplete models are misleading and can unnecessarily complicate the process. A possible way to ameliorate this issue would be to create and adjust models based on the data obtained from the factory floor.

One major challenge within the automotive manufacturing industry has been to, conveniently, access and extract useful data from the factory floor. Access to this data might provide a key step towards automatically building models. A possible way to extract this data would be using commonly used protocols such as OPC-UA [7] or Snap 7 [8]. However, the data collected using these protocols, provides access to the PLC and not to any robot data. And there lacks an efficient way to extract data from robots operating in the station. One major reason for this is the restrictions to access data imposed by robot vendors.

Today, based on predefined requirements, robots are programmed to send data to the PLC. This data usually includes alarms, warnings, and measurements made during execution. The PLC further aggregates the data and saves it in a central database. Data stored in the database is then used off-line at distinct time instances to calculate cycle times. To enable this, each robot requires additional code, in the robot itself and in the PLC, that has to be manually added and maintained. Therefore, it is currently difficult to track real-time operation of robots. This tracking, usually in the form of visualizations, help operators and engineers

tune and improve performance of the system, and also enable data-driven approaches for preemptive maintenance.

To this end, a method to capture robot data from existing and new manufacturing systems will prove beneficial.

1.1 Contribution

Error handling within the Swedish automotive industry was investigated in [9]. One major outcome of that study was the need to have data and tools on the factory floor to be able to: visualize the ongoing process; visualize the current state of the system, which in case of errors, will help with restarting the system; replay recorded data in a virtual model in order to aid training of operators at the station. In this work, we address the identified need and create tools and processes that will help mitigate problems discussed in [9].

The contribution of this paper can be classified into two parts. The first part deals with generation of data from the robot station. In the second part we demonstrate the applicability of the collected data by automatically creating models and visualizations.

The first contribution of this paper results in toolset that provides access to system level data so as to enable operators, engineers, and upper level management to make technical data driven decisions. In order to do this, this paper provides a data processing pipeline that starts with a generic and non-intrusive approach to capture low level data from an existing robot station, followed by, transforming this data into a higher abstraction as *operations*.

The second contribution deals with the use of this data for model generation. Here, the captured data is used for online visualization of robot operations. In addition, the data collected is also used off-line to understand the behaviour by creating and visualizing models on a relevant abstraction level that illustrate the overall execution, achieved using process mining tool ProM [10].

This article will provide an insight into the tool Sequence Planner [11] used to create the framework that captures and processes the data from a demo station at an automotive company. However, for practical details regarding application setup and configuration interested readers are referred to [12].

1.2 Outline

The article is structured as follows: Section 2 highlights and motivates the general architecture that has been developed to gather and process data in a distributed and non-intrusive manner. A more detailed explanation of the software and data transformations along with real-time visualizations of the gathered data is presented in Section 3. Section 4 describes how the collected data is used

off-line to understand the behaviour of the manufacturing station by creating and visualizing models that illustrate the overall execution. Finally, Section 5 concludes by shortly mentioning the future steps resulting from this work.

2 Software Architecture

Within the manufacturing community different production paradigms exist, like Reconfigurable, Flexible, Adaptive, Multi-agent, Holonic, manufacturing systems, to name a few [13]. At the core of these systems lies the idea that each subsystem performs one specific function. Similarly, in the computer science community the last few years have seen the development of Service Oriented Architectures (SOA), where each service performs one specific task. This parallelism in the two communities allows for easy integration of ideas between them.

Theorin et. al. [14] provide a broader discussion on various architectures, and introduce the Line Information System Architecture (LISA) that uses an Event-driven Service Architecture (EDA). Ideas from LISA are used to build a streaming data pipeline, which is basically a series of steps that transform real-time data between different formats and abstraction levels. In Figure 1, an overview of a pipeline is shown including the event stream generation, transformation and higher level services.

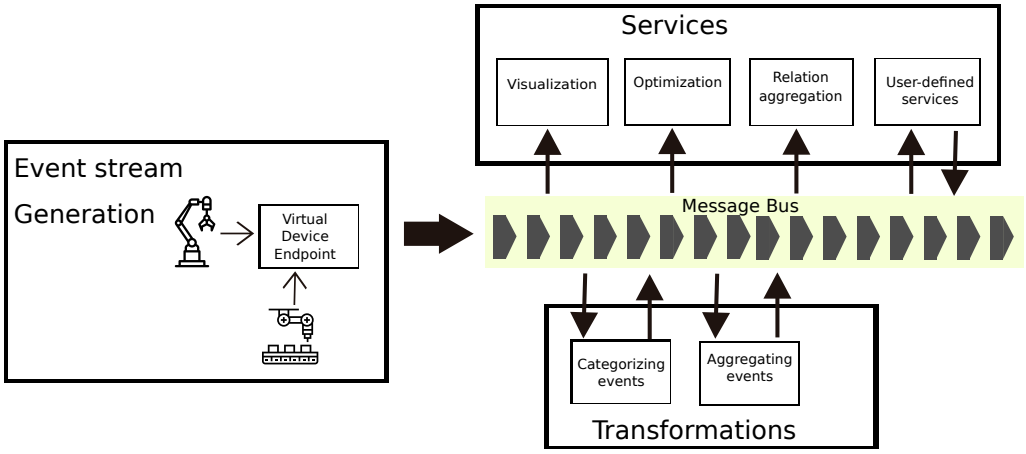


Figure 1: Overview of a general computational pipeline, along with their sub-functions.

2.1 Pipeline components

A data pipeline broadly consists of two types of components, namely a *message bus*, responsible for communication and *endpoints*, data processing modules that

process messages, connected to the message bus. Figure 1 shows how data is passed through a pipeline in the order to obtain desired results.

1. *Virtual Device (VD) Endpoints*: VD endpoints provide an entry point to generate event streams from the physical hardware such as robots, PLCs, scanners etc., onto the message bus. Implementing communication protocols within each low level systems in order to communicate with the message bus is a hard task, and not always feasible. Instead, the VD is a wrapper that provides a message-based interface; it thereby simplifies the architecture and allows plug-and-play design for hardware components, providing seamless integration for new devices.

   The output from a VD endpoint is a stream of simple, low level messages that can be interpreted and used by all other endpoints.

2. *Transformation Endpoints*: Low level systems communicate with simple messages which are sent by the VD onto the bus. Transformation endpoints convert such low level data, and generally data on a low abstraction level, into higher abstractions, thereby making the data more usable for other endpoints in the pipeline. There are three main types of transformations, `fill` and `map` both of which add additional information to the event streams, and `fold` which produces new events based on aggregating a number of previously received events. The `fill` transformation appends additional information to the events, while the `map` transformation uses current state information while appending to an event.

3. *Service Endpoints*: Service Endpoints provide a service – one specific function – with the incoming data as input and may or may not send processed data on the bus. That is to say, services consume data from one or more transformations, and then use this data to compute a result. Since services are based on user needs, the pipeline allows integration of new services without hampering the existing process. Examples of services may include, aggregation services, prediction services, and visualization services.

2.2 Message bus

The message bus forms the communication layer allowing interaction between all endpoints. There exist, in literature and in practice, a number of possible configurations to create a message bus. While configuring a bus, the main objective is to aim for low complexity and high scalability. Ideally, it must be possible to change or upgrade the message bus without major changes to the code.

Services, in the pipeline, are triggered to execute when a message arrives for it. To make this possible, messages on the bus are structured into *topics*, and each

service subscribes to one or more topics. When a message arrives on a specific topic subscribed by a service, that particular service is executed once.

To keep the implementation simple, the Apache ActiveMQ [15] message bus was chosen in the use case presented in the next section. The bus was tested with a total of about 20 interacting resources and several services running online; no issues related to performance were noticed. However, if higher throughput and a distributed nature is required, Apache Kafka [16] or any other compatible bus can be used with minimal changes to the existing system.

3 Robot event pipeline

The approach presented in this paper has been implemented at a body-in-white production station at an automotive company. The station performs spot welding using four robots on a variety of different car models produced by the company. Once the car body enters the station and is in position, the robots choose the appropriate predefined program depending on the car model, and then moving in the workspace between pre-programmed spatial positions where they perform operations such as welding. During these movements they might need to wait for each other when entering common shared zones. Apart from this, tip dressing operations – the process of cleaning the welding tip – are also performed regularly by the robots to maintain weld quality.

ABB robots are used in this station which are programmed using a high-level programming language – *RAPID* – developed by ABB for their industrial robots. This language supports *modular programming*, that is, these programs are divided into, smaller, self-contained, modules that contain a set of instructions corresponding to some physical division in the system. Each module is further divided into *routines* which correspond to a unit task performed by the robot. Each line in the program corresponds to an action by the robot, which is performed sequentially. That is, there are entry and exit points defined for the robot program. The line being executed is referred to by the *program pointer*. Apart from robot movements, the program also interfaces with the *signals* which provide input/output functionality. This code structure will prove beneficial when transforming the generated data into operations as explained later in this section.

The implementation is done using Sequence Planner (SP) [11], a tool developed at Chalmers University for modeling and analyzing automation systems. SP is developed as a micro-service architecture, where live data can be connected to distributed event pipelines, using a Virtual Device (VD) endpoint, and transformed into usable information for visualization as well as various algorithms. In this use case, an ABB endpoint, transformation pipelines, visualization, and data analysis was implemented in SP.

3.1 ABB endpoint

To feed live data from an ABB robot into SP, a ABB VD was developed to interface with ABB robots; which is achieved using the Robot SDK [17] provided by the robot manufacturer. This SDK provides a mechanism to subscribe to both the program pointer and the Input/Output signals present in the robot controller.

The VD endpoint uses the SDK to receive notifications on the event of a change in the program pointer or the signals; which is followed by sending a message – after appending a header – on the message bus with the appropriate contents. An example robot message for a *program pointer position*, transmitted when the program pointer advances in an execution step, is shown in Listing 2.1.

The message consists of a program pointer position, an address, and a header. The *programPointerPosition* block contains information regarding the position of the program pointer. It contains the names of the `module` and `routine` currently executed by the robot. Additional information pointing to the exact line number is available under `range`. The value of `time` contained here is the timestamp when this event was created. The `address` block contains the path, internal to the robot, that generated the event.

The header block (the bottom three lines) contains information that helps identify the source of each message. It consists of a `robotId` that identifies the robot responsible for generating the event; a timestamp for when the endpoint received the pointer change information is sent as `time`; and a `workCellId`, a unique number to represent the manufacturing station where the message was generated.

Listing 2.1: ProgramPointerPosition message sent when the pointer position advances one step

```
{
 "programPointerPosition": {
   "position": {
     "module": "LD930R8119",
     "routine": "D931SchDefault",
     "range": {
       "begin": {
         "column": 5,
         "row": 250
       },
       "end": {
         "column": 28,
         "row": 250
       }
     }
   },
   "time": "2017-01-12T17:03:54.942+01:00",
   "task": "T_ROB1"
 },
 "address": {
   "kind": "programPointer",
   "path": [
     "T_ROB1"
   ],
```

```
    "domain": "rapid"
  },
  "robotId": "r8255",
  "time": "2017-01-12T17:03:54.942+01:00",
  "workCellId": "1741010"
}
```

Instead of a programPointerPosition message the robot can generate a *newSignalState* event when an IO value changes, as shown in Listing 2.2. In this case, the signal message, similar to a program pointer position message, consists of a newSignalState and the header. The newSignalState block has the value of the signal, a flag defining if the signal is simulated and the quality. The `address` block, similar to the address block in programPointerPosition, provides the source of the signal. The header appended is the same as explained in Listing 2.1

Listing 2.2: An example newSignalState message sent on an update in signal value

```
{
  "newSignalState": {
    "value": 0.0,
    "simulated": false,
    "quality": {
      "value__": 1
    }
  },
  "address": {
    "signal": "O_Homepos",
    "domain": "io"
  },
  "robotId": "r8119",
  "time": "2017-01-13T12:34:32.904+01:00",
  "workCellId": "1741010"
}
```

3.2 Transformation endpoints

As mentioned earlier, transformation endpoints transform data from one abstraction to another. The incoming raw data from a robot as seen in Listing 2.1 needs to be refined into something more understandable. One sufficiently good abstraction to use is that of an *operation* [18]. The robot program structure, defined previously, is suitable to easily transform the raw data into operations. We create two types of operations from the raw data named *routines* and *wait*. Routines in the robot program represent the tasks performed by the robot, these constitute "move from a to b", "grip", "tip-dress" or "weld", and hence they are abstracted as operations, a concept easily grasped by humans. Wait operations are relate to the time when the robot is waiting to get access to a shared resource, which is accomplished using the keyword "WaitSignal" in the robot programs. Using this information, the following subsections provide a step by step approach to

transforming the raw data into operations.

Naming and Categorizing events

The first step in the transformation is to be able to differentiate between the incoming events. This is done by naming the incoming events according to the routine they were generated from. Furthermore, an event due to a wait must be differentiated from other tasks. For every robot message that arrives, text corresponding to the range specified by the program pointer is extracted from the robot program; this text is the instruction the robot currently performs. If the extracted instruction reads “WaitSignal” then the event is categorized as a “wait” event, else it belongs to a “routine”. A new message is then sent out appending the original message with tags “instruction”, containing the instruction extracted from the program, and “isWaiting” with a value true if the event is categorized as a “wait” else a value false. The resulting message from this step is seen in Listing 2.3

Listing 2.3: A parsed message with additional “instruction” and “isWaiting” keys


```
{
  "programPointerPosition": {
    "position": {
      "module": "LD930R8119",
      "routine": "D931SchDefault",
      "range": {
        "begin": {
          "column": 5,
          "row": 250
        },
        "end": {
          "column": 28,
          "row": 250
        }
      }
    },
    "time": "2017-01-12T17:03:54.942+01:00",
    "task": "T_ROB1"
  },
  "address": {
    "kind": "programPointer",
    "path": [
      "T_ROB1"
    ],
    "domain": "rapid"
  },
  "instruction": "WaitSignal ReleaseStation",
  "isWaiting": "True",
  "robotId": "r8255",
  "time": "2017-01-12T17:03:54.942+01:00",
  "workCellId": "1741010"
}
```


Listing 2.4: Three different operation events


```
{
  "activityId" : "80b1a21e-4535-4797-ad65-d0ec47e2fc99",
```

```
  "isStart" : true,
  "name" : "WaitSignal ReleaseStation2;",
  "robotId" : "r8255",
  "time" : "2017-01-13T12:37:01.907+01:00",
  "type" : "wait",
  "workCellId" : "1741010"
 },
 {
  "activityId" : "80b1a21e-4535-4797-ad65-d0ec47e2fc99",
  "isStart" : false,
  "name" : "WaitSignal ReleaseStation2;",
  "robotId" : "r8255",
  "time" : "2017-01-13T12:37:02.311+01:00",
  "type" : "wait",
  "workCellId" : "1741010"
 },
 {
  "activityId" : "048266f1-a071-4f1e-b77e-0e5de57a8c23",
  "isStart" : true,
  "name" : "B940ToPutFixt071_3",
  "robotId" : "r8255",
  "time" : "2017-01-13T12:36:34.951+01:00",
  "type" : "routines",
  "workCellId" : "1741010"
 }
```


Aggregating events to operations

The named and categorized messages need to be further processed. The next step in the pipeline is to aggregate the different messages into operations. This is done by an endpoint that listens to named and categorized events generated in the way described in the section 3.2. The endpoint also keeps track of all available resources and the routines being currently executed by each of them. The output from this endpoint generates *operation events* – i.e. the start and stop events for operations. An operation event has a name, a timestamp, a resource where it is executed, and a flag that defines if this is a start or stop event. Listing 2.4 shows three different operation events, two events for a wait operation and a start event for a routine, running on two different robots. The start and stop operation events are also shown with their respective timestamps. Furthermore, operation events can be aggregated to view a complete operation as seen in listing 2.5. However, there is an advantage to preserve operation events instead of merging them into operations, since start-stop events can be processed to understand underlying relations between operations. Hence, both these types of messages are available on the bus. As shall be seen further on in the paper, operation events are used during real-time processing of data, mainly for visualization, and aggregated operation events are used for off-line analysis.

Listing 2.5: An aggregated operation


```
{
  "activityId" : "80b1a21e-4535-4797-ad65-d0ec47e2fc99",
  "name" : "WaitSignal ReleaseStation2;",
```

```
  "robotId" : "r8255",
  "startTime" : "2017-01-13T12:37:01.907+01:00",
  "stopTime" : "2017-01-13T12:37:02.311+01:00",
  "type" : "wait",
  "workCellId" : "1741010"
}
```

Identifying cycles

At this point we have access to a list of sequential operations following the pattern shown in Listing 2.5. This list contains all operations that were executed over a given period of time. These operations mimic the execution in the physical station which is cyclic. A set of operations together form a cycle. Automatic identification of cycles in real-time is a challenge.

In this particular example station a cycle ends when all four robots are at the home position. Hence, we monitor all robots for a signalState message containing the O_homepos variable, seen in Listing 2.2, to establish when all resources are at home. An additional check, to avoid noise, is done to see if all robots are executing an operation called `main` at this time. Therefore, when all for robots execute operation `main` and the signalState message identifies them to be in their home position, a new cycle is logged. However, in a more complicated system, when robots are independent in their tasks, the described criterion may not hold; thereby making identification of cycles a challenging task. Section 4.2 briefly describes an off-line cycle detection method that can be applied to a more complicated station.

3.3 Services

Having real-time data from factory floors abstracted into operations, various algorithms can be run in the form of services, such as visualizations, optimization, and prediction to understand and improve the station. Two services are highlighted below, they help operators maintaining the station by visualizing the ongoing operation.

Real-time resource based operation visualization

Information such as execution time for each cycle, execution time for each operation, total waiting time, operations where robots wait for each other are, among others important in order to maintain and develop the station. An overview of ongoing operations can be visualized using real-time Gantt charts. Operators and engineers responsible for maintaining and developing the station use these charts to keep track of ongoing processes at the station. Figure 2 shows a snapshot of a real-time Gantt chart of ongoing resource operations, appended with additional

information such as execution time for each operation. In the figure, each robot is identified by its name and has two rows corresponding to it. The first row traces "routines" while the second row traces "wait" operations. Similarly, past cycles can be visualized in the similar way as seen in Figure 3. Which shows a historical cycle annotated with additional information such as cycle duration, start and stop time, and a unique name identifying the cycle.

Figure 2: A real-time Gantt view of operations running in the station.

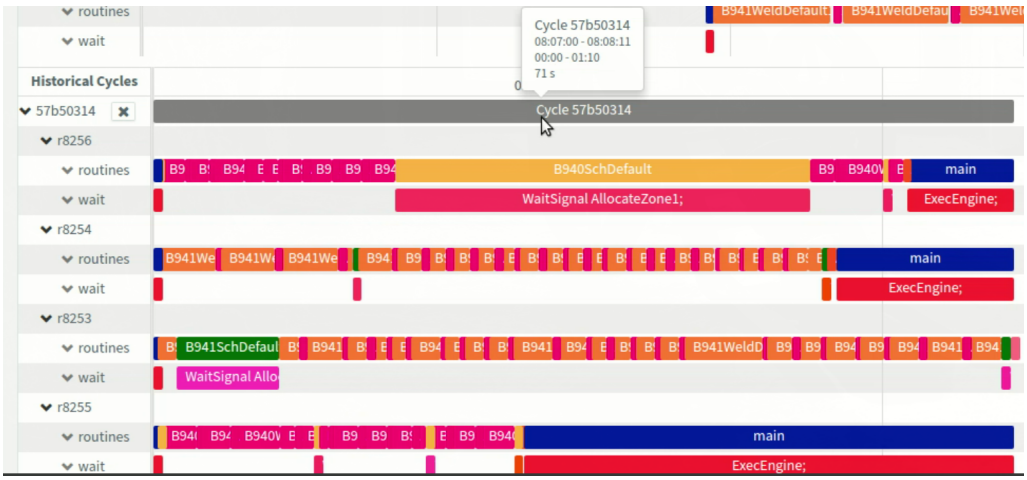


Figure 3: A historical Gantt view of the operation executed in the station.

Real-time operation-centered visualization

Knowledge relating to the relations between operations serve as important input during analysis and optimization of the system. Relations between various operations are interesting for several reasons. One of them being that, certain combinations might not be desired during execution. Instead of using a resource-centered view discussed in section 3.3, one can visualize operations and their relations in a Gantt view as seen in Figure 4.

Take as an example robot `r8255` highlighted within a box in the same figure. We see that `r8255_ArcSoftServoTrap`, which is the operation which

relates to the activation of the weld gun, runs after operations, `r8255_D910WeldDefault2` and `r8255_D910WeldDefault3`. Also, operations `r8255_D910WeldDefault2`, `r8255_D910WeldDefault3` and `r8255_D911WeldDefault1` run in a sequence, this relation can be deduced by a quick visual inspection of the Gantt chart. Though visual inspections, such as these, are possible, they are highly time-consuming and often inaccurate. As the number of interacting resources increases, so does the complexity in finding relations. Automated services using computational methods, are helpful in performing the same analysis, but with a higher degree of accuracy and is discussed in the next section.

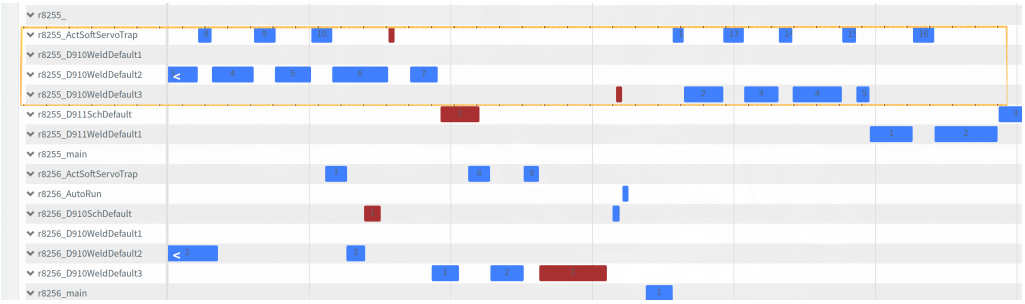


Figure 4: Real-time view of ongoing operations and the number of times they have run.

4 Towards creating models

The data generated and the messaging structure can be further used in various automated services. One such example is prediction services. These services use past and present data to make certain predictions regarding the factory floor. Predicting potential errors and failures can help prepare operators early on for what is coming and thereby mitigating downtime. For example, predicting the replacement of the welding tip by analyzing tip-dress operations that are responsible for cleaning the welding tip as explained by [12]. Similarly, access to several cycles of historic data opens up possibilities to predict a variety of KPIs such as cycle times, delays, maintenance requirement etc, that help in evaluation and maintenance of the station.

Another area of use relates to optimization services which can also benefit from the aggregated data. Optimization services use known models to find optimal sequences based on a given set of constraints. Dahl et. al. [11] present a method to optimize the sequence of operations with the aim of reducing cycle time. On the other hand, [19] and [20] focus on optimizing energy use by

finding optimal operation sequences, such that the overall cycle time is constant while specific operations run at lower speeds, thereby saving energy. These two methods can be combined and used for online optimization of sequences to find an optimal sequence that respects a given specification on time and energy. As these methods rely on the idea of operations, they can easily be integrated in the defined architecture to use the messaging structure.

Both, prediction services and optimization services, require a model defining the behaviour of the system to operate upon. These models could be created by operators with sufficient knowledge of the system. However, this requires an enormous amount of effort. First to create and then to maintain these models as the system evolves. It would be of interest if one could create a model of the manufacturing process automatically using the available data that would capture the underlying behaviour.

In the following section, focus will be on analyzing the data in an off-line manner in order to understand the manufacturing station by creating a model of it. Using the generated and captured data, as explained in the previous sections, it is possible to analyze and understand the behavior of the system. The aim of this section is to illustrate the potential uses of the data using specific tools and algorithms currently available. To this end, process mining has been used for automated analysis.

4.1 Process Mining

Process mining is a field of study that deals with process discovery from logged data [21]. In addition to that, process mining also includes conformance checking and enhancement. By looking at sufficiently large logs – spanning over several cycles – of labeled data, it is possible to discovery an accurate representation of the process. The output can be in the form of a Petri net, a transition graph, or a Business Process Model and Notation (BPMN). Process mining has shown significant benefit in understanding underlying task flows, bottlenecks, resource utilization and many other factors within large corporations [22, 23], and also proved beneficial in healthcare [24, 25, 26] to learn and improve the underlying process.

Within the manufacturing domain, there have been only a handful of studies on applying process mining to manufacturing systems. Yang et. al. [27] and Viale et. al. [28] present a method to apply process mining on manufacturing data. While the former uses structured and unstructured data generated from manufacturing system along with operators or workers to provide domain level knowledge, the latter works with definitions of the system provided by domain experts to find inconsistencies between model and process. Yahya [29] shares interesting insights into using process mining to understand manufacturing sys-

tems using artificially created logs.

However, there seem to be two main factors missing when aiming to apply process mining to manufacturing systems:

1. Existing methods to generate usable data seem to be scarce.
2. There is a lack of a defined data structure/abstraction that proves beneficial for generating models from factory data.

The method for data generation discussed in this paper can potentially fill this missing gap, allowing to use process mining to analyze manufacturing systems.

In this paper process mining algorithms were run using ProM [10] – a process mining tool box from Eindhoven University of Technology. There are other alternatives, both, open-source and commercial versions, that can be used in this case [30]. The choice of tool, however, is due to convenience and that it is open-source containing a variety of algorithms to choose from.

In order to run the algorithms in ProM, the data is to be structured as tuples of <caseId,eventId,attributes>. Here, caseId is an identifier for each cycle in the station. Each cycle identified by the caseId maps to multiple eventId's corresponding to operations contained in the cycle. Attributes usually consisting of timestamp, resource, cellId, etc, are not mandatory but add additional value to the data during processing. Identification of different production cycles – that requires mapping caseID to each operation – is therefore of key importance in order to be able to perform further analysis.

4.2 Identifying different product cycles

Using tools such as process mining require the event log to be divided into different cycles. As described earlier, the ability to identify cycles is highly dependent on the complexity of the manufacturing station. The data collected can be made up of individual events, in which case it does not contain any information regarding the overall cycle.

A possible solution to handle this lack of information and achieve automatic identification of cycles is presented in [31], where the aim is to find cycles for a program that performs I/O access to a disk. The authors apply frequency analysis to the collected data to estimate events that signify a cycle change. Another example is [32] where the authors use probabilistic methods of estimation-maximization to infer cycles on a general unlabelled event log.

In the present paper the estimation-maximization proposed by [32] has been used. Data from the complete manufacturing station was used to detect cycles and the results were not promising. The reason for this was mainly due to existing parallelism between all the robots in the station. This parallelism results in non-deterministic sequences in the aggregated data. However, applying the

estimation-maximization method to the data from a single robot resulted in detection of sufficiently accurate cycles when compared to the actual cycles logged in the PLC database.

Each cycle can then be visualized as a Gantt chart for each robot. The result is shown for two robots (r8253 and r8256) in Figure 5, Robot r8253 makes 26 welds in three different regions and robot r8256 makes 10 different welds. However, analyzing each cycle is tedious and still does not present an overall system behaviour. A possible solution is to apply mining and inference algorithms.

The event `ActSoftServoTrap` is generated when the robot performs a weld spot, and is, for simplicity, not treated as a different event from its parent event usually ending with the name "weldDefault" that defines the region being welded. Even though this event might sometimes be useful for analyzing the system, it does not add value within this paper, as resulting visualizations of each robot are spaghetti-like and cannot be easily interpreted. Avoiding the event, results in an interpretable visualization.

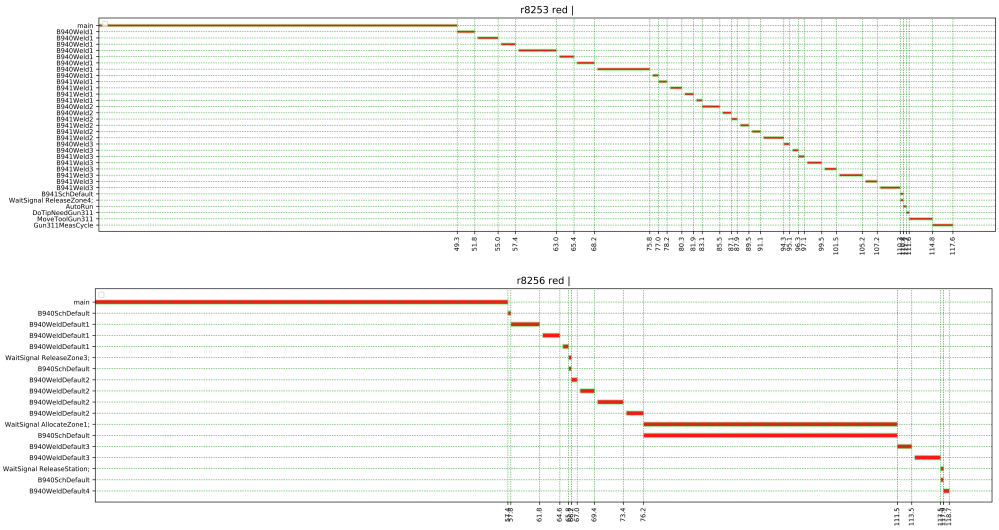


Figure 5: A Gantt chart plotted to show operations performed by two robots in one cycle of operation.

4.3 Operation view

Once cycles have been identified in the collected data, the operations are then grouped according to their respective cycles, resulting in the required format that can be used as input to ProM to find a general model that fits the data. Using the complete station data results in a large and unintelligible visualization. Hence, as a first step, we visualize the models obtained using 'routine' operations of a single robot.

A model is created using the "*Directly follows*" plugin in ProM, shown in Figure 6. The model visualizes the nominal behaviour of a single robot (r8255). Based on our knowledge about the station, we conclude that the three different sequences outlined in the figure represent the sequences unique for the three car models produced in the robot station. The operations outside the highlighted circles are common to all car models.

The nominal behavior described in Figure 6 lacks information regarding the relationship between the different operations. That is, there is no indication if two operations always occur after each other or if there is no dependency between them. To further see the relationship between the operations, and create a more generalized model of the data, the "*Inductive miner*" plugin in ProM is used. The resulting model is shown in Figure 7, here the relations between operations can also be deduced. The operations on the right hand side of the graph are executed only after at least one of those, in parallel, on the left have executed. The resulting models are large and cannot easily be accommodated within the limits of this paper, better quality images can be accessed along with the log files online [33].

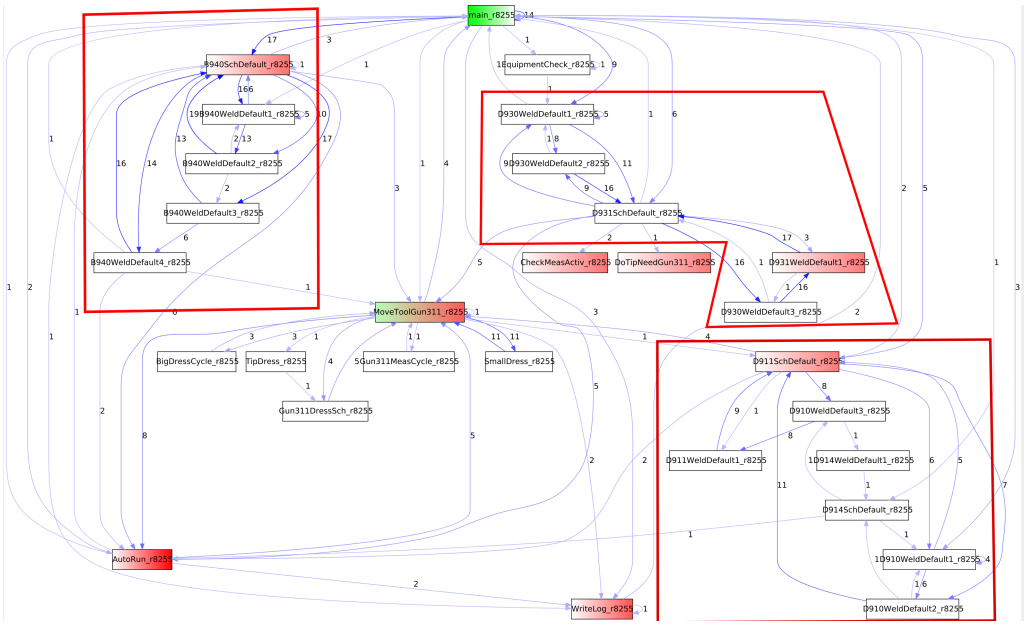


Figure 6: A simplified view of all observed sequences for robot `8255`. The three distinct sequences for each of the car models operated on by the robot are highlighted.

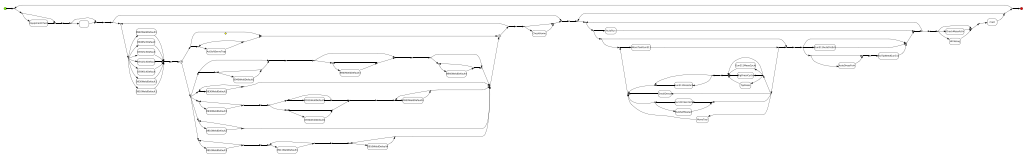


Figure 7: The result from the inductive miner plugin. Here we see the relations between different operations. A more legible image can be accessed online [33]

4.4 Resource view

Another point of interest would be to analyze product flow through the station. By that we mean, to visualize the order in which the robots would perform actions on the car being manufactured in the station.

The "*Inductive miner*" plugin is used here to process the complete data from the complete station. Figure 9 shows the resource view for the four different robots in the example station. As the station is simple and the robots are independent of each other, the resulting graph in Figure 9 shows four robots all in parallel indicating that there is no dependency between the robots.

Generating visual model for large complex stations will provide better understanding of the system and its flows. In order to highlight the advantages of having a good understanding of product flows a more complex manufacturing station is used. This station consists of six robots and can operate on three different products at a time. Figure 8 illustrates the product resource view for this complex station. When a product enters the station it is first serviced by two robots, `r8601` and `r8602`, in parallel. Then the product moves forward, to the second stage, where it is then serviced by three robots in parallel, `r8603`, `8604` and `r8605`. Once these three robots complete their task, the car moves to the next final stage where robot `r8606` performs some actions before leaving the station.

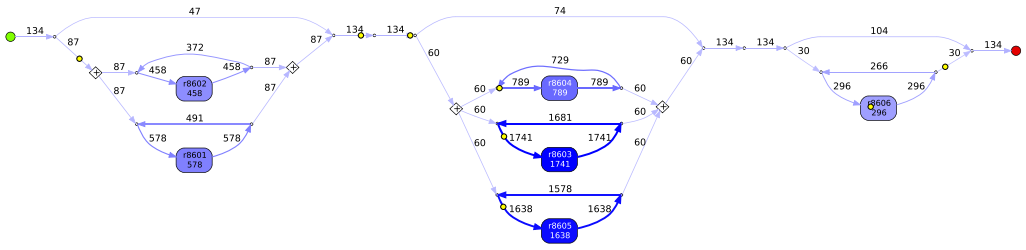


Figure 8: The resource centered view of a larger cell consisting of 6 robots. The figure shows the sequence in which a product entering the station will be serviced by the robots.

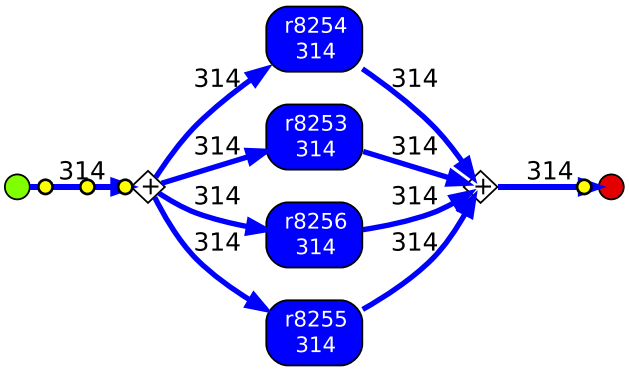


Figure 9: Viewing a resource centered view of the process, it can be deduced that the robots are independent of each other.

5 Conclusions and Future work

An architecture that enables capturing robot operation data from existing and new manufacturing stations was presented in this paper. The presented architecture was implemented on a robot station at an automotive manufacturing company using the software Sequence Planner. The implemented software makes it possible to listen to the robot program pointer to extract and abstract out the ongoing operation. This captured data, abstracted as operations, are then visualized in an online manner using Gantt charts. Which, potentially, will enable operators to track the progress of the station, thereby helping them during maintenance and error recovery tasks.

The data was also analyzed in an off-line manner by applying process mining techniques to understand the underlying behaviour – by creating models – of the station. We evaluated the use of process mining using the tool ProM to demonstrate the potential use of the abstracted data in order to create general models that represent the data. Two different views, of the data, were presented; one relating to the operation perspective and another representing the product flow between resources.

5.1 Future work

While process discovery techniques are beneficial during commissioning of manufacturing systems, a number of challenges need to be solved before they can be effectively used in regular operation. One problem is detection of cycles. The data aggregated does not indicate when a manufacturing cycle starts or completes. Hence, a pre-processing step before any further analysis is carried out is mandatory and poses a hindrance towards real-time model generation. The example shown in this paper, as a proof of concept, dealt with a simple station with

no dependencies between resources. Hence, it was relatively easy to identify cycles. However, in more complex stations, with several interacting resources and when each resource is dependent on the product flows from several different stations, the task of identifying cycles gets much more complicated. A major challenge remains to be solved before being able to automatically create models.

The methods discussed so far focus on obtaining data from robots. The natural next step is to also obtain data from PLC's that control the system. The authors are also not aware of any existing interfaces to access the PLC's in the manner described for robots in this paper. The data from PLC's is usually in the form of variable-value pairs and converting these, variable-value, pairs into operations is going to be challenging task.

6 Acknowledgements

This project was supported by ITEA3 VINNOVA ENTOC (2016-02716) and VINNOVA LISA 2 (2014-06258). Furthermore, the authors would like to thank Håkan Pettersson and Stefan Axelsson from Volvo Cars Corporation for their help and support during the implementation phase of this project.

7 Bibliography

[1] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan 2016, pp. 3928–3937.

[2] J. Campos, C. Seatzu, and X. Xie, *Formal methods in manufacturing*. CRC press, 2014.

[3] G. Frey and L. Litz, "Formal methods in PLC programming," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 4, 2000.

[4] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Integrated virtual preparation and commissioning: Supporting formal methods during automation systems development," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 1939–1944, 2016.

[5] M. Dahl, A. Albo, J. Eriksson, J. Pettersson, and P. Falkman, "Virtual reality commissioning in production systems preparation," in *22nd IEEE International Conference on Emerging Technologies And Factory Automation*, 2017.

[6] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *Journal of Computational Design and Engineering*, vol. 1, 2014.

[7] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[8] "Snap 7." [Online]. Available: http://snap7.sourceforge.net/

[9] A. Farooqui, P. Bergagard, P. Falkman, and M. Fabian, "Error handling within highly automated automotive industry: Current practice and research needs," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 9 2016.

[10] Process Mining Workbench, last accessed Mon Apr 3 10:30:08 2017. [Online]. Available: http://www.promtools.org/doku.php

[11] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Sequence planner: Supporting integrated virtual preparation and commissioning," in *The 20th World Congress of the International Federation of Automatic Control, 9-14 July 2017*, 2017.

[12] D. Nord and H. Wahlqvist, "The tweeting robot - collection and processing of data from industrial robots," Master's thesis, 2016.

[13] L. Ribeiro and J. Barata, "Survey paper: Re-thinking diagnosis for future automation systems: An analysis of current diagnostic practices and their applicability in emerging IT based production paradigms," *Comput. Ind.*, vol. 62, no. 7, Sep. 2011.

[14] A. Theorin, K. Bengtsson, J. Provost, M. Lieder, C. Johnsson, T. Lundholm, and B. Lennartson, "An event-driven manufacturing information system architecture," *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 547–554, 2015.

[15] "Apache Active MQ," 2017. [Online]. Available: http://activemq.apache.org

[16] "Apacha Kafka," 2017. [Online]. Available: https://kafka.apache.org/

[17] "ABB Robot SDK," 2017. [Online]. Available: http://developercenter.robotstudio.com

[18] K. Bengtsson, B. Lennartson, and C. Yuan, "The origin of operations: Interactions between the product and the manufacturing automation control system," *IFAC Proceedings Volumes*, vol. 42, 2009.

[19] S. Riazi, O. Wigström, K. Bengtsson, and B. Lennartson, "Energy and peak power optimization of time-bounded robot trajectories," *IEEE Transactions on Automation Science and Engineering*, 2017.

[20] N. Sundström, O. Wigström, S. Riazi, and B. Lennartson, "Conflict between energy, stability, and robustness in production schedules," *IEEE Transactions on Automation Science and Engineering*, vol. 14, pp. 658–668, 2017.

[21] W. van der Aalst, *Process Mining*. Springer Nature, 2016.

[22] W. van der Aalst *et al.*, "Business process mining: An industrial application," *Information Systems*, vol. 32, no. 5, pp. 713–732, 2007.

[23] W. M. P. van der Aalst, "Business process management: A comprehensive survey," *ISRN Software Engineering*, vol. 2013, pp. 1–37, 2013.

[24] R. S. Mans, M. H. Schonenberg, M. Song, W. M. P. van der Aalst, and P. J. M. Bakker, *Application of Process Mining in Healthcare - A Case Study in a Dutch Hospital*, ser. Biomedical Engineering Systems and Technologies. Springer Nature, 2008, pp. 425–438.

[25] A. Partington, M. Wynn, S. Suriadi, C. Ouyang, and J. Karnon, "Process mining for clinical processes," *ACM Transactions on Management Information Systems*, vol. 5, no. 4, pp. 1–18, 2015.

[26] E. Rojas, J. Munoz-Gama, M. Sepúlveda, and D. Capurro, "Process mining in healthcare: A literature review," *Journal of Biomedical Informatics*, vol. 61, pp. 224–236, 2016.

[27] H. Yang, M. Park, M. Cho, M. Song, and S. Kim, "A system architecture for manufacturing process analysis based on big data and process mining techniques," in *2014 IEEE International Conference on Big Data (Big Data)*, 10 2014.

[28] P. Viale, C. Frydman, and J. Pinaton, "New methodology for modeling large scale manufacturing process: Using process mining methods and experts' knowledge," in *2011 9th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, 12 2011.

[29] B. N. Yahya, "The development of manufacturing process analysis: Lesson learned from process mining," *Jurnal Teknik Industri*, vol. 16, no. 2, 2014.

[30] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *CoRR*, vol. abs/1705.02288, 2017.

[31] P. C. Diniz and D. R. Ferreira, "Automatic extraction of process control flow from i/o operations," in *Business Process Management*, M. Dumas, M. Reichert, and M.-C. Shan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 342–357.

[32] D. R. Ferreira and D. Gillblad, "Discovering process models from unlabelled event logs," in *Business Process Management*, U. Dayal, J. Eder, J. Koehler, and H. A. Reijers, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 143–158.

[33] "Access data online." [Online]. Available: https://chalmersuniversity.box.com/v/fromFactoryToProcess

Paper 3

Towards Automatic Learning of Discrete-Event Models using Queries and Observations.

Ashfaq Farooqui, Petter Falkman, and Martin Fabian.

Submitted for possible journal publication, 2018

Comment: The paper has been reformatted for readability, but is otherwise unchanged.

Towards Automatic Learning of Discrete-Event Models using Queries and Observations.

Ashfaq Farooqui, Petter Falkman, and Martin Fabian.

Abstract

Model-based techniques are being embraced by the manufacturing industry in their development framework. Though model-based approaches have advantages over existing methods, such as allowing offline verification and validation before physical commissioning, they also have their own challenges. Firstly, models are typically created manually and hence are prone to errors. Secondly, once a model is created, tested, and implemented within the factory floor, there is an added effort required to maintain and update the models. This paper is a preliminary study of the feasibility of automatically obtaining formal models from virtual simulations. Different techniques can be used to build models automatically, two of which are *active* and *passive* learning, respectively. We present the foundational algorithm from the active automata learning community to learn discrete-event models from virtual simulations. An abstract model in the form of operations is learned by applying this algorithm on a discrete simulation model. A major bottleneck identified when applying active learning is the generation of counterexamples. We present an approach to integrate active and passive learning; where passive models comprised of operation sequences observed during the nominal operation of the system, provide the counterexamples.

1 Introduction

The automotive manufacturing industry, and manufacturing industries in general, are gradually moving towards simulation-based techniques during the initial phase of setting up the manufacturing systems known as the *virtual commissioning* phase [1]. During this phase, a virtual model of the manufacturing station is first created in a simulation software and is tested in simulation to ensure correctness, before physically commissioning the station.

Also, model-based techniques that offer design, validation, verification, and testing, are being actively adopted within the manufacturing industry [2, 3] to formally ensure correctness of complex systems. These techniques rely on using a formal model that describes the station, which are then used to verify the

requirements and specifications of the station. The last few years have seen a drastic advancement in model-based algorithms, within both the more theoretical formal methods, as well as the application-oriented computer science domains. These advancements have rendered model-based methods feasible for use on regular manufacturing systems. Model-based techniques are usually coupled with virtual technologies such as simulations [4] and virtual reality [5] in the virtual commissioning phase [1], which leads to shorter physical commissioning times. However, creating formal models is a challenging task that requires skill, in-depth knowledge of the system, and creativity.

Models specifying the intended behaviour of the station need to be created early during the specification and design phases of virtual commissioning. Incorrect or incomplete models are misleading, and can unnecessarily complicate the development process. In reality, these models do not always capture the complete behaviour, and even when they do they become outdated as the station evolves over time. A possible method to deal with incorrect and outdated models could be to create them automatically. Several methods have been proposed in the literature dealing with automatically creating models. These methods work by either observing the behaviour of the components in the system, or by actively interacting with the system and then using specially designed techniques to construct a model. These techniques are now receiving increased attention within the verification and testing communities [6, 7], and are known as *automata learning* (a.k.a grammatical inference) [8, 9].

Automata learning techniques can be broadly classified into two types, *passive learning* and *active learning*. An application of passive learning was presented previously in [10]. Also, in [11] we demonstrated the feasibility of applying active learning to real-world systems. In active learning, a hypothetical model is first presented by querying the system; then an algorithm tries to find counterexamples that show that the model is incorrect with respect to the modeled system. The model is then automatically updated so as to also account for the suggested counterexample. This process is repeated till no counterexample can be found, meaning that that suggested model accurately captures the behaviour of the system. As there does not exist a behavioral model defining the system to compare with, finding counterexamples becomes a complex task. The algorithm defined in [11] exhaustively simulates all possible paths in the suggested model to find counterexamples. Exhaustive simulation of all the different behaviours in the simulator is time consuming. In this paper we look at alleviating the problem of finding counterexamples by extending the approach to let the active learner use the passive model.

1.1 Outline

Section 2 defines the terminologies used throughout the paper. Section 3 briefly summarizes the state of research for both active and passive algorithms, their shortcomings and their previous applications in manufacturing. Section 4 then presents the active learning algorithm L^*. We then propose an integrated approach in Section 5 to use both active and passive techniques towards finding counterexamples.The proposed approach is evaluated on a simulated robotic arm presented in Section 6. Finally, Section 7 concludes with some remarks and suggestions for future work.

2 Prerequisites

This section introduces the modeling formalisms *Automata* and *Operations*, together with the terminology used throughout the paper.

2.1 Alphabets, Words and Languages

Let Σ, known as the *alphabet*, represent a finite set of *symbols*, then Σ^* is used to denote the set of *words* of finite length over Σ including the empty word ε, i.e a set of sequences of symbols formed by *concatenation*. Concatenation of sets is represented using the dot ($\cdot$) operator, and concatenated symbols are written together without a space. For example, for two sets $A = \{a\}$ and $B = \{b\}$, set concatenation is represented by $A \cdot B$ and symbol concatenation by ab. Note that $A \cdot B = \{ab\}$. A language $\mathcal{L} \subseteq \Sigma^*$ contains the set of words over Σ including the empty word ε.

A set of words is said to be *prefix-closed* if the prefix of every member in the set is also a member. Suffix-closed sets are defined analogously.

2.2 Deterministic Finite State Automata

Definition 1 (DFA)**.** *An* automaton *is defined as a 4-tuple* $\langle S, \Sigma, \delta, s_q \rangle$*, where:*
S *is the set of states*
Σ *the alphabet contains the set of operations, as symbols*
δ *defines a transition function* $S \times \Sigma \to S$
s_q *is a set of states in* S *representing the marked states*

The marked language given by $\mathcal{L}_m \subseteq \mathcal{L}$, is the set of traces that result in a marked state. There are many automata that represent the same language, but it is known [12] that among all of these automata there is a *minimal* one, with the smallest number of states and transitions. This automaton is unique.

2.3 Operations

The system to be learned can perform several tasks, and these need to be predefined. To define the tasks we use the abstraction of *operations*. Each operation corresponds to one specific function performed by the target system.

Definition 2 (Operation)**.** *An* operation *is defined as a 4-tuple*

$$\langle PreGaurd, PreActions, PostGaurd, PostActions \rangle$$

where:
PreGuard, is a predicate over the state that defines when the operation is allowed to execute;
PreActions, defines assignments to configuration parameters in the state that will execute the operation in the target system;
PostGuard, is a predicate over the state that defines when the operation completes;
PostActions, defines assignments to configuration parameters in the state that ensures completion of the operation.

For simplicity, we will consider operations that are two-state. That is, when an operation is triggered it transforms the system from one state to another when the operation completes; no in-between states will be considered.

3 Background

A manufacturing line consists of several manufacturing stations, which are inturn built up of interacting parts such as robots, clamps, fixtures, and conveyors. Each station is able to perform several operations. These operations, executed by the specific resources in the station, are controlled by a PLC that controls the station. Based on the product requirements, the PLC selects a specific sequence of operations that need to be performed. The sequence of operations chosen are either hard-coded within the PLC or saved separately in a database that the PLC has access to. We would like to automatically create a formal model of all these hard-coded sequences in order to verify the requirements of the station. Additionally, the model can suggest a better or optimized sequence that can replace the older sequences. The following section looks at different ways to create these models by employing Grammatical Inference techniques.

Grammatical Inference [9] is associated with various fields of study, like computational linguistics, machine learning, formal learning theory, and computational biology, to name but a few. Hence, it is also known by different names depending on the field, such as Automata Learning, Grammatical Induction,

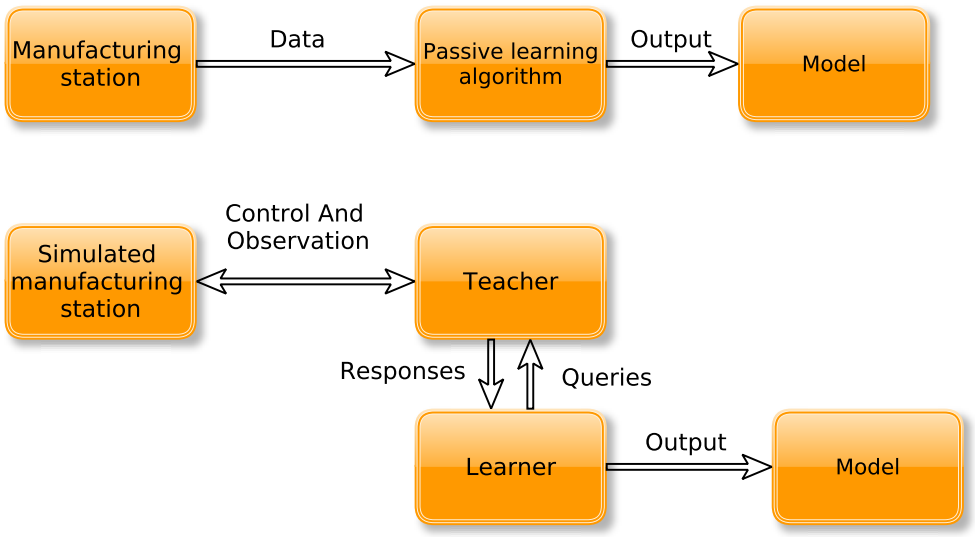


Figure 1: Difference between active and passive methods

Grammar Learning etc. An in-depth survey of grammar inference techniques is provided in [8, 13, 14].

A large body of work already exists in this field and can be classified into two categories, passive and active learning. Both these methods employ a learning algorithm we call the *learner* that creates the model either by processing the logs offline in case of passive learning, or by actively querying the system in case of active learning. Figure 1 visualizes the difference between the two. Passive learning deals with first collecting data from the station, and then by processing this data to build the model. Active learning is an interactive method where the learner interacts with the actual system to build the model.

3.1 Passive learning

Passive learning algorithms take as input sequences observed from the target system. The algorithm then builds a prefix-tree-acceptor (PTA), which is a tree-like DFA, built by considering all prefixes of an observation as states. The PTA is updated on every iteration, where states are merged based on certain heuristics. Passive learning algorithms are divided into *Exact Algorithms* and *Approximate Algorithms*.

Exact algorithms aim to create a model that exactly represents the input data. Some of the algorithms under this category are MMM [15], BICA [16], and EXBAR [17] – all of which start with a basic trivial model as initial hypothesis. Then, for every observed sequence of operations, a search is performed to find deviations between the model and the observations. If a deviation is found,

the hypothesis is updated so as to contain the new observation. This process is repeated for all observations in the log. The primary differences in these algorithms lies in the type of search performed to find deviations.

Approximate algorithms, on the other hand, rely on certain heuristics or decision techniques to provide a hypothesis, which is approximately close to reality. Examples of these algorithms are EDSM [18], SAGE [19] and ED-BEAM [17]. These algorithms work in similar ways to their exact counterparts, the difference lies in the way they perform the state merging. Additionally, these algorithms keep updating the hypothesis by backtracking and improving on previous merges. In [20] a general survey of state merging algorithms is presented and [9] introduces learning algorithms in general. Passive algorithms require event logs with both positive and negative outcomes in order to return a valid generalized model of the system.

Knowledge regarding the sequences of operations performed by the station is helpful in developing a model for it. A method for collecting data from robots of a manufacturing station is discussed in [21]. It describes an architecture in which the actions performed by the robots are captured using predefined adapters that connect to these robots. The actions are then processed and abstracted as operations. Each operation is appended with extra information such as start and stop time, and the name of the robot that performed the operation. Similarly, PLC commands are captured by listening to variable changes using the OPC-UA [22] protocol and store the ongoing operations to build observed sequences. Passive learning then uses several sequences together to create a model that captures the behaviour of the station.

Building models, using the data captured from robots in [21], is presented in [10], where the process mining toolbox called ProM [23] was used to create models from the captured data. The models obtained describe commonly observed sequences. To this end, mining algorithms use heuristics to create the model. Infrequent behavior is therefore treated as noise in the data and hence discarded.

The resulting models obtained in [10] provide insight into the behaviour of the manufacturing station. The different perspectives presented to analyze the behaviour are: relations between various operations, relations between the operations and resources, and the product flow through the station. However, the model did not capture any unseen behaviour of the station. In order to be able to capture a more general model we turn towards active learning techniques.

3.2 Active Learning

Algorithms where the learner actively queries the target system and learns more about it at every iteration are classified as *active learning*. The learner does not

have access to all the data to begin with, rather it has an interface to the target system in the form of a *teacher*. The learner first starts with an initial hypothesis, which is further improved at each stage by querying the actual system.

A seminal paper in the field of active learning is Dana Angluin's work on learning minimal DFA's using queries and counterexamples [24]. The outcome of this paper is the L^* algorithm that allows the learner to pose two types of questions to a teacher. Membership queries ask whether the language is accepted or not, and equivalence queries ask if the suggested hypothesis accurately represents the system. If the teacher answers "yes" to an equivalence query, the algorithm terminates and the current hypothesis is returned. Else, the teacher provides a counterexample disproving the given hypothesis. The hypothesis is then updated taking the counterexample into consideration. The algorithm repeats these steps and terminates when a valid hypothesis is found. Queries for membership are fairly easy to handle; equivalence queries on the other hand are proved to be a NP-complete problem [25].

From an algorithmic perspective, there have been only a handful of improvements and new approaches suggested in this field. Schapire [26] improves the L^* algorithm by handling the counterexamples that include a homing sequence when it is not possible to reset the target system. Kearns and Vazirani [27] introduce the idea of discrimination trees, which is further used by Malte et al. [28] who suggest the TTT algorithm.

From a more practical perspective, [24] has inspired tremendous amount of work that has yielded positive results. Active automata learning has been applied to verify communication protocols using Mealy machines [29, 30]. By using a suitable abstraction interface Arts [31] learn IO automata. Other techniques are directed towards learning models of software systems. Malte et al. [32] apply active automata learning towards learning models of software programs modeled as register automata, while Smeenk et al. [33] focus on learning embedded software programs.

A method to run active learning algorithms was presented in [11] using a simulated model of the actual system. The test was performed on a robotic arm that could perform eight different operations on a given matrix grid. The task was to learn the language of the simulated system using the L^* algorithm. The limitation of this method was to find counterexamples, which were obtained using a random walk algorithm. Counterexamples were generated by randomly choosing one transition to walk along the hypothesized DFA, and then running the same operation in the simulation until an inconsistency that disproves the hypothesis was found. This method worked well for smaller systems where the number of possible operations were few. However, as the number of possible operations increased, the processing time grew aggressively. Additionally, the algorithm would end up in loops and not terminate. This is a major problem

when applying active learning techniques to real-world systems. A possible step forward would be to leverage benefits of passive learning to improve actively learned models.

4 The L^* Algorithm

The L^* algorithm [24] learns a minimal DFA accepting a regular language $\mathcal{L}_m \subseteq \Sigma^*$ over a finite alphabet Σ. Two types of queries need to be answered to make the algorithm work, and these are answered by the teacher. For pedagogic reasons, we will refer to the algorithm as the learner that interacts with the teacher using queries. The two types of queries made by the learner are:

- **Membership queries:** given a word $w \in \Sigma^*$, the teacher replies positively if w belongs to $\mathcal{L}_m$, else the reply is negative.
- **Equivalence queries:** given a hypothesis DFA $\mathcal{H}$, the teacher must verify if $\mathcal{H}$ accurately represents the language $\mathcal{L}_m$. If not, the teacher must provide a counterexample $c \in \Sigma^*$, such that, c is incorrectly accepted or rejected by $\mathcal{H}$.

The algorithm terminates when the teacher cannot find such a counterexample. The L^* algorithm is outlined in Figure 2.

At any given time the learner updates its knowledge about the target language. Internally, this knowledge is represented as an *observation table*. The observation table has three parts, a non-empty finite prefix-closed set S, a non-empty finite suffix-closed set E, and a transition function T mapping $((S \cup S.\Sigma).E)$ to $\{0, 1\}$. An example is shown in Table 1. The table is made up of two parts: the top part corresponds to rows ranging over a finite set $S \subseteq \Sigma^*$; and the lower part, with rows ranging over $S.\Sigma$ i.e words of the form sa, where $s \in S$ and $a \in \Sigma$. Columns range over a finite set $E \subseteq \Sigma^*$. The function T is then the values represented by this table. If $u \in S \cup S.\Sigma$ and $v \in E$, then $T(u)(v)$ – the value in the cell corresponding to the row u and column v – results in 1 if $uv \in \mathcal{L}_m$ else 0. The cells in the observation table are filled using membership

		E		
		ε	a	aa
S	ε	0	1	0
$S.\Sigma$	a	1	0	0
	b	0	0	0

Table 1: A simple observation table where at least a is contained in $\mathcal{L}_m$.

queries. Table 1, with $\Sigma = \{a, b\}$, indicates $\mathcal{L}_m$ accepts at least a, and does not contain the words $\varepsilon,\ b,\ aa,$ or aaa.

The learner uses the observation table to construct a hypothesis DFA from the different rows in the table, where the rows represent the states of the DFA. Rows with the same content result in a single state. The marked states are represented by $row(s)$ where $s \in S$ and $T(s)(\varepsilon) = 1$. The initial state corresponds to a row with the empty word, i.e $row(\varepsilon)$.

In order to be able to construct a hypothesis, the observation table needs to be *closed* and *consistent*.

- An observation table is said to be **closed** if for all $t \in S, a \in \Sigma$ there is an $s \in S$ such that the $row(s) = row(t.a)$. In other words, each transition reaches some state in the hypothesis.
- A table is **consistent** if for $s_1 \in S$ and $s_2 \in S$ and $row(s_1) = row(s_2)$ then for all $a \in \Sigma, row(s_1.a) = row(s_2.a)$. In other words, there is no ambiguity in the transition.

The learner ensures that the table is closed and consistent by updating the sets S and E according to the algorithm in Figure 2. Once the table is closed and consistent, the learner submits a hypothesis to the teacher requesting an equivalence query. If the teacher provides a counterexample the observation table is updated with the traces from the counterexample, new queries are made, and a revised hypothesis is submitted when the table is again closed and consistent. This process is repeated until a hypothesis is accepted by the teacher.

Example run

To explain the above algorithm an example is presented in this section. Consider a language $\mathcal{L}_m = \{x \in \{a, b\}^* |$ the number of a's and b's are even$\}$. When the algorithm starts, the table is empty and represented by $S = E = \varepsilon$. Figure 3 shows the evolution of the observation table and the hypothesis. The learner starts with an empty table. Using membership queries it constructs the first closed and complete table as seen in Figure 3b, presenting a hypothesis to the teacher. The teacher then replies with a counterexample $abab$. The counterexample is taken into consideration and the observation table and hypothesis are further updated to the one seen in Figure 3c. At this point the hypothesis is accepted by the teacher as the model is complete.

5 Towards Integrating Active and Passive learning

Passive learning algorithms result in models that contain only the observed sequences, in contrast to the models obtained using active learning. The time

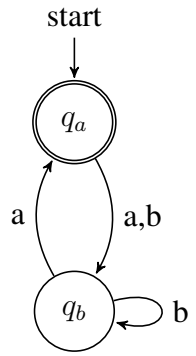
Result: A Hypothesis DFA $\mathcal{H}$
initialization $S, E \leftarrow \varepsilon$;
repeat
  while *the table is not closed or not consistent* **do**
    if *table is not closed* **then**
      find $u \in S, a \in A$ such that $row(ua) \neq row(s) \forall s \in S$;
      $S \leftarrow S \cup \{ua\}$;
    end
    if *table is not consistent* **then**
      find $s_1, s_2 \in S$, $a \in A$ and $e \in E$ such that
       $row(s_1) = row(s_2)$ and $row(s_1 ae) \neq row(s_2 ae)$;
      $E \leftarrow E \cup \{ae\}$;
    end
  end
  Construct the hypothesis $\mathcal{H}$ to the teacher **if** *the teacher replies* no *with a counterexample* c **then**
    $S \leftarrow S \cup prefixes(c)$
  end
until *the teacher replies* yes;
return $\mathcal{H}$

Figure 2: The L^* Algorithm

	ε
ε	1
a	0
b	0

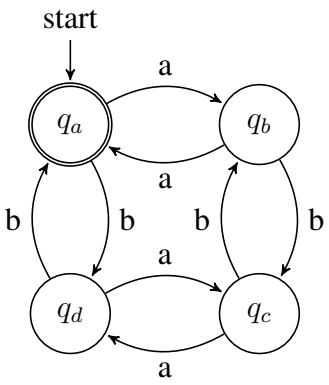
(a) Initial table

	ε
ε	1
a	0
b	0
aa	1
ab	0



(b) First hypothesis

	ε	a	b
ε	1	0	0
a	0	1	0
ab	0	0	0
aba	0	0	1
$abab$	1	0	0
b	0	0	1
aa	1	0	0
abb	0	1	0
$abaa$	0	0	0
$ababa$	0	1	0
$ababb$	0	0	1



(c) Updated table with counterexample and the second hypothesis

Figure 3: Evolution of the observation table and hypothesis while learning a language which accepts even number of a's and b's

needed for successfully running active learning algorithms is significantly higher, though. The contributing factor to this is finding good counterexamples, which relies on exhaustively simulating all possible scenarios.

To tackle this challenge, we integrate active and passive learning so that counterexamples can be found from passively learned models we call this method of learning L^+. Figure 4 illustrates the data flow. Data from a live manufacturing plant is collected for passive learning purposes. A simulated model of the same manufacturing station is connected to an implementation running the L^* algorithm, through a teacher. The teacher also has access to the passively learned model.

In order to find a counterexample, the teacher needs to compare the two models and find the paths that exist in the passive model but not in the hypothesis generated by the L^* learner. That is, rather than performing random walk on the

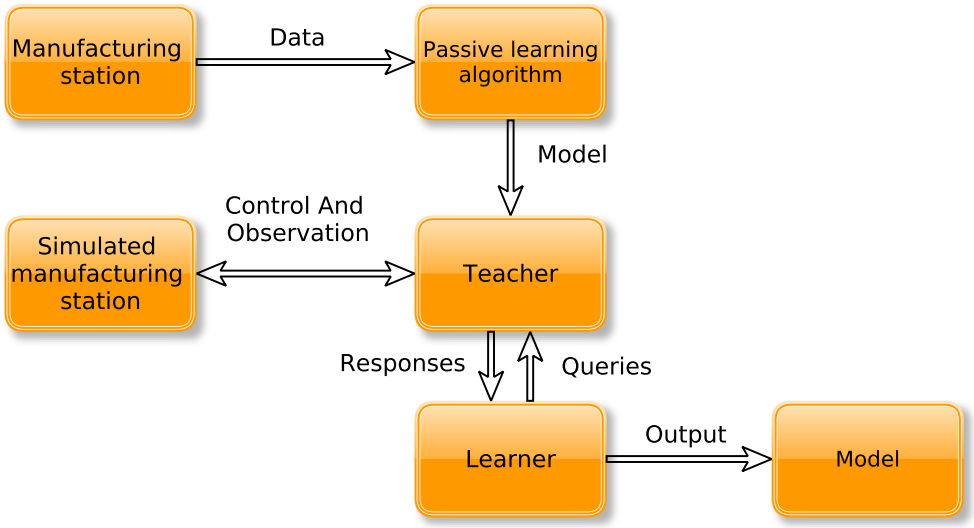


Figure 4: Proposed structure for L^+

generated hypothesis model and the simulation, as done in [11], we use the observed operation sequences from the passive model. A passive model is obtained by running a sufficient number of diverse cases so as to collect all the nominal sequences. If any of these sequences do not conform with the hypothesis presented by the learner, then a counterexample is found.

Figure 5 describes an algorithm to find counterexamples given a passive model. We first extract the unique sequences found in the passive model; it should be noted here that, for simplicity, loops are not taken into consideration. Then iterating over the list of sequences, for each sequence check if it exists in the hypothesis. If it does not exist we have found a counterexample. The algorithm terminates when there are no more sequences or if a counterexample is found.

This method, of using passively learned models along with active learning, improves the speed of learning. This improvement is because we no longer need to run the simulation to find counterexamples. Though there is an improvement in the time required to learn, this method is not perfect. As the collected data represents only valid operation sequences, a different method to find invalid operation sequences will still be needed. Additionally, for this to work, we assume that the complete nominal behaviour is captured in the passive model.

Another problem arises when there are loops present in the passive model. If so, the number of unique valid sequences generated from the passive model is unbounded. Hence, it is better to refine the sequences to avoid repeatedly traversing along loops. For larger systems it would be beneficial to use graph isomorphism techniques [34] that can compare models graphically to find counterexamples.

Data: The passive model, M
Input : Hypothesis H
Output: A sequence of operations that define a CE
Initialization $L \leftarrow$ unique paths in M
repeat
  $l \leftarrow$ first sequence in L;
  if l *exists in* H **then**
    remove l from L
  else
    CE is found l
  end
until *a CE is found or* L *is empty*;
if *CE found* **then**
  return l
else
  return *no CE exists*
end

Figure 5: Finding a counterexample (CE) from the captured nominal operation sequences

6 L^+ learning applied to a robotic arm

In this section we apply the L^+ learning method previously defined to construct a model of a simulated robotic arm. The arm can move in the X and Y directions, and is fitted with a gripper that allows the arm to grip objects. The gripper has the ability to extend and retract in order to grip. The program contains the required logic to control the arm, which includes moving the arm in four directions: up, down, left, and right; extending, retracting, closing, and opening the gripper.

The different operations are controlled by sending commands to the simulator. Additionally, the simulator program keeps track of the current position of the arm in both the X and Y direction using sensors. Figure 6 provides an image of the simulation.

In order to be able to learn the model of this robotic arm, the teacher must be able to communicate with the simulation environment. Furthermore, the algorithm requires the alphabet (the operations) as input, and also a technique to generate counterexamples. The experimental setup can find counterexamples using two alternative approaches, from the passive model, as well as the simulated system. In this section we explore these components and provide the end result from the learning algorithm.

The example will be treated as a purely discrete system with no parallelism. That is, only one operation is allowed to execute at any given time.

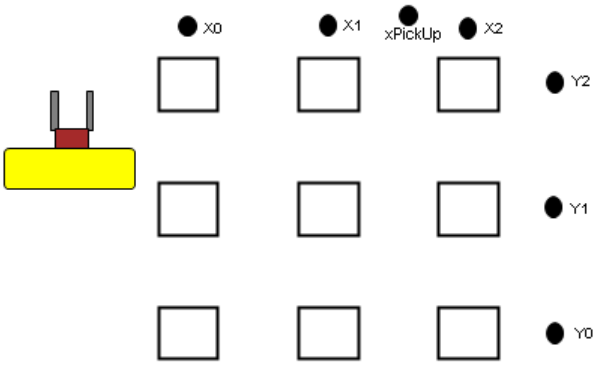


Figure 6: Simulation of the robotic arm.

6.1 Defining the system

Operations performed by the simulation need to be defined for the teacher and the learner in the format specified in Section 2. The guard predicates for each operation must specify exactly when the operation is allowed to execute and when it can finish. Correspondingly, the actions need to specify the variable assignments to start and stop the operations. The guards ensure that only one operation can execute at a time, even though it is possible that several operations are enabled. Additionally, the guards must ensure that the arm cannot move when it is extended, and can open and close the gripper only when it has already extended.

The goal is constructed according to what we want to learn about the system. For simplicity, we define the goal to be the initial state of the arm.

6.2 Results and Discussions

Using the setup as defined above the learning algorithm was capable of learning a model of the robotic arm. In order to learn how the algorithm scales as the simulation grows, the arm was allowed to move along different sizes of the grid.

To be able to build models passively we generated random sequences of observations from the simulator. Then, using ProM [23], a tool for process mining, models were created to represent the data. Figure 7 shows the model obtained using the inductive miner plugin, from ProM. The model was developed by generalizing 354 observed sequences, of which about 80 were unique, from a grid size of 6x6. The model represents the general relationship between the different available operations. For example, `grip` and `release` can occur only when `extend` has occurred. Also, the movement operations cannot occur when the arm is already extended. However, this passive model does not capture any information regarding the grid size.

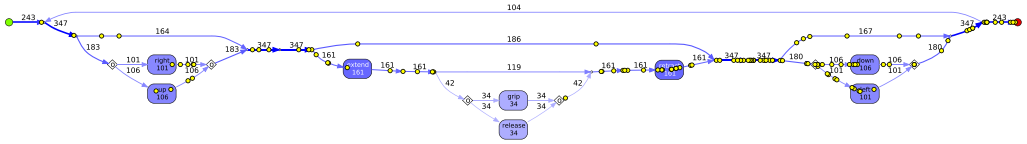


Figure 7: Model learned from observed sequences using ProM

In order to capture the grid in the model, we turn towards active learning. First, we create a model of the system using the simulation environment to find counterexamples. Figure 8 presents the learned model while learning the four operations related to gripping, while Figure 9 shows the model obtained on learning only the operations involved in motion. Both these were learned against a grid size of 3x3, seen in Figure 6. The model obtained is more accurate compared to the passively learned model, but finding counterexamples by simulation is time consuming as each sequence needs to be run in the simulator. Each operation requires at least 2-3 seconds to complete. Based on the number of sequences simulated and their length the time to simulate would be several minutes.

		L^+			L^*		
Grid	States	obs	Eq	Mq	Time(s)	Eq	Mq
2x2	17	30	3	5800	420	5	2980
3x3	37	30	4	12800	2530	8	17600
4x4	65	45	7	38400	4290	9	55230
5x5	101	34	8	59800	6400	10	102780

Table 2: Observations on the performance of L^* and L^+ while learning the model of the simulated robotic arm.

In order to avoid running simulations to find counterexamples we used sequences observed from the real system. We randomly generated several observation sequences to build a passive model. This model was then used to find counterexamples as explained previously in this paper.

Table 2 presents our observations from applying the L^* algorithm and the L^+ algorithm. The total number of states corresponding to the grid size is shown in columns *states* and *grid*, respectively. The column *obs* contains the number of unique observations that were captured to be processed in a passive model. *Eq* and *Mq* represent equivalence and membership queries, respectively. The

Figure 8: Learned model of extending and gripping

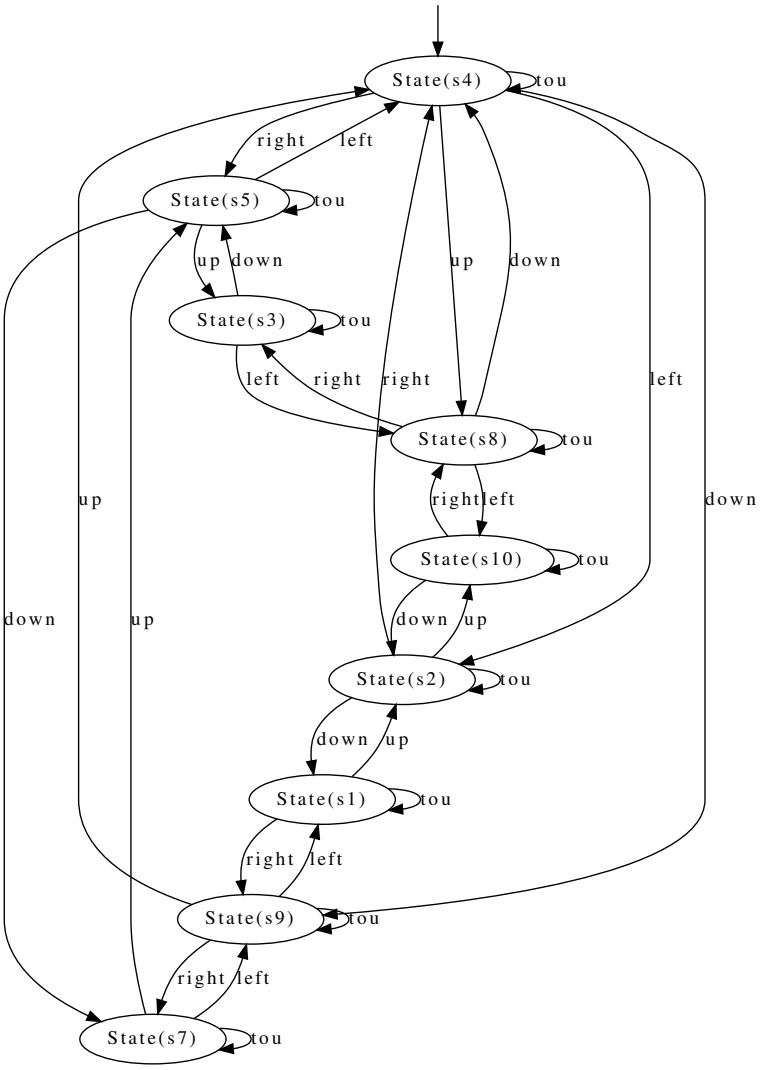


Figure 9: Learned model consisting of the four movements

total time taken to find counterexamples is under the column *time*, measured in seconds. This time corresponds to simulation time, where each operation is assumed to take 2 seconds to complete. Experimentally, we found that the number of equivalence queries were lesser when using a passive model. However, the difference was not large. The number of membership queries on the other hand showed a significant variation. This was because the counterexample and all its prefixes need to be populated in the observation table. If the counterexample is long and several of its prefixes are observed for the first time, the learner would require more membership queries to complete the table. On the other hand, if the the prefixes of the counterexample already exist in the observation table, the number of membership queries reduces. While in the case of L^*, the random walk determines the length of the counterexample. If it has a higher restart probability, restarting frequently, then the counterexamples are smaller in length. The comparison provided corresponds to the L^* in which the random walk algorithm had a restart probability of 20%.

The resulting models were no different from the models obtained by using only the L^* algorithm.Larger models required an increased number of observations to ensure all possible sequences were captured. Similarly, In the case of L^*, as the models grew in size, the random walk algorithm needed to have lower restart probability guaranteeing longer sequences.

7 Conclusion and Future work

In conclusion, this paper presented an approach to learn discrete-event models using active and passive learning techniques. To this end, the L^* algorithm, a fundamental active learning algorithm, was briefly described. A major bottleneck of applying this learning technique is to find counterexamples. This paper presented an approach – known as L^+– integrating active and passive learning techniques to help find counterexamples. Furthermore, a simulated robotic arm was learned using the techniques presented in this paper.

While using the observed sequences as counterexamples it was noted that the number of membership queries was usually more than needed. This increase depended on the size of the counterexample chosen. If a long sequence was chosen, then the observation table needs to be populated with the sequence and all its prefixes. This increases the number of membership queries needed to populate the observation table. A possible solution to this would be using the observation sequences in increasing order of size. However, it would potentially increase the number of equivalence queries. Finding a good heuristic to determine the quality of counterexamples will be of interest in this case.

According to the experiences from these experiments, several lines of work are identified that will help towards applying these algorithms practically. A

much more robust way to choose counterexamples is of prime importance. As it currently relies on randomized algorithms. Using ideas from Supervisory Control Theory [35] to synchronize the two models, modified such that some events are rendered uncontrollable, and then checking for controllability would result in counterexamples. This is particularly useful when the system grows large.

Another line of work aims at finding metrics to determine the quality of models learned. Metrics that enable us to determine the level of abstraction in a model as well as its usability, so as to be able to compare models learned using active and passive methods, respectively.

The third line of work, and maybe the most challenging, is to learn richer formalism in particular Extended Finite State Machine (EFSM) [36].

8 Bibliography

[1] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *Journal of Computational Design and Engineering*, vol. 1, 2014.

[2] J. Campos, C. Seatzu, and X. Xie, *Formal methods in manufacturing*. CRC press, 2014.

[3] G. Frey and L. Litz, "Formal methods in PLC programming," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 4, 2000.

[4] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Integrated virtual preparation and commissioning: Supporting formal methods during automation systems development," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 1939–1944, 2016.

[5] M. Dahl, A. Albo, J. Eriksson, J. Pettersson, and P. Falkman, "Virtual reality commissioning in production systems preparation," in *22nd IEEE International Conference on Emerging Technologies And Factory Automation*, 2017.

[6] C. Y. Cho, E. C. R. Shin, D. Song *et al.*, "Inference and analysis of formal models of botnet command and control protocols," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 426–439.

[7] T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, and B. Steffen, "On the correspondence between conformance testing and regular infer-

ence," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2005, pp. 175–189.

[8] C. de la Higuera, "A bibliographical study of grammatical inference," *Pattern Recognition*, vol. 38, no. 9, 2005.

[9] ——, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[10] A. Farooqui, K. Bengtsson, P. Falkman, and M. Fabian, "From factory floor to operation models: An approach to generate, transform, and visualize manufacturing systems," 2018, submitted to Journal of Manufacturing Science and Technology (CIRP-JMST).

[11] A. Farooqui, P. Falkman, and M. Fabian, "Towards automatic learning of discrete-event models from simulations," in *14th IEEE Conference on Automation Science and Engineering (CASE 2018)*, 2018, submitted.

[12] J. E. Hopcroft, R. Motwani, Rotwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computability*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[13] M. Bugalho and A. L. Oliveira, "Inference of regular languages using state merging algorithms with search," *Pattern Recogn.*, vol. 38, no. 9, 2005.

[14] R. Parekh and V. Honavar, "Grammar inference, automata induction, and language acquisition," *Handbook of natural language processing*, pp. 727–764, 2000.

[15] A. L. Oliveira and S. Edwards, "Limits of exact algorithms for inference of minimum size finite state machines," in *Proceedings of the 7th International Workshop on Algorithmic Learning Theory*, ser. ALT '96, 1996.

[16] A. L. Oliveira and J. a. P. M. Silva, "Efficient algorithms for the inference of minimum size DFAs," *Mach. Learn.*, vol. 44, no. 1-2, pp. 93–119, Jul. 2001.

[17] K. J. Lang, "Faster algorithms for finding minimal consistent DFAs," Tech. Rep., 1999.

[18] S. M. Lucas and T. J. Reynolds, "Learning DFA: evolution versus evidence driven state merging," in *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, vol. 1, Dec 2003, pp. 351–358.

[19] H. Juillé and J. B. Pollack, "A stochastic search approach to grammar induction," in *Grammatical Inference*, 1998.

[20] M. Bugalho and A. L. Oliveira, "Inference of regular languages using state merging algorithms with search," *Pattern Recogn.*, vol. 38, no. 9, Sep. 2005.

[21] A. Farooqui, K. Bengtsson, P. Falkman, and M. Fabian, "Real-time visualization of robot operation sequences," in *2018 IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2018)*, 2018.

[22] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[23] Process Mining Workbench, last accessed Mon Apr 3 10:30:08 2017. [Online]. Available: http://www.promtools.org/doku.php

[24] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87 – 106, 1987.

[25] S. Goldman and M. Kearns, "On the complexity of teaching," *J. Comput. Syst. Sci.*, vol. 50, 1995.

[26] R. E. Schapire, *The Design and Analysis of Efficient Learning Algorithms*. Cambridge, MA, USA: MIT Press, 1992.

[27] M. J. Kearns and U. V. Vazirani, *An Introduction to Computational Learning Theory*. Cambridge, MA, USA: MIT Press, 1994.

[28] M. Isberner, F. Howar, and B. Steffen, "The TTT algorithm: A redundancy-free approach to active automata learning," in *Runtime Verification*, B. Bonakdarpour and S. A. Smolka, Eds. Springer International Publishing, 2014, pp. 307–322.

[29] B. Steffen, F. Howar, and M. Merten, "Introduction to active automata learning from a practical perspective," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2011, pp. 256–296.

[30] B. Jonsson, *Learning of Automata Models Extended with Data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 327–349.

[31] F. Aarts and F. Vaandrager, "Learning I/O automata," in *CONCUR 2010 - Concurrency Theory*, P. Gastin and F. Laroussinie, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 71–85.

[32] M. Isberner, F. Howar, and B. Steffen, "Learning register automata: from languages to program structures," *Machine Learning*, vol. 96, no. 1, pp. 65–98, Jul 2014.

[33] W. Smeenk, J. Moerman, F. Vaandrager, and D. N. Jansen, "Applying automata learning to embedded control software," in *Formal Methods and Software Engineering*. Springer International Publishing, 2015.

[34] U. Schöning, "Graph isomorphism is in the low hierarchy," *Journal of Computer and System Sciences*, vol. 37, no. 3, pp. 312 – 323, 1988.

[35] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[36] R. Malik, M. Fabian, and K. Åkesson, "Modelling large-scale discrete-event systems using modules, aliases, and extended finite-state automata," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 7000 – 7005, 2011, 18th IFAC World Congress.