Technische Universität München – TUM School of Engineering and Design

Command Signal Configuration for Control Strategies of Discrete Production Systems

Jens Otto

Vollständiger Abdruck der von der
TUM School of Engineering and Design
der Technischen Universität München zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften
genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. Markus Lienkamp

Prüfer*innen der Dissertation:

1. Prof. Dr.-Ing. Birgit Vogel-Heuser

2. Prof Dr. rer. nat. Oliver Niggemann

Die Dissertation wurde am 19.04.2021 bei der Technischen Universität München
eingereicht und durch die
Fakultät für Maschinenwesen am 02.09.2021 angenommen.

Contents

1. Introduction

Optimal configuration of control strategies for discrete production systems is manually impossible due to the increasing frequency of changes. This thesis addresses this challenge with a combination of machine learning techniques and constrained optimization for the configuration of transport, storage, and pick-and-place systems. The novel approach finds optimal command signals without predefined behavior or simulation models. This chapter describes the motivation and scope of command signal configuration for control strategies in discrete manufacturing and builds on previous work by the author, and extends the published concepts [ON15, OVN16, OVN18b, OVN18a].

Section 1.1 introduces the motivation and scope of this thesis. Section 1.2 describes the requirements for an optimal command signal configuration solution. Section 1.3 explains how to read this work.

1.1. Motivation and Scope

Cyber-physical systems address the requirements of production systems; they are networks of software and hardware components that control time-dependent and concurrent physical processes [Lee08, Cal+17, Bos+17, Kar+18, Lei+18, Kar+19]. Cyber-physical production systems should adapt to new or changing production goals, such as new products or product variants, without requiring extensive manual engineering effort [RSV16, VH16].

In discrete manufacturing, cyber-physical production systems consist of modular hardware components such as robots, conveyors, compressors, laser units, fillers, or storage systems. The example production system shown in Figure 1 consists of three heterogeneous and reusable hardware components: a filler M_1, a robot M_2, and a conveyor system M_3. The production goal of this production system is to fill a bottle with material; for this purpose, filler M_1 fills a bottle, robot M_2 picks up and places the bottle on conveyor M_3, and conveyor M_3 transports the bottle to another location. Modular hardware design is outside the scope of this work, but from an automation perspective, modular hardware components must be mirrored by modular software components. A software component is an organizational unit of an automation software structure and implements a hardware component behavior, for example: fill, $pick$ and $move$ (see Figure 1: Automation software structure $A_1 - A_3$). In this work, the behavior of a software component is called “activity.”

The production goal is described by a behavior model that encodes sequences of activities. The aim of this work is not to introduce another modeling language. Inspired by [KST14] and the analyzed related work in model-based development, a behavior model can be described by a UML 2 activity diagram. Each swim lane reflects a hardware component, each activ-

ity reflects a software component, and fork nodes and merge nodes represent concurrent behaviors.

A software component (SWC) contains one or more control methods with command signals that implement the behavior of an activity with different control strategies (see Figure 1: A_1 : *fill-B1*(P_{11}), A_2 : $pick$-$C1(P_{21})$ or $pick$-$C2(P_{22})$, A_3 : $move$-$D1(P_{31})$ or $move$-$D2(P_{32})$). For example, in some production scenarios, a slow $pick$ activity with high precision is implemented by control method $pick$-$C1(P_{21})$ is required, and in other production scenarios, a fast $pick$ activity with low precision implemented by control method $pick$-$C2(P_{22})$ is sufficient, and so on.

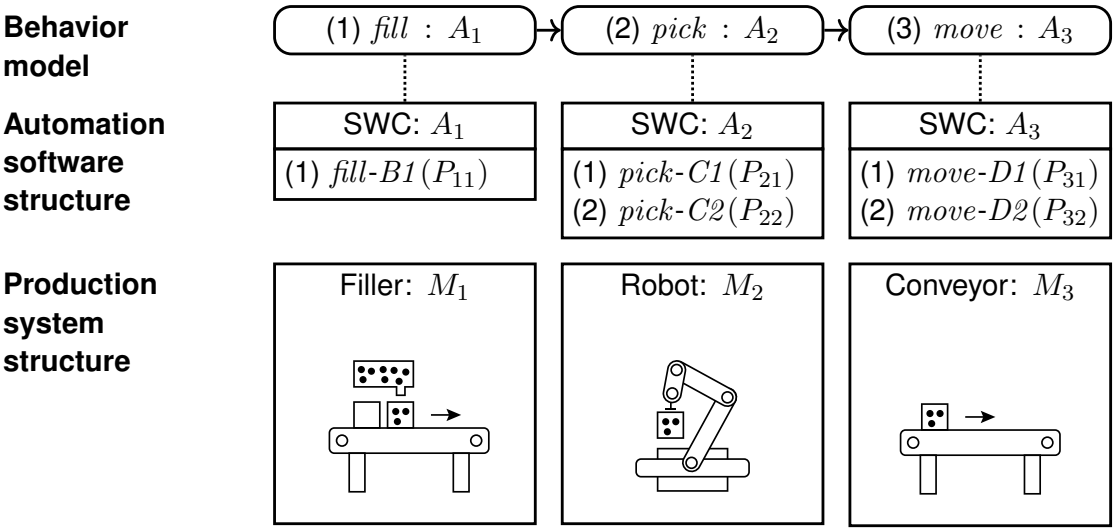


Figure 1 Concept of adapting reusable software components $A_1 - A_3$ with control methods A_1 : *fill-B1*, A_2 : $pick$-$C1$ or $pick$-$C2$, A_3 : $move$-$D1$ or $move$-$D2$ that implement different control strategies and command signals P_{11}, P_{21}, P_{22}, P_{31}, P_{32} to new or changing production goals (adapted from [OVN18b]).

The general idea is to increase the reusability of software components by adding degrees of freedom in the form of different control strategies with parameters (command signals) [HHL11, HHL13, ON15, OVN16, OVN18b, OVN18a]. The software components can then be configured for a specific production scenario and do not need to be rewritten for each adaption.

The scope of this work is a novel approach to machine learning and constrained optimization of command signal configuration that automates the following two tasks:

(i) Selection of optimal control methods from software components of an automation software structure that implement different control strategies; and

(ii) calculation of optimal command signals for the selected control strategies.

1.2. Requirements

This section describes the requirements for an optimal command signal configuration solution. The requirements for the solution are divided into the following three classes: requirements for an optimal command signal configuration described in Section 1.2.1, requirements for an optimization model described in Section 1.2.2, and requirements for a behavior model described in Section 1.2.3. Not only the requirements for an optimal command signal configuration but also the requirements for an optimization model and the requirements for a behavior model should be considered. This work is about a configuration approach that automates the following two tasks: (1) selecting optimal control methods from software components of an automation software structure implementing different control strategies and (2) calculating optimal command signals for selected control strategies.

The requirements are derived from three common application scenarios in discrete manufacturing, shown in Figure 2.

Figure 2 Overview of application scenarios: transport scenario, storage scenario, and pick-and-place scenario (adapted from [OVN16, OVN18b, OVN18a]).

The transport scenario describes a process to transport a product between locations. For example, transporting a product between different production modules or between production modules and storage solutions. The storage scenario describes a process to stock products, e.g., produced products that need to be stored temporarily. The pick-and-place scenario describes a process to pick and place products, e.g., pick a product from a conveyor system and place it in a production module.

1.2.1. Requirements for an Optimal Command Signal Configuration

This section describes six requirements R1 - R6 for an optimal command signal configuration.

R1 - Manual engineering steps: Requirement R1 states that an optimal command signal configuration solution should have no manual engineering steps because the vision of cyber-physical production systems is to adapt to new production goals without extensive manual engineering effort. Manual engineering effort is hard to quantify, so only the engineering steps required to create a configuration are considered.

R2 - Global optimum: Requirement R2 states that an optimal command signal configuration solution should find a global optimal command signal configuration for each software component, not just a local optimal command signal configuration. For example, an optimal command signal configuration for an automation software that leads to minimizing energy consumption within a given target time.

R3 - Implicit timing: Requirement R3 states that an optimal command signal configuration solution should specifically consider timing signals, as they are necessary to find an optimal command signal configuration within a given target time.

R4 - Sequences of control methods: Requirement R4 states that an optimal command signal configuration should use behavior knowledge to find an optimal configuration, as they encode the activities necessary to meet a given production goal.

R5 - Parallel sequences of control methods: Requirement R5 states that an optimal command signal configuration should incorporate knowledge of parallel sequences of control methods since production systems can perform multiple parallel activities.

R6 - Selection of different control methods: Requirement R6 states that an optimal command signal configuration should accommodate different control methods for a given activity, increasing the reusability of software components. For example, a robot may use a simple control strategy for simple movements and a model-predictive control approach for complex movements.

Table 1 summarizes the six requirements R1 - R6 and the corresponding scenarios. The reduction of manual engineering steps applies to all three scenarios because the production systems are cyber-physical production systems. Therefore, an optimal command signal configuration should use behavior knowledge in sequences of control methods. Only the pick-and-place scenario has parallel behavior, and only the storage and pick-and-place scenarios have activities with more than one control method, so selecting optimal control methods from software components of an automation software structure that implement different control strategies is necessary.

ID	Transport	Storage	Pick-and-place
R1	$+$	$+$	$+$
R2	$+$	$+$	$+$
R3	$+$	$+$	$+$
R4	$+$	$+$	$+$
R5	$-$	$-$	$+$
R6	$-$	$+$	$+$

Table 1 Overview of requirements for an optimal command signal configuration and the corresponding scenarios ($+$ applicable and $-$ not applicable).

1.2.2. Requirements for an Optimization Model

This section describes six requirements M1 - M6 for an optimization model. In this work, command signal configuration is defined as an optimization problem. Optimization modeling can be used to solve real-world problems [Sch04]. These real-world problems can be used to build optimization models, which can then be used in combination with observed data, such as energy consumption of production systems, to compute solutions, e.g., optimal command signal configurations, using suitable algorithms. In general, an optimization model consists either of a concave, non-concave, convex, non-convex, linear, or non-linear objective function [BV04]. Unlike optimization problems, constrained optimization problems have additional linear or nonlinear constraints on the variables of the objective function. Constrained optimization problems are the focus of this work. Another classification criterion is the use of only discrete, only continuous, or a mixture of continuous and discrete variables [Rot11].

An optimization model should fulfill the following requirements:

M1 - Linear objective function: Requirement M1 states that an optimization model must be able to describe a linear objective function.

M2 - Nonlinear objective function: Requirement M2 states that an optimization model must be able to describe a nonlinear objective function.

M3 - Linear constraints: Requirement M3 states that an optimization model must be able to describe linear constraints.

M4 - Nonlinear constraints: Requirement M4 states that an optimization model must be able to describe nonlinear constraints.

M5 - Continuous variables: Requirement M4 states that an optimization model must be able to describe continuous variables.

M6 - Discrete variables: Requirement M6 states that an optimization model must be able to describe discrete variables.

Table 2 summarizes the six requirements M1 - M6 for an optimization model and the corresponding scenarios. Each scenario has a nonlinear objective function because, in this work, the cost values are energy consumption values. Nonlinear constraints are not required, and only linear constraints are necessary to describe an optimization model for each scenario. Continuous variables are used to describe energy consumption, and discrete variables are used to describe the selection of control methods. Only the storage and pick-and-place scenarios have activities with more than one control method, so discrete variables are not required for the transport scenario.

ID	Transport	Storage	Pick-and-place
M1	−	−	−
M2	+	+	+
M3	+	+	+
M4	−	−	−
M5	+	+	+
M6	−	+	+

Table 2 Requirements M1 - M6 for an optimization model and the corresponding scenarios (+ applicable and − not applicable).

1.2.3. Requirements for a Behavior Model

This section describes four requirements B1 - B4 for behavior models. An optimal command signal configuration should use behavior knowledge to find an optimal command signal configuration because they encode the activities required to meet a specific production goal. The following requirements are defined to compare existing behavior models for technical systems such as cyber-physical production systems:

B1 - Time behavior: Requirement B1 specifies whether timing is part of the underlying behavior model. Section 1.2.1 describes that an optimal command signal configuration solution should consider timing signals in particular since they are necessary to find an optimal control configuration within a given target time. For this purpose, timing can be used to find optimal command signal configurations.

B2 - Continuous behavior: Requirement B2 specifies whether continuous behavior is a part of the underlying behavior model. Section 2.2.1 describes two types of observations: (1) time and (2) energy consumption. These observations are used to learn cost models for each control strategy represented by a control method. For this purpose, a continuous behavior can be used to create cost models.

B3 - Cost behavior: Requirement B3 specifies whether cost behavior is part of the underlying behavior model. Section 2.2 describes that without the possibility of feedback from a production system, e.g., in the form of energy consumption observations or time observations, the selection of optimal control strategies from software components of an automation software structure and the calculation of optimal command signals for selected control strategies is not possible. For this purpose, cost behavior can be used to compare control strategies and command signal configurations.

B4 - Parallel behavior: Requirement B4 specifies whether parallel behavior is part of the underlying behavior model. Section 1.2 describes that an optimal command signal configuration should be able to incorporate knowledge about parallel sequences of control methods since production systems have many parallel activities. For this purpose, parallel behavior can be used to account for knowledge about sequences of control methods.

Table 3 summarizes the four requirements B1 - B4 for a behavior model and their corresponding scenario. The behavior model for each scenario should describe the timing and cost behavior for each control method since the scope of this work is a command signal configuration approach. This approach automates the following two tasks: (1) selecting optimal control methods from software components of an automation software structure that implement different control strategies, and (2) calculating optimal command signals for selected control strategies. For this goal, no additional description of continuous behavior is required. Only the pick-and-place scenario involves parallel behavior, so the behavior model should be able to describe it.

ID	Transport	Storage	Pick-and-place
B1	$+$	$+$	$+$
B2	$-$	$-$	$-$
B3	$+$	$+$	$+$
B4	$-$	$-$	$+$

Table 3 Requirements B1 - B4 for a behavior model and the corresponding scenarios ($+$ applicable and $-$ not applicable).

1.3. Structure of Work

Chapter 2 describes the limitations of the research scope. The research scope constraints are divided into the following five parts: parameters classification, component level observations, control level observations, optimization process, and development process. Chapter 3 analyzes the state of the art. The analysis of state of the art is divided into three sections: existing configuration approaches, optimization models, and behavior models. The existing configuration approaches are generalized into classes. Based on the defined requirements, it is analyzed which configuration class, optimization model, and behavior model can be used to realize a new command signal configuration approach. Chapter 4 formally describes the command signal configuration problem, which must be solved by a configuration approach to automate the task of manual configuration. The notations of the concepts automation software structure, command signals, decision parameters, command signal constraints, cost models, sequences of control methods, optimal command signal configuration, and the task to find an optimal command signal configuration are described. An automation software structure describes components with one or more control methods that implement control strategies. Command signals are used to increase the reusability of automation software components by adding degrees of freedom to control methods that implement control strategies. Command signal constraints restrict command signals and describe valid ranges of values for command signal configurations. Cost values are energy consumption values in this work. Cost models are required to compare control strategies and command signal configurations. A cost model is a mathematical description of measured cost values that allows predicting the expected cost for a command signal configuration.

Chapter 5 introduces the CyberOpt framework. The CyberOpt framework is a formal frame-

work for command signal configuration and describes tasks necessary to find an optimal command signal configuration. A task description is independent of a solution technique. The framework consists of six tasks: Task 1: find a valid command signal configuration, Task 2: record new data from the production system, Task 3: learn cost models, Task 4: learn sequences of control methods, Task 5: find an optimal command signal configuration, and Task 6: improve cost models. Chapter 6 describes the CyberOpt algorithm, a novel approach to machine learning and constrained optimization of command signal configuration. The CyberOpt algorithm automates the following two tasks: (1) selecting optimal control methods from software components of automation software that implement different control strategies, and (2) calculating optimal command signals for selected control strategies. CyberOpt uses the following four machine learning techniques: (1) a machine learning technique of regression and polynomial expansion to learn cost models from observations of the production system, (2) the expected improvement criterion to calculate new valid command signal configurations that should be evaluated to obtain more observations, (3) a machine learning technique named process mining to learn sequences of control methods from production system observations, and (4) mixed-integer nonlinear programming to solve the command signal configuration problem. The CyberOpt algorithm consists of the following five subalgorithms: an algorithm named CyberOpt-SIC that realizes Task 1, an algorithm named CyberOpt-LCM that realizes Task 3, an algorithm named CyberOpt-LSC that realizes Task 4, an algorithm named CyberOpt-SPC that realizes Task 5, and an algorithm named CyberOpt-ICM that realizes Task 6. Chapter 7 describes application scenarios from discrete manufacturing. The automation software structure, command signals, command signal constraints, and control method sequences are described for each scenario. Chapter 8 evaluates the CyberOpt algorithm. The following five experiments are defined: (1) ground truth, (2) random walk, (3) black-box optimization, (4) CyberOpt approach, and (5) CyberOpt MEM. The results of the ground truth experiment are used as reference results. The random walk experiment attempts to find an optimal command signal configuration with random sampling. The black-box optimization attempts to find an optimal command signal configuration. The CyberOpt and CyberOpt MEM experiments attempt to find an optimal command signal configuration using the CyberOpt algorithm. The memory operation mode MEM uses all energy consumption values from previous calculations. Chapter 9 evaluates the CyberOpt subalgorithms. The evaluation aims to prove that the four machine learning techniques used in the CyberOpt algorithm reduce manual engineering effort. Four hypotheses are defined. Hypothesis H1 describes that an optimal command signal configuration solution does not require manual engineering steps. Hypothesis H2 describes that a machine learning algorithm exists that learns cost models that can then be used by a command signal configuration algorithm to find optimal command signal configurations. Hypothesis H3 describes that a machine learning algorithm exists that learns sequences of control methods that can then be used by a command signal configuration algorithm to find optimal command signal configurations. Hypothesis H4 describes that an algorithm exists that finds optimal command signal configurations. Subhypotheses are defined to prove the four hypotheses of this work. Chapter 10 describes the assessment of the introduced requirements for an optimal command signal configuration so-

lution, requirements for an optimization model, and requirements for a behavior model. The CyberOpt algorithm consists of five subalgorithms. In the summary, all requirements for each subalgorithm are assessed. Chapter 11 describes the summary of this work and provides an outlook of this research.

The contents and contributions of this work are based on previous publications of the author, namely [ON15, OVN16, OVN18b, OVN18a]. A brief summary of the contributions and contents of the publications is given in the following list:

[ON15] A thorough search of the literature found that this paper is the first to present a solution to the parameterization problem of adaptable software systems. Due to the nature of cyber-physical production systems, a direct computation of parameters is not possible. Instead, an iteration-based approach is required that uses a model of both the plant and the automation system.

[OVN16] This paper presents a solution for automatically determining the parameters for the automation software of cyber-physical production systems. A scenario from discrete manufacturing illustrates the underlying concepts.

[OVN18b] This paper addresses this key research question and presents a novel approach for parameter estimation approach to select optimal system configurations for cyber-physical production systems automatically. Various scenarios from discrete production systems are used to evaluate the solution approach.

[OVN18a] Models, e.g., of energy consumption, are learned automatically from data observed during production system operation. Therefore, manual engineering effort is minimized, as postulated by the cyber-physical production system paradigm. The presented approach combines MINLP, process mining, and black-box optimization techniques to calculate optimal timing parameter configurations for automation software components with free parameters in the discrete manufacturing domain.

2. Research Scope

This chapter describes the limitations of the research scope. This chapter is based on previous work by the author and extends the published analyses [ON15, OVN16, OVN18b, OVN18a]. The scope of this work is a novel approach to command signal configuration based on machine learning and constrained optimization that automates the following two tasks: (1) selecting optimal control methods from software components of an automation software implementing different control strategies and (2) calculating optimal command signals for selected control strategies. The approach is not a general-purpose solution for all cyber-physical production systems. In this work, only cyber-physical production systems from the discrete manufacturing field are considered. The limitations of the research scope are divided into the following five parts: classification of parameters, component level observations, control level observations, optimization process, and development process.

Section 2.1 classifies the parameters of automation software components, e.g., automation software components developed according to IEC 61131 - 3. Section 2.2 classifies feedback from production systems into component-level observations and control-level observations. Section 2.3 describes the general idea of adapting automation software components with different control strategies and parameters to specific production scenarios through a three-step optimization process. Section 2.4 describes a possible integration of this optimization process into a V-Model XT-based development process.

2.1. Classification of Parameters

Software parameters increase the reusability of automation software components. Automation software parameters add degrees of freedom to automation software components to be configured for a specific production scenario. In this work, automation software parameters are classified into three different classes, shown in Figure 3: decision parameters, timing parameters, and command signals. A decision parameter describes which control strategy of an automation software component should be used to perform a given activity. Only one control strategy can be selected from an automation software component. For example, the automation software component A_3 of the example production system (see Figure 1) implements the activity $move$ with two control strategies. The control strategy $move\text{-}D1(P_{31})$ uses linear acceleration and deceleration ramps and the control strategy $move\text{-}D2(P_{32})$ uses nonlinear acceleration and deceleration ramps. A timing parameter of a control strategy describes the time period within which the control strategy should execute the activity. For example, the conveyor hardware component M_3 can execute the control strategy $move\text{-}D1(P_{31})$ within 10 seconds, 15 seconds, or 20 seconds (see Figure 3: Timing parameters). Command signals specify the degrees of freedom of control strategy implementations. For example, acceleration and deceleration ramp command signals for control strategy $move\text{-}D1(P_{31})$ that

implements activity $move$ (see Figure 3: Command signals). In many cases, there are other command signals in a system that are hidden or not directly accessible to software components. For example, when conveyor systems use frequency converters, the command signals for the acceleration and deceleration ramps are configured separately. In this work, it is assumed that command signals are directly accessible by software components.

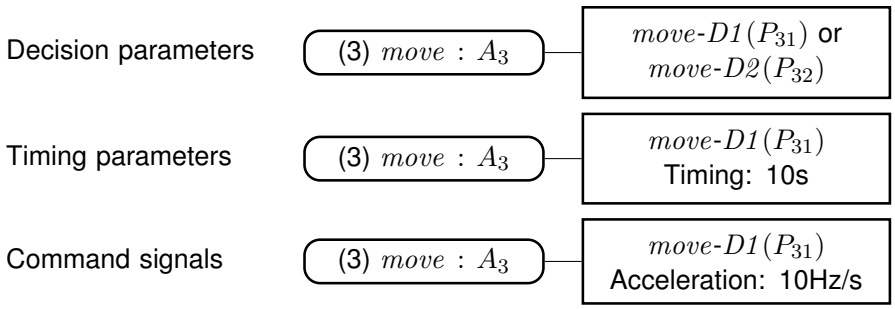


Figure 3 Classification of parameters: decision parameters, timing parameters and command signals (adapted from [OVN18b, OVN18a]).

2.2. Classification of Observations

This section classifies the observations. Without the possibility of feedback from a production system, e.g., in the form of energy consumption observations or time observations, the selection of optimal control strategies from software components of an automation software structure and the calculation of optimal command signals for selected control strategies are not possible. Therefore, feedback in the form of observations of the production system is needed to calculate cost models that can then be used to compare different control strategies and command signal configurations. A cost model is a mathematical description of measured cost values that allows predicting expected costs (e.g., energy consumption). Control strategies implement the behavior of activities in terms of various control methods with parameters.

In general, control strategies can be classified into open-loop and closed-loop control strategies [Oga01], as shown in Figure 4. Open-loop control strategies use no feedback. The control strategy implementation is independent of the output of the controlled system. For example, a washing machine uses an open-loop control strategy. The control implementation is based on a timer, e.g., to use the spin mode for 600 seconds and then use the wash mode for 300 seconds. The control strategy is not to observe, e.g., the cleanliness level of the clothes [Oga01]. In discrete manufacturing, open-loop control strategies are common because fewer sensors are required for a hardware design, or the feedback from a production system is hard to measure or economically not feasible. For example, the filler hardware component M_1 shown in Figure 4 has an open-loop control strategy implementation fill-$B1(P_{11})$. The fill level is controlled by a timer and does not depend on, e.g., a sensor. However, every open-loop system has at least two observable physical quantities: the amount of time and energy consumption. Closed-loop control strategies use feedback. The control strategy implementation depends on the output of the controlled system. For example, a room

temperature control system measures the temperature and compares the measured temperature value with a reference temperature value. Depending on the results, the control strategy heats up or not [Oga01]. In discrete manufacturing, closed-loop control strategies are used if hardware components need to be controlled, e.g., with high-precision motions. For example, the robot hardware component M_2 shown in Figure 4 uses a closed-loop control strategy implementation $pick\text{-}C1(P_{21})$. In many cases, closed-loop control strategies are black boxes where command signals cannot be changed, nor observations can be read, e.g., for safety reasons. Incorrect command signals from robots can cause damage. However, these systems also have observable physical quantities such as time and energy consumption, and in many cases, high-level command signals such as positions and velocities.

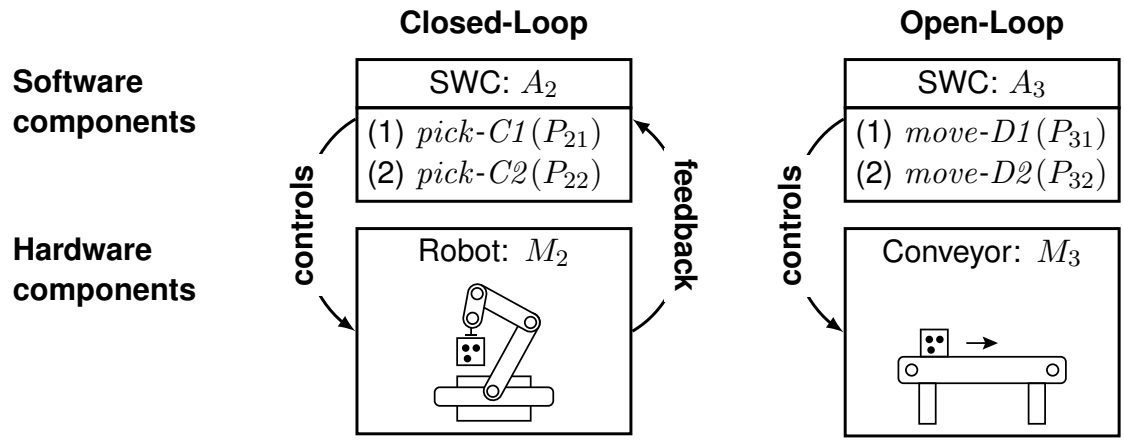


Figure 4 Classification of control strategies: closed-loop control strategies with feedback (left) and open-loop control strategies without feedback (right) (adapted from [OVN18b, OVN18a]).

2.2.1. Component Level Observations

Since the amount of time and energy consumed are the only universally observable quantities, no distinction is made between open-loop and closed-loop control strategies. Control strategies are encapsulated as control methods with command signals and observations are classified into two classes, shown in Figure 5: time quantity and energy consumption quantity.

Observation 1 (Time quantity): How much time does a control strategy take to perform an activity? For example, the robot hardware component M_2 takes 10 seconds to execute the control method $pick\text{-}C1(P_{21})$ and 20 seconds to execute the control method $pick\text{-}C2(P_{22})$.

Observation 2 (Energy consumption quantity): How much energy does a control strategy require to perform an activity? For example, the robot hardware component M_2 requires 800 Joules to execute the control method $pick\text{-}C1(P_{21})$ and 1600 Joules to execute the control method $pick\text{-}C2(P_{22})$.

In this work, the two types of observations described above are used to learn cost models (energy consumption) for each control strategy represented by a control method with command signals. This step is described by a task description with inputs and outputs, called

"learn cost models". The inputs are time and energy consumption values, and the outputs are cost models. The task to learn cost models is formalized in Section 5.3.

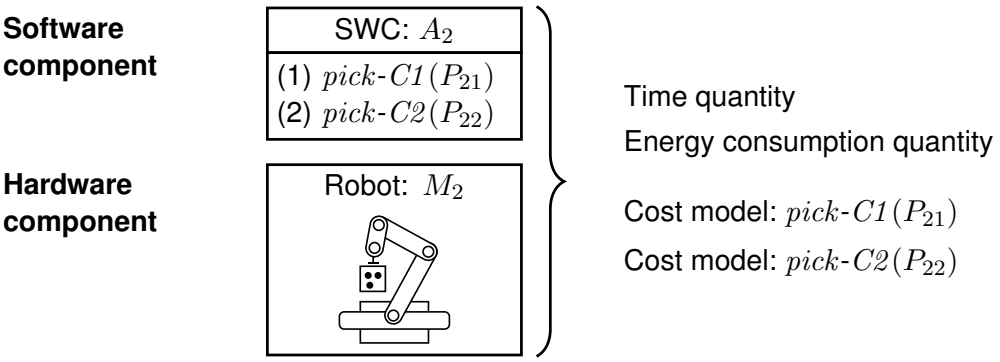


Figure 5 Classification of production system observations: time quantity and energy consumption quantity (adapted from [OVN18b, OVN18a]).

2.2.2. Control Level Observations

In this work, a third observation class is defined, "sequences of control methods". The production goal is described by a behavior model. Behavior models encode sequences of activities, illustrated in Figure 6, e.g., $fill$, $pick$, and $move$.

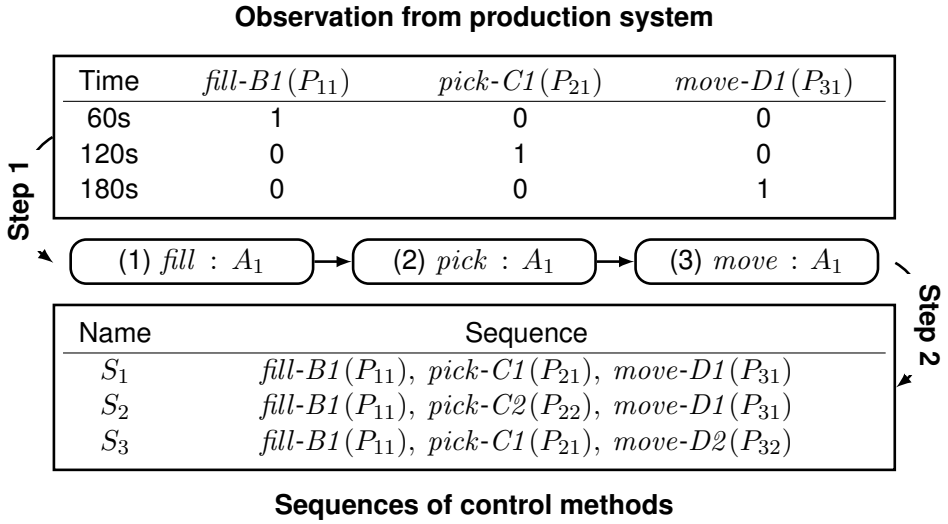


Figure 6 Classification of production system observations: sequences of control methods (adapted from [OVN18a]).

During the operation of a production system, the sequences of control method calls are observable. For example, the following sequence of control methods can be observed in the example production system shown in Figure 6: Step 1: $fill$-$B1(P_{11})$, $pick$-$C1(P_{21})$, and $move$-$D1(P_{31})$. A behavior model can be learned from these control method call sequences. The knowledge from this behavior model combined with an automation software structure can then be used to calculate all possible sequences of control methods. This step is described by a task with inputs and outputs, called "learn sequences of control methods". The inputs to this task are sequences of control method calls, and the outputs are all possible sequences of control methods, shown in Figure 6: Step 2. The task of learning sequences of control methods is formalized in Section 5.4.

2.3. Optimization Process

This section describes the idea of adapting automation software components with different control strategies and command signals to specific production scenarios through a three-step optimization process. The three-step optimization process consists of three decision stages: selecting the optimal sequence of control methods, calculating the optimal timing parameters, and calculating the optimal command signals. Based on the knowledge about the production goal in the form of a behavioral model that encodes sequences of activities, e.g., fill, $pick$ and $move$, knowledge about the automation software structure $A_1 - A_3$, and cost models for each control method, the following three decisions must be made:

Decision 1 (Optimal sequence of control methods): An optimal control strategy must be selected for each software component. For this purpose, cost models from production system observations are used. For example, should the control strategy $pick\text{-}C1(P_{21})$ or the control strategy $pick\text{-}C2(P_{22})$ be used for the activity $pick$.

Decision 2 (Optimal timing parameters): The optimal timing parameters must be calculated for each selected control strategy. For example, should the control strategy $pick\text{-}C2(P_{22})$ execute the activity $pick$ within 10 seconds, 15 seconds or 20 seconds.

Decision 3 (Optimal command signals): The optimal command signals must be calculated for each selected control strategy. For example, acceleration and deceleration command signals for control method $\mathit{fill}\text{-}B1(P_{11})$ or control method $move\text{-}D1(P_{31})$.

The three-step optimization process is described by a command signal configuration problem that is formally defined in Section 4.8.

2.4. Development Process

Technical behavior models are mostly used in combination with development processes and assist engineers in developing technical systems. In general, the specification and implementation of a cyber-physical production system can be carried out using the V-Model XT-based development process [Vog+15], shown in Figure 7.

The V-Model XT-based development process defines (1) project-independent and (2) project-related activities. The results of the project-independent activities are partial solutions, such as automation software components with parameters that are stored in a solution repository. In order to reduce costs and shorten project duration, partial solutions can be reused to create, for example, a system design. If a cyber-physical production system is specified with a V-Model XT-based development process, then the automation software structure, command signals, and decision parameters can be described in the activity of the project-independent

“solution element implementation”. Cost models for each control method can be created in the project-independent “solution test” activity. The sequences of control methods can be calculated from an activity diagram created in the project-related “system design” activity. An optimal command signal configuration can be calculated in the activity of the project-related “system integration”.

Model-based development for automation software is proposed by recent research [Hei+13, OBV15, Vog+15] and increases the quality and effectiveness of embedded software [Lie+14]. First, structural and behavioral models are created with engineering tools [OBV15, Hei+13, HB13], then generators translate these models into a specific source code in a forward engineering step, i.e. IEC 61131 - 3 [Tik+14] for automation software. For example, the modular automation for reuse in manufacturing systems approach [OBV15] focuses on central programmable logic controller systems to support model-driven engineering of object-oriented manufacturing automation software. The notations are based on the Unified Modeling Language (UML) [Ric+10] and the Systems Modeling Language [DeT+13] with the intent of improving the representation of the relationships between structure and behavior. The behavior model is based on the state diagram and the activity diagram of UML. Also, the MechatronicUML approach [Hei+13] integrates software and control engineering with reconfiguration support [HB13]. The main concept of MechatronicUML is the specification of structure and behavior, and the modeling language is a refinement of UML.

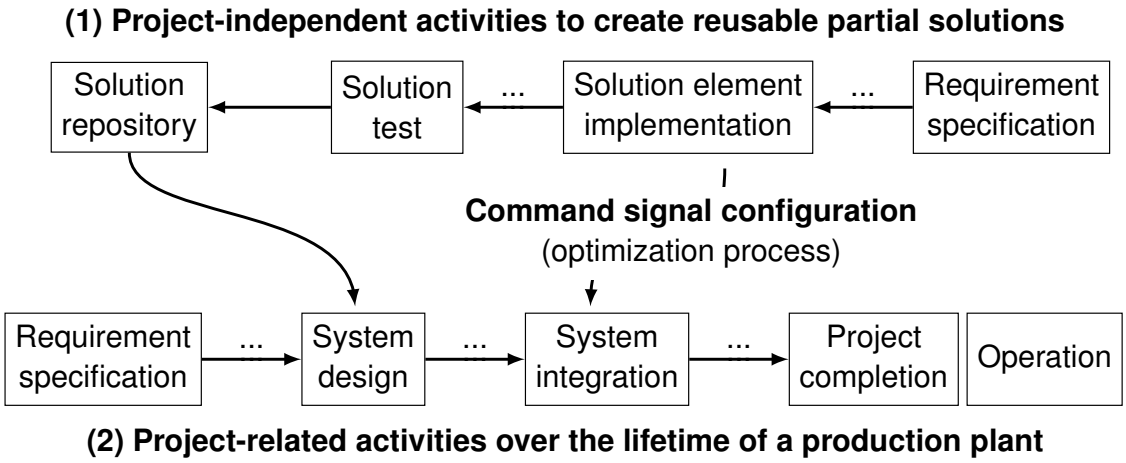


Figure 7 V-Model XT-based development process for cyber-physical production systems (adapted from [Vog+15, OVN16]).

3. State of the Art

This chapter summarizes the state of the art analysis in the field of parameter configuration. This chapter is based on previous work by the author and extends the published analyses [ON15, OVN16, OVN18b, OVN18a]. The chapter is divided into three main concepts: existing approaches to parameter configuration, existing optimization models and existing behavior models. The existing approaches to parameter configuration are arranged as classes. Based on the requirements that are described in Section 1.2, it is analyzed which class of parameter configuration approaches, which optimization model, and which behavior model can be used to automate the manual parameter configuration for automation software components in the domain of discrete manufacturing. Requirements R1 - R6 for an optimal parameter configuration, which are described in Section 1.2.1, are used to compare the following parameter configuration classes: manual parameter configuration, simulation-based parameter configuration, data-driven parameter configuration, and hybrid parameter configuration. Requirements M1 - M6 for an optimization model, which are described in Section 1.2.2, are used to compare the following optimization models: linear programming, nonlinear programming, mixed-integer linear programming, mixed-integer nonlinear programming, knapsack problem, and multiple-choice knapsack problem. Requirements B1 - B4 for a behavior model, which are described in Section 1.2.3, are used to compare the following concepts: timed automata, hybrid timed automata, priced timed automata, activity diagram, and petri net.

Section 3.1 describes the comparison of existing parameter configuration approaches, Section 3.2 describes the comparison of existing optimization models, Section 3.3 describes the comparison of existing behavior models and Section 3.4 describes the conclusions from the state of the art.

3.1. Parameter Configuration Approaches

This section classifies and compares existing parameter configuration approaches. Parameter configuration approaches can be classified into the following four classes: manual parameter configuration, simulation-based parameter configuration, data-driven parameter configuration and hybrid parameter configuration. Manual parameter configuration is performed by a domain expert who specifies parameter configurations for software components of an automation software structure based on their domain knowledge. Simulation-based parameter configuration uses manually predefined optimization models and optimization algorithms to find optimal parameter configurations. Data-driven parameter configuration configures parameters during the operation of the production system based on feedback. Hybrid parameter configuration is a combination of simulation-based and data-driven parameter configuration approaches.

Section 3.1.1 describes the manual parameter configuration class, Section 3.1.2 describes the simulation-based parameter configuration class, Section 3.1.3 describes the data-driven parameter configuration class, and Section 3.1.4 describes the hybrid parameter configuration class.

3.1.1. Manual Parameter Configuration

This section describes the manual parameter configuration class. Manual parameter configuration means that a domain expert tries to find parameter configurations for software components of an automation software structure based on his domain knowledge [ON15, OVN16, OVN18b, Zou+18]. Manual parameter configuration performed by a domain expert can be described by the following four steps: (1) a domain expert selects a plausible parameter configuration according to his domain knowledge, (2) the automation software is set up with the selected parameter configuration, (3) costs are measured during the operation of the production system, e.g., the overall energy consumption, and (4) the domain expert evaluates the costs and changes the parameter configuration based on his domain knowledge. Parameters are changed until the domain expert thinks an optimal parameter configuration is found. Manual parameter configuration has the highest manual engineering effort (R1). There is no evidence that manual parameter configuration finds a global optimal parameter configuration (R2). Timing parameters are likely to be considered by a domain expert (R3). Sequences of control methods (R4), parallel sequences of control methods (R5), and different control methods (R6) are probably not considered.

3.1.2. Simulation-Based Parameter Configuration

This section describes the simulation-based parameter configuration class. Simulation-based parameter configuration is performed by a simulation model and an optimization algorithm and can be described by the following four steps: (1) an optimization algorithm calculates a valid parameter configuration, (2) the automation software is set up with the calculated parameter configuration, (3) during operation of the simulation model, costs are measured, e.g., the overall energy consumption, and (4) an optimization algorithm evaluates the costs and calculates a new parameter configuration. Parameters are changed until a predefined solution quality threshold value or time limit value is reached. Simulation-based parameter configuration approaches use optimization modeling [Sch04, Zou+18] to build simulation models of production systems. Important for the use of optimization modeling is the computability of models [Rot11]. The computability depends on the level of detail modeled by real production systems. In cases of costly evaluations, e.g., physical simulations, black-box optimization approaches can be used [HHL13, CN14, ON15]. The objective function is unknown, and an evaluation of the simulation model is performed for each parameter configuration. Model-based black-box optimization approaches construct a regression model, also called a response surface model or surrogate model, that predicts the costs. Then, this model is used for optimization. Sequential model-based optimization, described in [HHL11], iterates between fitting a model and gathering additional data. It uses Gaussian process models [Ras06] to build a response surface model that predicts which data to evaluate. Gaussian

processes are a generic supervised learning method used, for example, to solve regression problems. Parameter configurations that should be evaluated are calculated using an expected improvement criterion-based on [JSW98] to enhance the quality of surrogate models. In [Ana+14], the authors developed an approach to building holistic system designs to identify optimization potential for mechatronic systems. An approach for calculating optimal schedules for production systems is introduced in [Ked+15]. The production system is described by production resources, capabilities, and the material flow between production resources. Workflows consist of production steps to which capabilities are assigned and translated into sequences of actions that are translated into timed automata. In [Höl+12], a hierarchical multiobjective optimization of drive modules of a rail-bound transportation system is introduced. The drive modules are divided into different modules with suitable optimization models that are arranged hierarchically. The solution optimizes from the bottom up.
Simulation-based parameter configuration involves a high manual engineering effort (R1) because simulation models must be created manually from data, and expert knowledge is required to create them. Furthermore, it is impossible to simulate the real world entirely for computability reasons. Simulation-based parameter configuration is capable of finding a global optimum (R2), but the simulation model may be inaccurate. The class is not able to model implicit timing parameters (R3), does not use explicit sequences of control methods (R4), does not use parallel sequences of control methods (R5), and is not explicitly able to select different control methods (R6).

3.1.3. Data-Driven Parameter Configuration

This section describes the data-driven parameter configuration class. In data-driven parameter configuration approaches, parameters are configured during the operation of the production system based on feedback. Data-driven parameter configuration based on feedback can be described by the following four steps: (1) an algorithm calculates a valid parameter configuration based on the current feedback from the production system in the form of observations, e.g., energy consumption, (2) the automation software is set up with the calculated parameter configuration, (3) feedback is measured at runtime of the production system, and (4) an algorithm directly adjusts the parameter configuration. In [RLL10], an approach to calculating concurrent behavior of a production system from observed data is introduced. The method is based on a formalism called non-deterministic autonomous automaton with output and uses simulated annealing to divide the system into subsystems. An optimization for laser welding at runtime is introduced in [Gün+14]. The solution is able to observe the laser welding process, extract knowledge, and find a strategy to control and optimize it. High dimensional sensor data are used to improve a laser welding system using neural networks and reinforcement learning. The neural networks reduce the dimensionality of the sensor data, and reinforcement learning is used to control the process. Data-driven parameter configuration involves minimal manual engineering effort (R1). However, the data-driven parameter configuration is not a generic solution and is mostly specific for one use case, e.g., laser welding. Data-driven parameter configuration can find a global optimum (R2) and model implicit timing parameters (R3). The class does not use explicit sequences of control methods (R4), parallel

sequences of control methods (R5), or different control methods (R6).

3.1.4. Hybrid-Based Parameter Configuration

Hybrid-based parameter configuration approaches combine simulation-based and data-driven parameter configuration approaches and use data and internal models. Hybrid-based parameter configuration approaches are based on feedback and internal simulation models. They can be described by the following four steps: (1) an algorithm calculates a valid parameter configuration based on the current feedback of the production system in the form of observations, e.g., energy consumption, (2) the automation software is set up with the calculated parameter configuration based on the internal simulation models and the current feedback, (3) feedback is measured at runtime of the production system, and (4) an algorithm directly adjusts the parameter configuration. Model predictive control approaches can be used to calculate parameter configurations at runtime, e.g., coke furnaces [ZXG14] and high-performance control tasks such as electric drives [WB10, Rod+13, SMG13]. In [WNS15], an approach to find optimal sequences of movement parameters is introduced. The approach minimizes energy consumption by adjusting the movement of electric drives. The idea of using energy as a planning resource is introduced in [Vog+14] as model-based energy management of manufacturing plants in real-time. The approach consists of the following steps: creating a simulation model of the production system, synchronization with the real production system, simulation of the parameterized simulation model, and optimization of the energetic behavior based on simulation results. Hybrid-based parameter configuration is a compromise between model creation effort and optimization abilities. The class combines simulation models with observed data from the runtime of production systems and has no manual engineering effort (R1), is able to find a global optimum (R2), and can model implicit timing parameters (R3). The class does not use explicit sequences of control methods (R4), parallel sequences of control methods (R5), or different control methods (R6).

3.1.5. Comparison of Parameter Configuration Approaches

Table 4 describes the comparison of parameter configuration classes and the CyberOpt approach, based on the six requirements R1 - R6 for an optimal parameter configuration approach, described in Section 1.2.1. The manual parameter configuration has the highest manual engineering effort (R1) and can be considered the upper limit, whereas data-driven parameter configuration is the lower limit without any manual engineering effort. However, data-driven parameter configuration approaches are not generic solutions and are mostly specific for one use case, e.g., laser welding. Simulation-based parameter configuration has high manual engineering effort because simulation models must be manually created from data, and expert knowledge is required to create them. Furthermore, it is impossible to simulate the real world entirely for computability reasons. Hybrid parameter configuration is a compromise between the model-building effort and the optimization abilities. They combine simulation models with observed data from the runtime of production systems. Except for manual parameter configuration, the simulation-based, data-driven, and hybrid-based approaches are able to find a global optimum (R2) and model implicit timing parameters (R3).

None of the approaches use explicit sequences of control methods (R4) or parallel sequences of control methods (R5) or are able to handle different control methods (R6). CyberOpt can be classified as a data-driven and hybrid command signal configuration solution. Observed data from production system runtime and machine learning techniques are used to reduce manual engineering effort (R1). Cost models and sequences of control methods are learned, and cost models are improved. The learned cost models with implicit timing parameters (R3), the learned behavior knowledge in the form of sequences of control methods R4 - R5, and the knowledge of the software structure (R6) can then be used to calculate a global optimal command signal configuration (R2) for a production system by solving a command signal configuration problem.

Parameter Configuration Class	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**
Manual parameter configuration	$-$	0	0	0	0	0
Simulation-based parameter configuration	$-$	$+$	$+$	0	0	0
Data-driven parameter configuration	$+$	$+$	$+$	0	0	0
Hybrid parameter configuration	$+$	$+$	$+$	0	0	0
CyberOpt approach	$+$	$+$	$+$	$+$	$+$	$+$

Table 4 Overview of different parameter configuration classes, the CyberOpt approach and requirements: ($+$ fulfilled, 0 not specified and $-$ not fulfilled).

3.2. Optimization Models

This section describes a classification of optimization models, solving techniques, and mathematical solvers. The idea is to automatically calculate an optimal command signal configuration that can then be used to set up a production system. In this work, command signal configuration is defined as an optimization problem. The command signal configuration problem described in Section 4.8 must be solved to find optimal configurations. This section examines existing optimization problem classes: linear programming, nonlinear programming, mixed-integer linear programming, mixed-integer nonlinear programming, knapsack problem, and multiple-choice knapsack problem.

Optimization modeling can be used to solve real-world problems [Sch04]. Optimization models can be created from these real-world problems, which can then be used in combination with observed data, such as energy consumption from production systems, to compute solutions, e.g., optimal command signal configurations, using suitable algorithms. An optimization model generally consists either of a concave, non-concave, convex, nonconvex, linear, or nonlinear objective function [BV04]. Unlike optimization problems, constrained optimization problems have additional linear or nonlinear constraints on variables of the objective function. Constrained optimization problems are the focus of this work. Another classification criterion is the use of only discrete, only continuous, or a mixture of continuous and discrete variables [Rot11].

Important for the usage of optimization modeling is the concept of computability of models [Rot11]. Computability depends on the model formalism used. Computational complexity theory allows statements about the computability of optimization problems. The complexity classes P, NP, NP-complete, and NP-hard are sets of problems with the same asymptotic behavior of time and space required to solve them [Rot11]. The set of class P describes problems that are easily solved by algorithms in polynomial time. The set of class NP-hard describes problems that are hard to solve because no polynomial-time algorithm is known, and it is impossible to verify whether a solution is a feasible solution in polynomial time. The set of class NP-complete describes problems in the set of class NP-hard and the set of class NP. This means that these problems are difficult to solve because no polynomial-time algorithm is known, but it is possible to check whether a solution is feasible in polynomial time.

Section 3.2.1 describes the classification of optimization problems, Section 3.2.2 describes various solving techniques for solving convex and nonconvex mixed-integer nonlinear programming problems, Section 3.2.3 describes solvers for mixed-integer nonlinear programming problems, and Section 3.2.4 describes the comparison of optimization models.

3.2.1. Classification of Optimization Problems

This section describes the classification of optimization problems. The following list of optimization problems are classified: (1) *mixed-integer nonlinear programming*, (2) *mixed-integer linear programming*, (3) *nonlinear programming*, (4) *linear programming*, (5) *knapsack problem*, and (6) *multiple-choice knapsack problem*.

The notation of the mixed-integer nonlinear programming (MINLP) [LL12] concept consists of a linear objective function or a nonlinear objective function (M1 and M2), linear or nonlinear constraints (M3 and M4), and continuous and discrete variables (M5 and M6). MINLP problems belong to the NP-hard complexity class [BL12a]. The formal definition of the MINLP concept is described in Definition 1.

Definition 1 (Mixed Integer Nonlinear Programming):

$$\min\{f^0(x,y) : f^j(x,y) \leq 0 \quad (j = 1,\ldots,m),\ x \in \mathbb{Z}_+^{n_1},\ y \in \mathbb{R}_+^{n_2}\}, \tag{3.1}$$

where n_1 describes the number of integer-constrained variables, n_2 describes the number of continuous variables, m describes the number of constraints, $f^j(x,y)$ are arbitrary function mapping $\mathbb{Z}_+^{n_1} \times \mathbb{R}_+^{n_2}$ to the set of real numbers and $f^0,\ldots,f^n$ are nonlinear functions.

The notation of the mixed-integer linear programming (MILP) [Mar99] concept consists of a linear objective function (M1), linear constraints (M3), and continuous and discrete variables (M5 and M6). MILP problems belong to the NP-hard complexity class [Jün+09]. The formal definition of the MILP concept is described in Definition 2.

Definition 2 (Mixed Integer Linear Programming):

$$\min\{f^0(x,y) : f^j(x,y) \leq 0 \quad (j = 1,\ldots,m),\ x \in \mathbb{Z}_+^{n_1},\ y \in \mathbb{R}_+^{n_2}\}, \tag{3.2}$$

where n_1 describes the number of integer-constrained variables, n_2 describes the number of continuous variables, m describes the number of constraints, $f^j(x,y)$ are arbitrary function mapping $\mathbb{Z}_+^{n_1} \times \mathbb{R}_+^{n_2}$ to the set of real numbers and $f^0,\ldots,f^n$ are linear functions.

The notation of the nonlinear programming (NLP) [LY15] concept consists of a nonlinear objective function (M2), linear constraints or nonlinear constraints (M3 and M4), and continuous variables (M5). NLP problems belong to the NP-hard complexity class [MK87]. The formal definition of the NLP concept is described in Definition 3.

Definition 3 (Nonlinear Programming):

$$\min\{f^0(x) : f^j(x) \leq 0 \quad (j = 1,\ldots,m),\ x \in \mathbb{R}_+^{n_1}\}, \tag{3.3}$$

where n_1 describes the number of continuous variables, m describes the number of constraints, $f^j(x)$ are arbitrary function mapping $\mathbb{R}_+^{n_1}$ to the set of real numbers and $f^0,\ldots,f^n$ are nonlinear functions.

The notation of the linear programming (LP) concept [Dan98] consists of a linear objective function (M1), linear constraints and continuous variables (M3 and M4). LP problems belong to the P complexity class [Kha80]. The formal definition of the LP concept is described in Definition 4.

Definition 4 (Linear Programming):

$$\min\{f^0(x) : f^j(x) \leq 0 \quad (j = 1,\ldots,m),\ x \in \mathbb{R}_+^{n_1}\}, \tag{3.4}$$

where n_1 describes the number of continuous variables, m describes the number of constraints, $f^j(x)$ are arbitrary function mapping $\mathbb{R}_+^{n_1}$ to the set of real numbers and $f^0,\ldots,f^n$ are linear functions.

The knapsack problem (KP) and the multiple-choice knapsack problem (MCKP) are subclasses of optimization problems with discrete variables, where the domain of discrete variables is reduced to the domain of binary variables [MPT00, KPP04]. KP is in the complexity class NP-complete, and MCKP is in the NP-hard complexity class [KPP04]. The knapsack problem (KP) is described in Definition 5.

Definition 5 (Knapsack Problem):

Given profit values $p_i \in \mathbb{R}$ and weight values $w_i \in \mathbb{R}$ the problem is defined as follows:

$$\text{maximize:} \quad \sum_{i=1}^{k} p_i \, b_i \tag{3.5}$$

$$\text{s.t.:} \quad \sum_{i=1}^{k} w_i \, b_i \leq Obj, \tag{3.6}$$

$$b_i \in \{0, 1\}, i = 1, \ldots, k \tag{3.7}$$

A similar problem is the well-studied multiple-choice knapsack problem (MCKP), where the goal is to choose exactly one item j from each of the k classes $N_i, i = 1, \ldots, k$ such that the profit sum is maximized.

Definition 6 (Multiple-Choice Knapsack Problem):

Given profit values $p_{ij} \in \mathbb{R}$ and weight values $w_{ij} \in \mathbb{R}$ the problem is defined as follows:

$$\text{maximize:} \quad \sum_{i=1}^{k} \sum_{j \in N_i} p_{ij} \, b_{ij} \tag{3.8}$$

$$\text{s.t.:} \quad \sum_{i=1}^{k} \sum_{j \in N_i} w_{ij} \, b_{ij} \leq Obj, \tag{3.9}$$

$$\sum_{j \in N_i} b_{ij} = 1, i = 1, \ldots, k, \tag{3.10}$$

$$b_{ij} \in \{0, 1\}, i = 1, \ldots, k, j \in N_i \tag{3.11}$$

Despite the theoretical fact that NLP, MILP, MINLP, and MCKP belong to the NP-hard complexity class, research has been published to implement efficient algorithms to solve these optimization problems. Section 3.2.2 describes various solving techniques for convex and nonconvex MINLP problems.

3.2.2. Solving Techniques for Optimization Problems

The command signal configuration problem described in Section 4.8 is defined as a mixed-integer nonlinear programming (MINLP) problem with a nonlinear objective function, linear constraints, and a mixture of continuous and binary variables. Therefore, this section describes different solving techniques for convex and nonconvex MINLP problems.

If the objective function f^0 and all constraints $f^1, \ldots, f^m$ of an MINLP problem are convex, then the MINLP problem is called convex MINLP problem. Convex MINLP problems can be solved by, e.g., branch-and-bound, outer approximation, and the extended cutting plane

method. If the objective function f^0 or one constraint $f^i, i > 1$ of an MINLP problem is nonconvex, then the MINLP is called nonconvex. In this case, the nonconvex functions must be replaced by underestimating or overestimating functions, or by separable functions, or the problem must be rewritten by factorization [BL12b].

Branch-and-Bound: The branch-and-bound technique is a general approach to solve optimization problems. Branch-and-bound is also known as divide-and-conquer. The general idea is to divide an optimization problem into subproblems until the subproblems are easy to solve. The branch-and-bound technique was introduced for solving MILP problems [LD60]. This technique was later used to solve MINLP problems [Dak65, GR85]. The branch-and-bound technique begins with the relaxation of integer constraints on MINLP problems. An MINLP problem is then reduced to an NLP problem and can be solved by an NLP solver. If the NLP problem is infeasible, then the MINLP problem is also infeasible. If the solution to the NLP problem is an integer, then it also solves the MINLP problem. Otherwise, the branch-and-bound technique creates a search tree. The nodes of the search tree correspond to the NLP subproblems, and edges of the search tree correspond to the branching decisions [Bel+13]. The division of subproblems into smaller subproblems is called branching. Standard branching describes a simple branching rule [Kro+18, Bel+13, BL12b]. If an integer-constrained variable x_i takes a fractional value x_i^*, e.g. $x_i^* = 0.5$, then two subproblems with additional constraints are created: Subproblem 1 with the additional constraint $x_i \leq x_i^*$ and Subproblem 2 with the additional constraint $x_i \geq x_i^*$. Spatial branching describes the partitioning of the domain of continuous variables [BL12b]. For example, consider a continuous variable y_i whose domain is $[l_i, u_i]$, then a value b with $l_i < b < u_i$ is chosen. Two subproblems are created: Subproblem 1 with the domain $[l_i, b]$ and a Subproblem 2 with the domain $[b, u_i]$. Subproblems are removed from the search tree under the following three conditions: (1) the NLP subproblem is feasible, and its optimal solution is accurate within a specified tolerance, (2) the associated lower bound is not better than the best upper bound found so far, or (3) the NLP subproblem is infeasible.

Cutting Planes: Tight continuous relaxation (NLP subproblem) is important to avoid large branch-and-bound search trees. Cutting planes can be used to strengthen continuous relaxation. The general idea is to cut off fractional optimal solutions by adding cutting planes. This should lead to a significant reduction in the size of the search tree [SM99, Bel+13]. The extended cutting plane technique [WP95, WP02] uses a linearization of the nonlinear constraints. More precisely, the technique constructs an iteratively improving polyhedral outer approximation [Kro+18]. There are several cutting-plane approaches to tighten the convex hull of MINLP problems; details are described in [Bel+13]: mixed-Integer rounding cuts [AMR01], perspective cuts [AG06], and disjunctive cutting planes [CS99].

Outer Approximation: The general idea of outer approximation is to create an equivalent linear representation of MINLP problems and apply relaxation. The outer approximation is a decomposition technique [DG86]. The optimal solution of MINLP problems is found by solving

a sequence of MILP and NLP subproblems [FL94]. First, an upper bound is calculated by solving an NLP problem with fixed integer variables. Second, a lower bound is calculated by solving an MILP problem based on the results of the NLP problem. If the upper and lower bounds match, an optimal solution is found. Otherwise, an integer cut is added to the NLP problem, and the NLP problem and the MILP problem are solved again, and so on. The detailed MILP problem and NLP problem are described in [Kro+18].

Primal Heuristics: Primal heuristics should find good feasible solutions with less computational effort than solving the original problem. A good feasible solution avoids large branch-and-bound search trees and provides a tight upper bound [Kro+18]. An example is the undercover primal heuristic [BG14]. The general idea is to explore an MINLP problem by a sub-MIP problem. The sub-MIP problem is created by fixing a minimal set of variables sufficient to linearize all nonlinear constraints. The heuristic is based on the observation that any given MINLP problem can be reduced to a sub-MIP problem by fixing certain variables to a value within their bounds. Any feasible solution to this sub-MIP problem is then a feasible solution to the original MINLP problem.

Preprocessing: Preprocessing improves solving performance. The feasibility-based bound tightening technique and the optimization-based bound tightening technique tighten the bounds of an MINLP problem. The feasibility-based bound tightening technique analyzes constraints sequentially to tighten variable ranges [Bel+10], while the optimization-based bound tightening technique [LM06] solves relaxed problems where each variable is maximized and minimized to tighten variable bounds. A tighter polyhedral outer approximation can be reached by reformulating the original problem. Nonlinear constraints are divided into multiple constraints by introducing new variables [HBO14, Lub+16].

3.2.3. Solver for Mixed Integer Nonlinear Programming Problems

This section describes six different solvers that use the solving techniques described above. Each solver is capable of solving convex and nonconvex MINLP problems.

AlphaECP: The Alpha Extended Cutting Plane approach is an MINLP problem solver. The solver is capable of solving general MINLP problems and uses the extended cutting plane method. The method only requires solving a mixed integer programming subproblem in each iteration. The mixed-integer programming subproblems are solved in intermediate iterations to the optimal solution, feasibility, or just an integer relaxed solution [WP02, PW00, SW01, WP95, Wes+98].

ANTIGONE: The Algorithms for coNTinuous Integer Global Optimization of Nonlinear Equations approach is an MINLP problem solver. The solver is capable of solving convex problems and nonconvex problems. It uses reformulations to decompose nonlinear functions into, e.g., linear, quadratic, and exponential terms. A branch-and-cut approach solves generated convex relaxations of the decomposed nonconvex terms [MF13, MF14, MSF15].

BARON: The Branch-And-Reduce Optimization Navigator approach is an MINLP problem solver. The solver uses a polyhedral branch-and-bound technique that solves LP relaxations in branch-and-bound nodes, MILP relaxations, nonlinear relaxations, convex underestimators, and concave overestimators in combination with a spatial branch-and-bound, automatic reformulations, convexity identification, and decomposition of nonconvex functions into simpler functions with known convex or concave relaxations [RS95, RS96, Sah96, TS05, Zho+17].

BONMIN and Couenne: The Basic Open-source Nonlinear Mixed INteger programming approach is an MINLP problem solver. The solver uses an NLP-based branch-and-bound, a branch-and-cut, an outer approximation decomposition, and a hybrid outer approximation-based branch-and-cut technique [Bon+08]. The Convex Over and Under ENvelopes for Nonlinear Estimation approach is an MINLP problem solver. The solver is capable of solving convex and nonconvex MINLP problems and uses an LP-based spatial branch-and-bound technique, bound reduction, and primal heuristics [Bel+09]. Couenne extends the BONMIN approach to calculate valid linear outer approximations for nonconvex MINLP problems.

DICOPT: The DIscrete Continuous OPTimizer approach is an MINLP problem solver. The solver uses outer approximation and solves a series of NLP problems and MIP problems. The performance is improved by a feasible pump-primal heuristic [Ber+20].

SCIP: The Solving Constraint Integer Programs approach is an MINLP problem solver. The solver uses a polyhedral outer approximation, a spatial branch-and-bound, LP relaxations, constraint programming, cutting planes, and primal heuristics [Ach09, VG17].

3.2.4. Comparison of Optimization Models

Table 5 describes the comparison of optimization problems based on six requirements M1 - M6 for an optimization model described in Section 1.2.2.

Optimization Problem	M1	M2	M3	M4	M5	M6
Mixed-integer nonlinear programming	$+$	$+$	$+$	$+$	$+$	$+$
Mixed-integer linear programming	$+$	$-$	$+$	$-$	$+$	$+$
Nonlinear programming	$+$	$+$	$+$	$+$	$+$	$-$
Linear programming	$+$	$-$	$+$	$-$	$+$	$-$
Knapsack problem	$+$	0	$+$	0	0	$+$
Multiple-choice knapsack problem	$+$	0	$+$	0	0	$+$
Command signal configuration problem	$+$	$+$	$+$	$+$	$+$	$+$

Table 5 Overview of different approaches and requirements: ($+$ fulfilled, 0 not specified, $-$ not fulfilled).

MINLP is capable of modeling a linear objective function or a nonlinear objective function, linear constraints, nonlinear constraints, continuous variables, and discrete variables, thus satisfying requirements M1 - M6. MILP can be used to model a linear case of MINLP. MILP is capable of modeling a linear objective function, linear constraints, continuous variables,

and discrete variables, and thus satisfies requirements M1, M3, M5, and M6. NLP has the same capability as MINLP except for discrete variables. NLP is capable of modeling a linear objective function or a nonlinear objective function, linear constraints, nonlinear constraints, continuous variables, and discrete variables, and thus satisfies M1 - M5. LP is capable of modeling a linear objective function, linear constraints, and continuous variables and thus satisfies M1, M3, and M5. KP and MCKP are capable of modeling a linear objective function, linear constraints, and discrete variables. A nonlinear objective function, nonlinear constraints, and continuous variables are not provided, so they are marked as unspecified. They satisfy requirements M1, M3, and M6. The command signal configuration problem is defined as a mixed-integer nonlinear programming problem with a nonlinear objective function, linear constraints, and a mixture of continuous and binary variables, see Section 4.8. A command signal configuration problem satisfies requirements M1 - M6.

The modeling abilities come at a price. Table 6 summarizes the optimization problems and their corresponding complexity classes. Only LP is in the P complexity class. MINLP, MILP, NLP, MCKP, and command signal configuration problems are in the NP-hard complexity class. KP is in the NP-complete complexity class.

Complexity Class	**MINLP**	**MILP**	**NLP**	**LP**	**KP**	**MCKP**	**CSCP**
NP-hard	$+$	$+$	$+$	$-$	$-$	$+$	$+$
NP-complete	$-$	$-$	$-$	$-$	$+$	$-$	$-$
P	$-$	$-$	$-$	$+$	$-$	$-$	$-$

Table 6 Overview of different optimization problems and complexity classes: ($+$ fulfilled, $-$ not fulfilled, CSCP = Command Signal Configuration Problem).

3.3. Behavior Models

This section describes existing automata-based behavior models and engineering behavior models. An optimal command signal configuration should use behavior knowledge to find optimal command signal configurations because they encode the activities that are necessary to meet a specific production goal of a cyber-physical production system. Behavior models can be classified into sequential and parallel behavior models [Men+03]. The sequential behavior models compared are *timed automata*, *hybrid timed automata*, and *priced timed automata*, and the parallel behavior models compared are *activity diagram* and *petri net*.

Section 3.3.1 describes automata-based behavior models, Section 3.3.2 describes engineering behavior models, and Section 3.3.3 describes the comparison of behavior models.

3.3.1. Automata-Based Behavior Models

This section explains the differences between the concepts of timed automata, hybrid timed automata, and priced timed automata. Recent research has focused on building behavior models. Behavior models can be used to validate, analyze, control, and simulate technical

systems. Each behavior model describes different aspects of a technical system. Timed automata (TA) can model the behavior of real-time systems over time [AD94], e.g., open-loop control strategies that do not use feedback. Hybrid timed automata (HTA) are a generalization of TA and can model continuous behavior of real-time systems over time [Alu+93], e.g., closed-loop control strategies that use feedback. Priced timed automata (PTA) are TA that can model linear costs of real-time systems [Beh+01], e.g., a linear approximation of the energy consumption of open-loop control strategies.

The example of a filling process, shown in Figure 8, is introduced to explain the differences between the concept timed automata, the concept hybrid timed automata, and the concept priced timed automata.

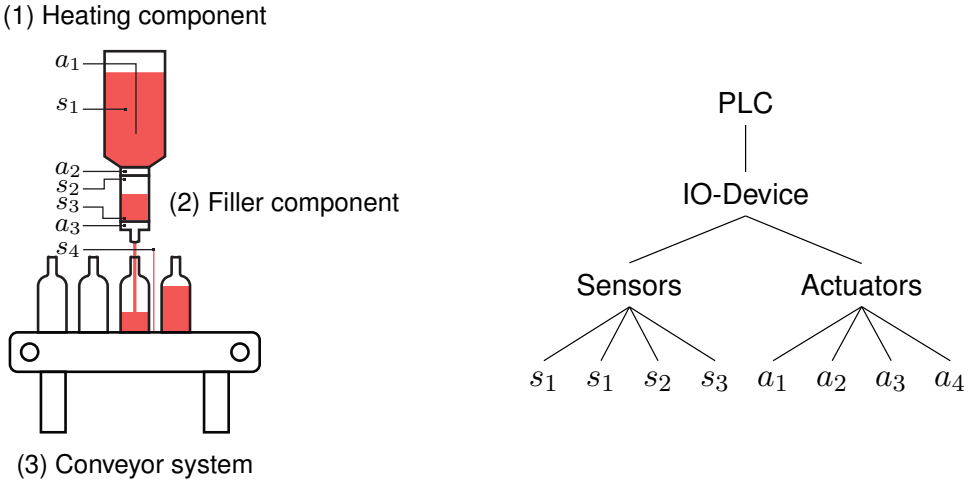


Figure 8 Example of a filling process: actuator a_1 controls the heating component, actuators a_2, a_3 control valve 1 and valve 2 of the filler component, actuator a_4 controls the conveyor system, sensor s_1 measures the temperature of the water, sensors s_2, s_3 are filling level sensors, and sensor s_4 is a light barrier sensor (adapted from [ON15]).

The goal of this filling process is to fill $0.100l$ heated water into a bottle. The water must be heated to $70°C$ degrees Celsius, e.g., to kill bacteria before it can be filled into the bottle. The example filling process is implemented by three automation components: (1) a heating component, (2) a filling component, and (3) a conveying system. The automation components are controlled by four actuators a_1 – a_4 and four sensors s_1 – s_4, which are connected to a programmable logic controller (PLC) via an IO device. Actuator a_1 controls the heating component. If actuator $a_1 = 0$, then the heating component is off. If actuator $a_1 = 1$, then the heating component is on. Sensor s_1 measures the temperature of the water. Actuator a_2 and actuator a_3 control valve 1 and valve 2 of the filling component. Sensor s_2 and sensor s_3 are filling level sensors. Actuator a_4 controls the conveyor system and s_4 is a light barrier sensor.

Timed Automata:

The concept is that timed automata (TA) are able to model the behavior of real-time systems over time, e.g., the timing behavior of cyber-physical production systems (B1). Continuous behavior (B2), cost behavior (B3), and parallel behavior (B4) are not part of the TA concept.

The formal definition of the TA concept is described in Definition 7.

Definition 7 (Timed Automata):

This is a special case of *timed automata* adapted from [Mai+11]. A timed automata $TA = (L, l_0, F, \Sigma, T, \Delta, c)$ is described as a 7-tuple, where

(i) **States:** States are defined as a finite set L. The state $l_0 \in L$ describes the initial state, e.g., the start state of a production system. The finite set $F \subseteq S$ describes the set of final states, e.g., the product is being manufactured, or the production system is in an error state. For example, the states of the example filling process are: heating l_0, measuring l_1, filling l_2, and moving l_3.

(ii) **Events:** The finite set Σ describes the alphabet comprising all events. For example, the alphabet comprising all events of the example filling process is $\Sigma = \{s_1, s_2, s_3, s_4\}$. An event s_1, s_2, s_3, or s_4 is triggered when the sensor valve is one, e.g., $s_1 = 1$.

(iii) **Transitions:** Transitions between states are described by the finite set $T \subseteq S \times \Sigma \times S$. For example, the following list of transitions exists for the sample filling process: between state l_0 and state l_1, between state l_1 and state l_2, between state l_2 and state l_3, and between state l_3 and state l_0.

(iv) **Single clock:** The single clock c is used to record the time. The clock is reset at each transition.

(v) **Timing constraints:** The finite set Δ describes timing constraints with $\delta : T \rightarrow I$, where $\delta \in \Delta$ and I is the set of time intervals. For example, the timing constraint $c \geq 208s$ describes that a transition is only taken after 208 seconds.

Figure 9 illustrates the timed automata concept of the example filling process.

State l_0 (heating): The filling process begins with the heating of the water $a_1 = 1$. Event s_1 is triggered and the heating of the water ends after $208s$. The state is switched from state l_0 to state l_1, the heating element is deactivated $a_1 = 0$, and valve $a_2 = 1$ is open.

State l_1 (measuring): Event s_2 is triggered, after $6s$ the filler is filled with $0.100l$ heated water. The state is switched from state l_1 to state l_2, valve a_2 is closed, and valve a_3 is open.

State l_2 (filling): Event s_3 is triggered, after $6s$ the bottle is filled with heated water. The state is switched from state l_2 to state l_3, valve a_3 is closed, and the conveyor $a_4 = 1$ starts to move the bottle.

State l_3 (moving): Event s_4 is triggered, after $10s$ the bottle is moved. The state is switched from state l_3 to state l_0, the conveyor system is off $a_4 = 0$, the heating element on $a_1 = 1$, and the filling process starts again.

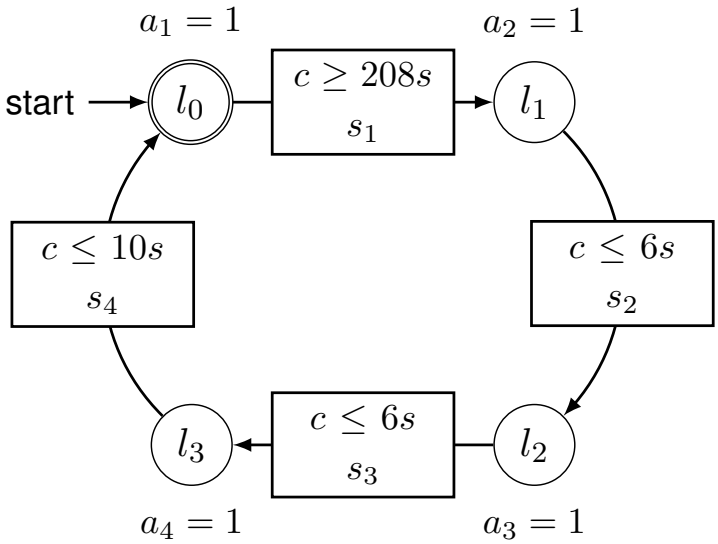


Figure 9 Timed automata of the example filling process: actuator a_1 controls the heating component, actuators a_2, a_3 control valve 1 and valve 2 of the filler component, actuator a_4 controls the conveyor system, sensor s_1 measures the temperature of the water, sensors s_2, s_3 are filling level sensors, and sensor s_4 is a light barrier sensor. Sensor signals $s_1 - s_2$ are events.

Table 7 summarizes the sequences of actuator signals $a_1 - a_4$ and sensor signals $s_1 - s_4$ of the example filling process. The column "Step" describes the iteration of the filling process from state l_0 to state l_0 via the states l_1, l_2, l_3, the column "Time" describes the overall time in seconds, and the column "Constraint" describes the timing constraints of the timed automata.

			Actuators				**Sensors**			
Step	**Time**	**Constraint**	a_1	a_2	a_3	a_4	s_1	s_2	s_3	s_4
1	$0s$	$c \geq 208s$	1	0	0	0	1	0	0	0
1	$208s$	$c \leq 6s$	0	1	0	0	0	1	0	0
1	$304s$	$c \leq 6s$	0	0	1	0	0	0	1	0
1	$310s$	$c \leq 10s$	0	0	0	1	0	0	0	1
2	$518s$	$c \geq 208s$	1	0	0	0	1	0	0	0
2		...								
3	$540s$	$c \geq 208s$	1	0	0	0	1	0	0	0
		...								

Table 7 Timed automata of the example filling process: the column "Step" describes the iteration of the filling process, the column "Time" describes the overall time in seconds, the column "Constraint" describes timing constraints, the columns $a_1 - a_4$ describe sequences of actuator signals, and the columns $s_1 - s_4$ describe sensor signals.

Hybrid Timed Automata:

Hybrid timed automata (HTA) are capable of modeling continuous behavior (B2) over time (B1) of real-time systems. Cost behavior and parallel behavior (B3 and B4) are not part of the hybrid timed automata concept. The formal definition of the HTA concept is described in Definition 8.

Definition 8 (Hybrid Timed Automata):

This is a special case of *hybrid timed automata* adapted from [Nig+12]. $HTA = (L, l_0, F, \Sigma, T, \Delta, c, \Theta)$ is described as an 8-tuple, where

(i) **States:** States are defined as a finite set L. The state $l_0 \in L$ describes the initial state, e.g., the start state of a production system. The finite set $F \subseteq S$ describes the set of final states, e.g., the product is being manufactured, or the production system is in an error state. For example, the states of the example filling process are: heating l_0, measuring l_1, filling l_2, and moving l_3.

(ii) **Events:** The finite set Σ describes the alphabet comprising all events. For example, the alphabet comprising all events of the example filling process is $\Sigma = \{s_1, s_2, s_3, s_4\}$. An event s_1, s_2, s_3 or s_4 is triggered, if the sensor valve is one, e.g., $s_1 = 1$.

(iii) **Transitions:** Transitions between states are described by the finite set $T \subseteq S \times \Sigma \times S$. For example, the following list of transitions exists for the sample filling process: between state l_0 and state l_1, between state l_1 and state l_2, between state l_2 and state l_3, and between state l_3 and state l_0.

(iv) **Single clock:** The single clock c is used to record the time. The clock is reset at each transition.

(v) **Timing constraints:** The finite set Δ describes timing constraints with $\delta : T \rightarrow I$, where $\delta \in \Delta$ and I is the set of time intervals. For example, the timing constraint $c \geq 208s$ describes that a transition occurs only after 208 seconds.

(vi) **Continuous behavior:** Continuous behavior Θ describes a finite set of functions with elements $\theta_l : R^n \rightarrow R^m$, $\forall l \in L, n \in N, m \in N$; i.e. θ_l is the function expressing single valve changes within a single state l.

Figure 10 illustrates the concept of hybrid timed automata of the example filling process.

State l_0 (heating): The filling process begins with the heating of the water $a_1 = 1$. When $70°C$ are reached, then event s_1 is triggered and heating of the water ends after $208s$. The state is switched from state l_0 to state l_1, the heating element is deactivated $a_1 = 0$, and valve $a_2 = 1$ is open.

State l_1 (measuring): Event s_2 is triggered. After $6s$ the filler is filled with $0.100l$ heated water. The state is switched from state l_1 to state l_2, valve a_2 is closed, and valve a_3 is open.

State l_2 (filling): Event s_3 is triggered. After $6s$ the bottle is filled with heated water. The state is switched from state l_2 to state l_3, valve a_3 is closed, and the conveyor $a_4 = 1$ starts to move the bottle.

State l_3 (moving): Event s_4 is triggered. After $10s$ the bottle is moved. The state is switched from state l_3 to state l_0, the conveyor system is off $a_4 = 0$, the heating element is on $a_1 = 1$, and the filling process starts again.

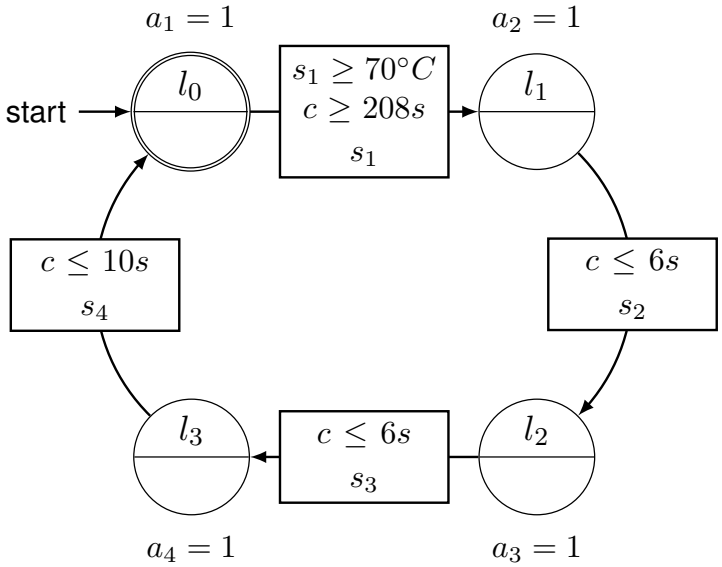


Figure 10 Hybrid timed automata of the example filling process: actuator a_1 controls the heating element, actuators a_2, a_3 control valve 1 and valve 2, actuator a_4 controls the conveyor system, sensor s_1 measures the temperature of the water, sensors s_2, s_3 are filling level sensors, and sensor s_4 is a light barrier sensor. Sensor signals $s_1 - s_2$ are events..

Table 8 summarizes the sequences of actuator signals $a_1 - a_4$ and sensor signals $s_1 - s_4$ of the example filling process. The column "Step" describes the iteration of the filling process from state l_0 to state l_0 via the states l_1, l_2, l_3, the column "Time" describes the overall time in seconds, and the column "Constraint" describes the timing and continuous constraints of the hybrid timed automata.

			Actuators				**Sensors**			
Step	**Time**	**Constraint**	a_1	a_2	a_3	a_4	s_1	s_2	s_3	s_4
1	$0s$	$s_1 \geq 70°C$	1	0	0	0	70	0	0	0
1	$208s$	$c \leq 6s$	0	1	0	0	68	1	0	0
1	$304s$	$c \leq 6s$	0	0	1	0	65	0	1	0
1	$310s$	$c \leq 10s$	0	0	0	1	60	0	0	1
2	$518s$	$s_1 \geq 70°C$	1	0	0	0	70	0	0	0
2		...								
3	$540s$	$s_1 \geq 70°C$	1	0	0	0	70	0	0	0
		...								

Table 8 Hybrid timed automata of the example filling process: the column "Step" describes the iteration of the filling process, the column "Time" describes the overall time in seconds, the column "Constraint" describes timing and continuous constraints, the columns $a_1 - a_4$ describe sequences of actuator signals, and the columns $s_1 - s_4$ describe sensor signals.

Priced Timed Automata:

Priced timed automata (PTA) extends timed automata (B1) with prices (B3). Prices allow the calculation of the cost of reaching a designated set of target states. Continuous behavior (B2) and parallel behavior (B4) are not part of the concept of priced timed automata. The formal definition of the concept PTA is described in Definition 9.

Definition 9 (Priced Timed Automata):

$PTA = (L, l_0, F, \Sigma, T, \Delta, c, P)$ is described as a 7-tuple, where

(i) **States:** States are defined as a finite set L. The state $l_0 \in L$ describes the initial state, e.g., the start state of a production system. The finite set $F \subseteq S$ describes the set of final states, e.g., the product is being manufactured, or the production system is in an error state. For example, the states of the example filling process are: heating l_0, measuring l_1, filling l_2, and moving l_3.

(ii) **Events:** The finite set Σ describes the alphabet comprising all events. For example, the alphabet comprising all events of the example filling process is $\Sigma = \{s_1, s_2, s_3, s_4\}$. An event s_1, s_2, s_3, or s_4 is triggered when the sensor valve is one, e.g., $s_1 = 1$.

(iii) **Transitions:** Transitions between states are described by the finite set $T \subseteq S \times \Sigma \times S$. For example, the following list of transitions exists for the sample filling process: between state l_0 and state l_1, between state l_1 and state l_2, between state l_2 and state l_3, and between state l_3 and state l_0.

(iv) **Single clock:** The single clock c is used to record time. The clock is reset at each transition.

(v) **Timing constraints:** The finite set Δ describes timing constraints with $\delta : T \rightarrow I$, where $\delta \in \Delta$ and I is the set of time intervals. For example, the timing constraint $c \geq 208s$ describes that a transition occurs only after 208 seconds.

(vi) **Prices:** $P : L \rightarrow \mathbb{N}$ assigns prices to states. For example, the state l_0 has a price of $10J$, e.g., energy consumption in Joules. More precisely, the price describes a cost rate. The costs of heating the state l_0 is $10J * 208s = 2080J$.

Figure 11 illustrates the priced timed automata of the example filling process.

State l_0 (heating): The filling process begins with the heating of the water $a_1 = 1$. Event s_1 is triggered and the heating of the water ends after $208s$. The state is switched from state l_0 to state l_1, the heating element is deactivated $a_1 = 0$, and valve $a_2 = 1$ is open.

State l_1 (measuring): Event s_2 is triggered. After $6s$ the filler is filled with $0.100l$ water. The state is switched from state l_1 to state l_2, valve a_2 is closed, and valve a_3 is open.

State l_2 (filling): Event s_3 is triggered. After $6s$ the bottle is filled with heated water. The state is switched from state l_2 to state l_3, valve a_3 is closed, and the conveyor $a_4 = 1$ starts to move the bottle.

State l_3 (moving): Event s_4 is triggered. After $10s$ the bottle is moved. The state is switched from state l_3 to state l_0, the conveyor system is off $a_4 = 0$, the heating element is on $a_1 = 1$, and the filling process starts again.

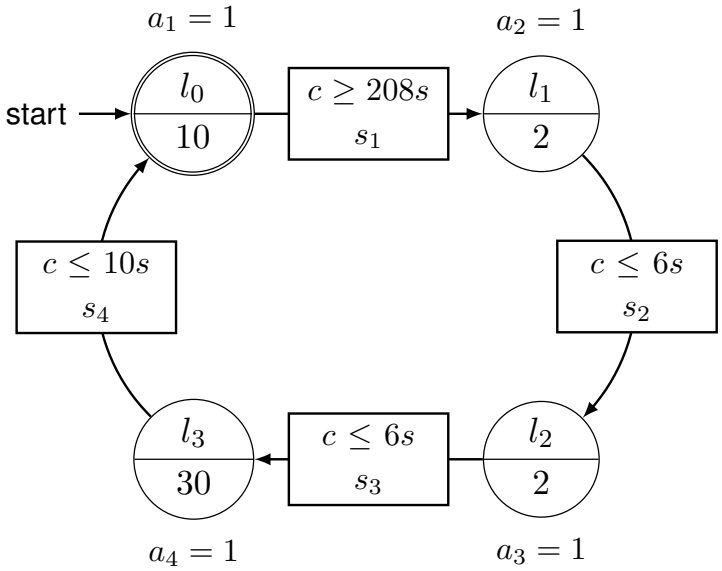


Figure 11 Priced timed automata of the example filling process: actuator a_1 controls the heating element, actuators a_2, a_3 control valve 1 and valve 2, actuator a_4 controls the conveyor system, sensor s_1 measures the temperature of the water, sensors s_2, s_3 are filling level sensors, and sensor s_4 is a light barrier sensor. Sensor signals $s_1 - s_2$ are events..

Table 9 summarizes the sequences of actuator signals $a_1 - a_4$ and sensor signals $s_1 - s_4$ of the example filling process. The column "Step" describes the iteration of the filling process from state l_0 to state l_0 via the states l_1, l_2, l_3, the column "Time" describes the overall time in seconds, the column "Price" describes the energy consumption in Joules, and the column "Constraint" describes the timing constraints of the timed automata.

				Actuators				**Sensors**			
Step	**Time**	**Cost**	**Constraint**	a_1	a_2	a_3	a_4	s_1	s_2	s_3	s_4
1	$0s$	0	$c \geq 208s$	1	0	0	0	1	0	0	0
1	$208s$	$2080J$	$c \leq 6s$	0	1	0	0	0	1	0	0
1	$304s$	$2092J$	$c \leq 6s$	0	0	1	0	0	0	1	0
1	$310s$	$2104J$	$c \leq 10s$	0	0	0	1	0	0	0	1
2	$518s$	$4484J$	$c \geq 208s$	1	0	0	0	1	0	0	0
2			...								
3	$540s$	$8968J$	$c \geq 208s$	1	0	0	0	1	0	0	0
			...								

Table 9 Priced timed automata of the example filling process: the column "Step" describes the iteration of the filling process, the column "Time" describes the overall time in seconds, the column "Cost" describes the overall energy consumption, the column "Constraint" describes timing constraints, the columns $a_1 - a_4$ describe sequences of actuator signals, and the columns $s_1 - s_4$ describe sensor signals.

3.3.2. Engineering Behavior Models

This section describes engineering behavior models. Recent research has focused on building engineering behavior models. Engineering behavior models can be used to validate, analyze, control, and simulate technical systems. Engineering behavior models are mostly used in combination with development processes. Each engineering behavior model describes different aspects of a technical system. Activity diagrams encode sequences of activities and consist of initial nodes, final nodes, activities, fork nodes, merge nodes and swim lanes [KST14]. Each swim lane reflects a hardware component, each activity reflects a software component, and fork nodes and merge nodes represent concurrent behavior. Petri nets describe the asynchronous, concurrent, distributed, and parallel behavior of information processing systems [Mur89], technical systems, or cyber-physical production systems.

Activity Diagram:

Knowledge about the production goal in the form of behavior models that encode sequences of activities is required to select an optimal control strategy for each activity and to calculate optimal parameters for each selected control strategy, as described by the three-step decision process in Section 2.3. This knowledge can be described by a UML 2 activity diagram [KST14]. Activity diagrams encode sequences of activities and consist of initial nodes, final nodes, activities, fork nodes, merge nodes, and swim lanes (B4). Timing behavior (B1), continuous behavior (B2), and cost behavior (B3) are not part of the activity diagram concept. It is described in Definition 10.

Definition 10 (Activity Diagram):

The subset of the UML 2 activity diagram consists of the following elements:

(i) **Initial node:** The node l_1 describes the start of a production scenario. It is called initial node, illustrated in Figure 12: Initial.

(ii) **Final node:** The node l_e describes the end of a production scenario. It is called final node, illustrated in Figure 12: Final.

(iii) **Activities:** An activity represents a software component with one or more control methods, illustrated in Figure 12: Activities.

(iv) **Fork nodes:** A fork node is a node that splits sequences of activities into multiple concurrent sequences of activities, illustrated in Figure 12: Fork.

(v) **Merge nodes:** A merge node is a node that brings together multiple concurrent sequences of activities, illustrated in Figure 12: Merge.

(vi) **Swim lanes:** A swim lane represents a modular hardware component, illustrated in Figure 12: Swim lanes.

Figure 12 illustrates the visual representation of the subset of the UML 2 activity diagram. Each swim lane reflects a hardware component, each activity reflects a software component, and fork nodes and merge nodes represent concurrent behavior.

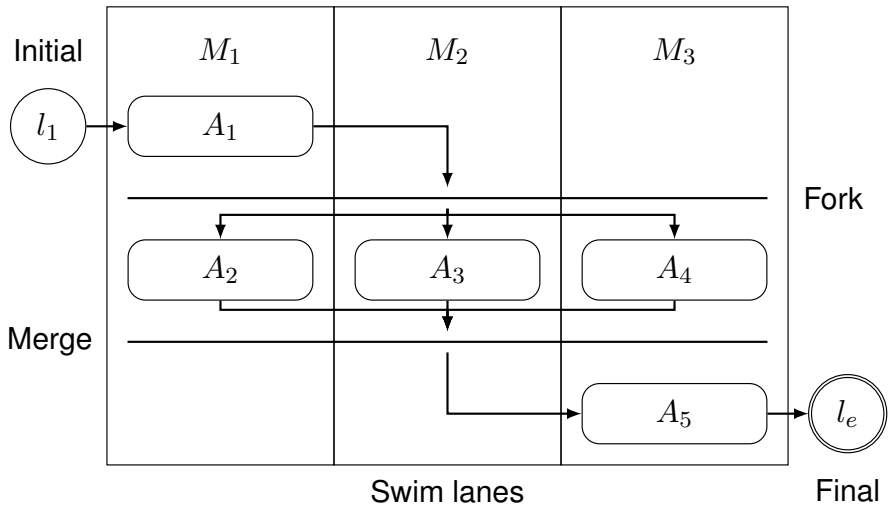


Figure 12 Subset of the UML 2 activity diagram: the initial node l_1 describes the start of a production scenario, the final node l_e describes the end of a production scenario, $A_1 - A_5$ describe activities, the fork node splits the sequences of activities into multiple concurrent sequences of activities, the merge node brings together multiple concurrent sequences of activities, and the swim lanes represent modular hardware components $M_1 - M_3$ (adapted from [OVN18b]).

Petri Net:

Petri nets describe the asynchronous, concurrent, distributed, and parallel behavior of information processing systems [Mur89], technical systems, or cyber-physical production systems (B4). Timing behavior (B1), continuous behavior (B2), and cost behavior (B3) are not a part of the concept of the Petri net. The formal definition of the concept of the Petri net is described in Definition 11.

Definition 11 (Petri Net):

A Petri net is a 5-tuple, $PN = (P, T, F, W, M_0)$, where:

(i) **Places:** P is defined as a finite set of places $P = \{p_1, \ldots, p_m\}$,

(ii) **Transitions:** T is defined as a finite set of transitions $T = \{t_1, \ldots, t_n\}$,

(iii) **Arcs:** F is defined as a finite set of arcs $F \subseteq (P \times T) \cup (T \times P)$,

(iv) **Weight function:** Weight function W is defined as $W : F \to \mathbb{N}$,

(v) **Initial marking:** Initial marking is defined as $M_0 : P \to \mathbb{N}$,

$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$

3.3.3. Comparison of Behavior Models

Table 10 describes the comparison of the behavior models presented above. None of the existing behavior models covers all requirements B1 - B4 described in Section 1.2.3.

Timed automata are capable of modeling the behavior of real-time systems over time, e.g., the timing behavior of cyber-physical production systems (B1). Continuous behavior (B2), cost behavior (B3), and parallel behavior (B4) are not part of the concept of timed automata. Hybrid timed automata are capable of modeling continuous behavior (B2) over time (B1) of real-time systems. Cost behavior (B3) and parallel behavior (B4) are not part of the concept of hybrid timed automata. Priced timed automata extend timed automata (B1) with prices (B3). Prices allow the calculation of the cost of reaching a designated set of target states. Continuous behavior (B2) and parallel behavior (B4) are not part of the concept of priced timed automata. Activity diagrams encode sequences of activities and consist of initial nodes, final nodes, activities, fork nodes, merge nodes, and swim lanes (B4). Timing behavior (B1), continuous behavior (B2), and cost behavior (B3) are not part of the concept activity diagram. Petri nets describe the asynchronous, concurrent, distributed, and parallel behavior of information processing, technical, or cyber-physical production systems (B4). Timing behavior (B1), continuous behavior (B2), and cost behavior (B3) are not part of the Petri net concept. The behavior model of CyberOpt is simply defined as a directed graph of activities (B4) with UML 2 activity diagram semantics. The design concept separation of concerns (SoC) is used in this work. Cost models (B3) are learned from timing behavior (B1) and continuous behavior (B2). The activities refer to control methods, and each control method has a cost model. Therefore, the behavior model of CyberOpt satisfies all requirements B1 - B4.

Behavior Model	**B1**	**B2**	**B3**	**B4**
Timed automata	$+$	$-$	$-$	$-$
Hybrid timed automata	$+$	$+$	$-$	$-$
Priced timed automata	$+$	$-$	$+$	$-$
Activity diagram	$-$	$-$	$-$	$+$
Petri net	$-$	$-$	$-$	$+$
CyberOpt approach	$+$	$+$	$+$	$+$

Table 10 Classification of different behavior models: ($+$ fulfilled and $-$ not fulfilled).

3.4. Conclusions From State of the Art

This section describes the conclusions drawn from state of the art. The analyzed state of the art is divided into the following three sections: (1) existing parameter configuration approaches, described in Section 3.1, (2) optimization models, described in Section 3.2, and (3) behavior models, described in Section 3.3.

The comparison of parameter configuration classes described in Section 3.1.5 has shown that none of the existing parameter solutions satisfies all defined requirements R1 - R6, but a com-

bination of data-driven and hybrid-based parameter configuration can satisfy all requirements. The manual parameter configuration involves the highest manual engineering effort and can be considered as the upper limit, while the data-driven parameter configuration represents the lower limit without any manual engineering effort. However, data-driven parameter configuration approaches are not generic solutions and are mostly specific for one use case, e.g., laser welding. Simulation-based parameter configuration involves a high manual engineering effort because simulation models must be created from data manually, and expert knowledge is required to create them. Furthermore, it is impossible to simulate the real world entirely, for computability reasons. Hybrid-based parameter configurations are a compromise between the model-building effort and optimization capabilities. They combine simulation models with observed data from the runtime of production systems. Except manual parameter configuration, the simulation-based, data-driven, and hybrid-based approaches are able to find a global optimum and model implicit timing parameters. None of the approaches use explicit sequences of control methods or parallel sequences of control methods or are able to handle different control methods. CyberOpt can be classified as a data-driven and hybrid-based command signal configuration solution. Observed data from production system runtime and machine learning techniques are used to reduce manual engineering effort. Cost models and sequences of control methods are learned, and cost models are improved. The learned cost models with implicit timing parameters, the learned behavior knowledge in the form of sequences of control methods, and the knowledge of the software structure can then be used to calculate a global optimal command signal configuration for a production system.

The comparison of optimization models described in Section 3.2.4 has shown that the command signal configuration problem introduced in Section 4.8 can be classified as a mixed-integer nonlinear programming problem with a nonlinear objective function, linear constraints, and a mixture of continuous and binary variables. Therefore, the command signal configuration problem of CyberOpt satisfies all the requirements M1 - M6. The modeling abilities come at a price. The command signal configuration problem belongs to the class of NP-hard complexity. Section 3.2.2 and Section 3.2.3 have shown that solving techniques and solvers exist to solve such problems.

The comparison of behavior models described in Section 3.3.3 has outlined that both classes of models, automata-based behavior models as well as engineering behavior models, can be used to represent different aspects of a technical system. No behavior model satisfies all requirements B1 - B4. The behavior model of CyberOpt is simply defined as a directed graph of activities with the semantics of the UML 2 activity diagram. Using the design concept of separation of concerns (SoC), it is unnecessary to define a behavior model representing all aspects (timing behavior, continuous behavior, cost behavior, and parallel behavior). Cost models are learned from timing behavior and continuous behavior. Activities refer to control methods, and each control method has a cost model. Therefore, the behavior model of CyberOpt satisfies all requirements B1 - B4.

4. Problem Description

This chapter describes the command signal configuration problem. This chapter is based on previous work by the author and expands the published concepts [ON15, OVN16, OVN18b, OVN18a]. The command signal configuration problem must be solved by a configuration approach in order to automate the task of manual configuration for automation software components in discrete manufacturing. The command signal configuration problem is described by a task with inputs and outputs as shown in Figure 13. The task describes a three-step decision process, see Section 2.3. The inputs of the task are: an automation software structure $\bar{A}$, described in Definition 12; command signals $\bar{P}$, described in Definition 13; decision parameters B, described in Definition 14; parameter constraints G, described in Definition 15; cost models C, described in Definition 16; sequences of control methods $\bar{S}$ described in Definition 17; and an objective time value obj, described in Definition 19. The output of the command signal configuration problem description is an optimal command signal configuration $X^* = (X, opt)$.

Inputs

Symbol	Name	Reference
$\bar{A}$	Automation software structure	Definition 12
$\bar{P}$	Command signals	Definition 13
B	Decision parameters	Definition 14
G	Command signal constraints	Definition 15
C	Cost models	Definition 16
$\bar{S}$	Sequences of controls methods	Definition 17
obj	Objective time value	Definition 19

Command signal configuration problem

Outputs

Symbol	Name	Reference
$X^* = (X, opt)$	Optimal command signal configuration, where $X = (\bar{P}, B)$ is a valid command signal configuration and opt is the cost value	Definition 18

Figure 13 Overview of the command signal configuration problem, described as a task with inputs and outputs.

4.1. Automation Software Structure

An automation software structure describes automation software components with one or more control methods that implement control strategies.

Definition 12 describes the formal notation of an automation software structure.

Definition 12 (Automation Software Structure):

$\bar{A} = (A_1, \ldots, A_m)$, $m \in \mathbb{N}_1$ is defined as an m-tuple of one or more reusable software components. It is called automation software structure.

Each reusable software component $A_i = \{1, \ldots, n_i\}$, $n_i \in \mathbb{N}_1$ is defined as finite set of control method indices and $n = \sum_{i=1}^{m} n_i$ is defined as the number of all control methods.

Example 1 describes the automation software structure of the example production system (see Figure 1). The automation software structure $\bar{A} = (A_1, A_2, A_3)$ has three automation software components $A_1 - A_3$ and five control methods $n = n_1 + n_2 + n_3 = 5$, see Equation 4.5. The software component A_1 has one control method *fill-B1*(P_{11}), the software component A_2 has two control methods: *pick-C1*(P_{21}), *pick-C2*(P_{22}), and the software component A_3 has two control methods: *move-D1*(P_{31}), *move-D2*(P_{32}).

Example 1 (Automation Software Structure):

The automation software structure of the example production system (see Figure 1):

SWC: $\boldsymbol{A_1}$
(1) *fill-B1*(P_{11})

SWC: $\boldsymbol{A_2}$
(1) *pick-C1*(P_{21})
(2) *pick-C2*(P_{22})

SWC: $\boldsymbol{A_3}$
(1) *move-D1*(P_{31})
(2) *move-D2*(P_{32})

$$\bar{A} = (A_1, A_2, A_3), \qquad m = 3 \tag{4.1}$$

$$A_1 = \{1\}, \qquad n_1 = 1 \tag{4.2}$$

$$A_2 = \{1, 2\}, \qquad n_2 = 2 \tag{4.3}$$

$$A_3 = \{1, 2\}, \qquad n_3 = 2 \tag{4.4}$$

$$n = n_1 + n_2 + n_3 = 5 \tag{4.5}$$

4.2. Command Signals

Command signals are used to increase the reusability of automation software components by adding degrees of freedom to control methods that implement control strategies.

Parameters are classified into three classes, see Section 2.1: (1) timing parameters, (2) command signals, and (3) decision parameters. Each control method j of the software component A_i has a timing parameter, command signals, and a decision parameter. The decision parameters are described in Section 4.3. The timing parameters of control strategies describe

the time period within which the control strategy should perform its activity, e.g., execute the control strategy $move\text{-}D1(P_{31})$ within 10 seconds, 15 seconds, or 20 seconds. Command signals specify the degrees of freedom of control strategy implementations, e.g., acceleration and deceleration parameters of conveyor systems.

Definition 13 describes the formal notation of command signals.

Definition 13 (Command Signals):

$\bar{P} = (P_{ij} \mid i \in \{1, \ldots, m\} \wedge j \in A_i)$ is defined as an n-tuple of all command signals, where $P_{ij} = (p_{ij}^{(1)}, \ldots, p_{ij}^{(k)})$, $k \in \mathbb{N}^{\geq 1}$ are the command signals of the control method j from the software component A_i.

The command signals $(p_{ij}^{(1)} \in D_{ij}^{(1)}), \ldots, (p_{ij}^{(k)} \in D_{ij}^{(k)})$ are defined by domains. A domain is a finite set of possible command signal values.

The command signal $p_{ij}^{(1)}$ describes the time to execute a control method j from software component A_i. Note that timing parameters are also command signals.

Example 2 describes all parameters of the example production system (see Figure 1). Equation 4.6 describes all command signals $\bar{P} = (P_{11}, P_{21}, P_{22}, P_{31}, P_{32})$. Equations 4.7 - 4.11 describe the parameters of the control methods $\textit{fill-B1}(P_{11})$, $move\text{-}D1(P_{31})$, $move\text{-}D2(P_{32})$, $pick\text{-}C1(P_{21})$ and $pick\text{-}C2(P_{22})$. The domains of these parameters are described in Equation 4.12.

Example 2 (Command Signals):

The command signals of the example production system (see Figure 1):

SWC: A_1
(1) $\textit{fill-B1}(\boldsymbol{P_{11}})$

SWC: A_2
(1) $pick\text{-}C1(\boldsymbol{P_{21}})$
(2) $pick\text{-}C2(\boldsymbol{P_{22}})$

SWC: A_3
(1) $move\text{-}D1(\boldsymbol{P_{31}})$
(2) $move\text{-}D2(\boldsymbol{P_{32}})$

$$\bar{P} = (P_{11}, P_{21}, P_{22}, P_{31}, P_{32}) \tag{4.6}$$

$$P_{11} = (p_{11}^{(1)}, p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)}) \tag{4.7}$$

$$P_{21} = (p_{21}^{(1)}) \tag{4.8}$$

$$P_{22} = (p_{22}^{(1)}) \tag{4.9}$$

$$P_{31} = (p_{31}^{(1)}) \tag{4.10}$$

$$P_{32} = (p_{32}^{(1)}) \tag{4.11}$$

$$D_{11}^{(1)}, D_{11}^{(2)}, D_{11}^{(3)}, D_{11}^{(4)}, D_{21}^{(1)}, D_{22}^{(1)}, D_{31}^{(1)}, D_{32}^{(1)} = \mathbb{R}^{>0} \tag{4.12}$$

Parameters $p_{11}^{(1)}, p_{21}^{(1)}, p_{22}^{(1)}, p_{31}^{(1)}$ and $p_{32}^{(1)}$ are timing parameters. Possible timing parameter values are described in Example 3: Equation 4.13 - Equation 4.15.

Example 3 (Timing Parameters):

Timing parameter values (s=seconds) of the example production system (see Figure 1):

$$p_{11}^{(1)} = 5s \tag{4.13}$$

$$\ldots \tag{4.14}$$

$$p_{32}^{(1)} = 10s \tag{4.15}$$

The parameters $p_{31}^{(2)}, p_{31}^{(3)}$, and $p_{31}^{(4)}$ of the example production system are command signals (see Figure 1). Possible command signal values are described in Example 4: Equation 4.16 - Equation 4.18.

Example 4 (Command Signals):

Command signals values of the example production system (Hz=Hertz) (see Figure 1):

$$p_{31}^{(2)} = 1Hz/s \tag{4.16}$$

$$p_{31}^{(3)} = 10Hz \tag{4.17}$$

$$p_{31}^{(4)} = 20Hz/s \tag{4.18}$$

4.3. Decision Parameters

A decision parameter describes which control method j of a software component A_i should be used to perform its corresponding activity. Only one control strategy can be selected for a software component. For example, only the control method $pick\text{-}C1(P_{21})$ or the control method $pick\text{-}C2(P_{22})$ can be used for the activity $pick$.

Definition 14 describes the formal notation of decision parameters.

Definition 14 (Decision Parameters):

$B = (b_{ij} \mid i \in \{1, \ldots, m\} \ \wedge \ j \in A_i \wedge b_{ij} \in \{0, 1\})$ is defined as an n-tuple of 1 or 0 values. If $b_{ij} = 1$, then the control method j from the software component A_i is used, and if $b_{ij} = 0$, then the control method j from software component A_i is not used.

Possible decision parameters of the example production system are described in Example 5: Equation 4.19 - Equation 4.20.

Example 5 (Decision Parameters):

Decision parameters of the example production system (see Figure 1):

$$B = (b_{11}, b_{21}, b_{22}, b_{31}, b_{32}) = (1, 1, 0, 1, 0) \tag{4.19}$$

$$B = (b_{11}, b_{21}, b_{22}, b_{31}, b_{32}) = (1, 0, 1, 1, 0) \tag{4.20}$$

4.4. Command Signal Constraints

Command signal constraints restrict command signals. They describe valid values of command signal configurations.

Definition 15 describes the formal notation of command signal constraints. Command signal constraints are defined as command signal constraint functions. The command signals $(p_{ij}^{(1)} \in D_{ij}^{(1)}), \ldots, (p_{ij}^{(k)} \in D_{ij}^{(k)})$ are defined by domains, see Definition 13. A command signal constraint function $g_{ij} : D_{ij}^{(1)} \times \cdots \times D_{ij}^{(k)} \rightarrow \{0, 1\}$ describes valid command signal values of timing parameters $p_{ij}^{(1)}$ and command signal parameters $p_{ij}^{(2)} \ldots p_{ij}^{(k)}$ of a control method j from software component A_i.

The function value of a command signal constraint function encodes the results of constraints. If $g_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)}) = 0$, then the command signal constraints are not fulfilled. If $g_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)}) = 1$, then the command signal constraints are fulfilled.

For example, the constraint $5s \leq p_{11}^{(1)} \leq 10s$ restricts the value of the timing parameter $p_{11}^{(1)} \in [5s, 10s]$. Note that in this case, defining constraints is equivalent to restricting the domains. In this work, all constraints are defined by command signal constraint functions.

Definition 15 (Command Signal Constraints):

$G = (g_{ij} \mid i \in \{1, \ldots, m\} \ \wedge \ j \in A_i)$ is defined as an n-tuple of command signal constraint functions, where $g_{ij} : D_{ij}^{(1)} \times \cdots \times D_{ij}^{(k)} \rightarrow \{0, 1\}$. It is called command signal constraint.

The command signal constraints of command signals of the example production system are described in Example 6. Equation 4.21 defines five command signal constraint functions $G = (g_{11}, g_{21}, g_{22}, g_{31}, g_{32})$.

The constraint function $g_{11}(p_{11}^{(1)}, p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)})$ defines a constraint for the timing parameter $p_{11}^{(1)}$ and constraints for the command signals $p_{11}^{(2)}, p_{11}^{(3)}$ and $p_{11}^{(4)}$, see Equation 4.22. The constraint functions $g_{21}(p_{21}^{(1)})$, $g_{22}(p_{22}^{(1)})$, $g_{31}(p_{31}^{(1)})$ and $g_{32}(p_{32}^{(1)})$ define a constraint for each timing parameter, see Equation 4.23.

Example 6 (Command Signal Constraints):

Command signal constraints of the example production system (see Figure 1):

$$G = (g_{11}, g_{21}, g_{22}, g_{31}, g_{32}) \tag{4.21}$$

$$g_{31}(p_{11}^{(1)}, p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)}) = \begin{cases} 1 & 7.7s \leq p_{11}^{(1)} \leq 26.25s \\ 1 & 1Hz/s \leq p_{11}^{(2)} \leq 20Hz/s \\ 1 & 10Hz \leq p_{11}^{(3)} \leq 50Hz \\ 1 & 1Hz/s \leq p_{11}^{(4)} \leq 20Hz/s \\ 1 & p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.22}$$

$$g_{11}(p_{31}^{(1)}), g_{21}(p_{21}^{(1)}), g_{22}(p_{22}^{(1)}), g_{32}(p_{32}^{(1)}) = \begin{cases} 1 & 5s \leq p_{ij}^{(1)} \leq 10s \text{ (timing parameter)} \\ 0 & \text{otherwise.} \end{cases} \tag{4.23}$$

4.5. Cost Models

Cost models are required for comparing control strategies and command signal configurations. A cost model is a mathematical description of measured cost values during the operation of a production system that allows the prediction of expected cost for a command signal configuration. In this work, the cost values are the energy consumption values.

Definition 16 describes the formal notation of cost models. Cost models are defined as cost functions. The command signals $(p_{ij}^{(1)} \in D_{ij}^{(1)}), \ldots, (p_{ij}^{(k)} \in D_{ij}^{(k)})$ are defined by domains, see Definition 13. Each control method j from the software component A_i has a cost function $c_{ij} : D_{ij}^{(1)} \times \cdots \times D_{ij}^{(k)} \to \mathbb{R}$. A cost function $c_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)})$ maps the command signal values $p_{ij}^{(1)}, \ldots, p_{ij}^{(k)}$ to a cost value.

Definition 16 (Cost Models):

$C = (c_{ij} \mid i \in \{1, \ldots, m\} \wedge j \in A_i)$ is defined as an n-tuple of cost functions $c_{ij} : D_{ij}^{(1)} \times \cdots \times D_{ij}^{(k)} \to \mathbb{R}$. It is called cost model.

The cost models for each control method of the example production system are described in Example 7. For each control method j from the software component A_i, a static cost function is defined in Equation 4.24 - Equation 4.26. This is just an example to explain the concept of optimal command signal configuration. Cost models in the real world are, e.g., complex nonlinear functions.

Example 7 (Cost Models):

Cost models of the example production system (see Figure 1):

$$C = (c_{11}, c_{21}, c_{22}, c_{31}, c_{32}) \tag{4.24}$$

$$c_{11}(p_{11}^{(1)}, p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)}), c_{22}(p_{22}^{(1)}), c_{31}(p_{31}^{(1)}) = 1 \tag{4.25}$$

$$c_{21}(p_{21}^{(1)}), c_{32}(p_{32}^{(1)}) = 10 \tag{4.26}$$

4.6. Sequences of Control Methods

Sequences of control methods describe all possible execution paths of an automation software structure. The sequences must be calculated from behavioral models that encode sequences of activities.

Definition 17 describes the formal notation of sequences of control methods.

Definition 17 (Sequences of Control Methods):

$\bar{S} = (S_1, \ldots, S_h, \ldots, S_r)$, $r \in \mathbb{N}_1$ are defined as an r-tuple of so-called possible execution paths of control methods. They are called sequences of control methods.

Each path $S_h = \{ij \mid i \in \{1, \ldots, m\} \wedge j \in A_i\}$ is defined as a finite set of index values ij used to denote domains D_{ij}, constraints g_{ij}, and costs c_{ij} of control method command signals P_{ij}. Note that it is called "sequences of control methods," but a path is defined as indices of command signals, see Example 8.

The sequences of control methods of the example production system are described in Example 8. The sequences of control methods are defined in Equations 4.27 - 4.31.

Example 8 (Sequences of Control Methods):

Sequences of control methods of the example production system (see Figure 1):

$$\bar{S} = (S_1, S_2, S_3, S_4) \tag{4.27}$$

$$S_1 = \{11, 21, 31\} \quad \textit{fill-B1}(P_{11}), \textit{pick-C1}(P_{21}), \textit{move-D1}(P_{31}) \tag{4.28}$$

$$S_2 = \{11, 22, 31\} \quad \textit{fill-B1}(P_{11}), \textit{pick-C2}(P_{22}), \textit{move-D1}(P_{31}) \tag{4.29}$$

$$S_3 = \{11, 21, 32\} \quad \textit{fill-B1}(P_{11}), \textit{pick-C1}(P_{21}), \textit{move-D2}(P_{32}) \tag{4.30}$$

$$S_4 = \{11, 22, 32\} \quad \textit{fill-B1}(P_{11}), \textit{pick-C2}(P_{22}), \textit{move-D2}(P_{32}) \tag{4.31}$$

4.7. Optimal Command Signal Configuration

An optimal command signal configuration is the result of a command signal configuration. An optimal command signal configuration is a command signal configuration with a minimal cost value. In this work, cost values are energy consumption values, and “optimal” means energy efficient.

Definition 18 describes the formal notation of an optimal command signal configuration.

Definition 18 (Optimal Command Signal Configuration):

Let $X = (\bar{P}, B)$ be a command signal configuration of command signals $\bar{P}$ (see Definition 13) and decision parameters B (see Definition 14). An optimal command signal configuration is a tuple $X^* = (X, opt)$. The value opt describes the cost value of the command signal configuration X.

The costs are calculated for the three valid command signal configurations of the example production system. The optimal command signal configuration X^* is $X^{(2)}$, see Equation 4.33, because it has the lowest cost value.

Example 9 (Optimal Command Signal Configuration):

Valid command signal configurations of the example production system (see Figure 1):

$$X^{(1)} = (X^{(1)},\ c_{11} + c_{21} + c_{31} = 13) \tag{4.32}$$

$$X^{(2)} = (X^{(2)},\ c_{11} + c_{22} + c_{31} = 3) = X^* \tag{4.33}$$

$$X^{(3)} = (X^{(3)},\ c_{11} + c_{22} + c_{32} = 13) \tag{4.34}$$

4.8. Command Signal Configuration Problem

This section describes the command signal configuration problem. The objective is to automatically calculate an optimal command signal configuration that can then be used to set up a production system. Given an automation software structure $\bar{A}$ (see Definition 12), command signals $\bar{P}$ (see Definition 13), decision parameters B (see Definition 14), command signal constraints G (see Definition 15), cost models C (see Definition 16) for each control method, sequences of control methods $\bar{S}$ (see Definition 16), and an objective time value obj defined in Definition 19, the objective is to find an optimal command signal configuration $X^* = (X, opt)$ (see Definition 18) that minimizes the costs. In this work, cost values are energy consumption values and "optimal" means energy efficient.

Definition 19 (Objective Time):

$obj \in \mathbb{R}^{>0}$ defines the objective time period for the execution of all control methods for each path S_h, see Definition 17. It is called objective time.

Definition 20 defines the formal notation of the command signal configuration problem. The minimization criterion is described in Equation 4.35. The command signals $(p_{ij}^{(1)} \in D_{ij}^{(1)}), \ldots, (p_{ij}^{(k)} \in D_{ij}^{(k)})$ are defined by domains, see Definition 13. The sum of the costs $c_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)})$ of all selected control methods should be minimized under:

Equation 4.36: Objective time constraint: The objective time constraint defines that the sum of the chosen values for the timing parameters $p_{ij}^{(1)}$, $p_{ij}^{(1)} \in D_{ij}^{(1)}$ of all selected control methods $p_{ij}^{(1)}\, b_{ij}$, $b_{ij} \in \{0, 1\}$ is smaller than the objective time value obj.

Equation 4.37: Command signal constraints are valid constraints: All command signal constraints are valid constraints describes that each command signal constraint function g_{ij}, $\forall\, i \in \{1, \ldots, m\}$, $j \in A_i$ must be valid $g_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)}) = 1$.

Equation 4.38: Only one control method constraint: The only one control method constraint describes that only one control method j can be selected from a software component A_i. The selected control method j is then used to execute the corresponding activity.

Definition 20 (Command Signal Configuration Problem):

Given an automation software structure $\bar{A}$ (see Definition 12), command signals $\bar{P}$ (see Definition 13), decision parameters B (see Definition 14), command signal constraints G (see Definition 15), cost models C (see Definition 16), sequences of control methods $\bar{S}$ (see Definition 17), and an objective time value obj (see Definition 19), the objective is to find an optimal command signal configuration $X^* = (X, opt)$ (see Definition 18) that minimizes the cost sum:

$$\begin{gathered} \min \sum_{i \in \{1,\ldots,m\}} \sum_{j \in A_i} c_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)})\, b_{ij}, \\ p_{ij}^{(1)} \in D_{ij}^{(1)}, \ldots, p_{ij}^{(k)} \in D_{ij}^{(k)}, b_{ij} \in \{0,1\} \end{gathered} \tag{4.35}$$

under the following constraints:

(i) **Objective time constraint:**

$$\begin{gathered} \sum_{ij \,\in\, S_h} p_{ij}^{(1)}\, b_{ij} \leq obj, \\ \forall\, h \in \{1,\ldots,r\}, \quad p_{ij}^{(1)} \in D_{ij}^{(1)}, b_{ij} \in \{0,1\}, obj \in \mathbb{R}^{>0} \end{gathered} \tag{4.36}$$

(ii) **Command signal constraints are valid constraints:**

$$\begin{gathered} g_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)}) = 1, \\ \forall\, i \in \{1,\ldots,m\},\ j \in A_i \end{gathered} \tag{4.37}$$

(iii) **Only one control method constraint:**

$$\begin{gathered} \sum_{j \,\in A_i} b_{ij} \ = 1, \\ \forall\, i \in \{1,\ldots,m\}, b_{ij} \in \{0,1\} \end{gathered} \tag{4.38}$$

5. CyberOpt Framework

This chapter introduces the CyberOpt framework. This chapter is based on previous work by the author and expands the published ideas [ON15, OVN16, OVN18b, OVN18a]. The CyberOpt framework is a formal framework for command signal configuration. Chapter 4 describes the command signal configuration problem. The command signal configuration problem must be solved by a configuration approach in order to find an optimal configuration. An optimal command signal configuration is a configuration with a minimal cost value. Cost values in this work are energy consumption values. The command signal configuration problem is described by the following concepts: an automation software structure $\bar{A}$ (see Definition 12), command signals $\bar{P}$ (see Definition 13), decision parameters B (see Definition 14), command signal constraints G (see Definition 15), cost models C (see Definition 16) for each control method, sequences of control methods $\bar{S}$ (see Definition 17), and an objective time value obj (see Definition 19). The CyberOpt framework, shown in Figure 14, describes tasks required to find an optimal command signal configuration $X^* = (X, opt)$ (see Definition 18).

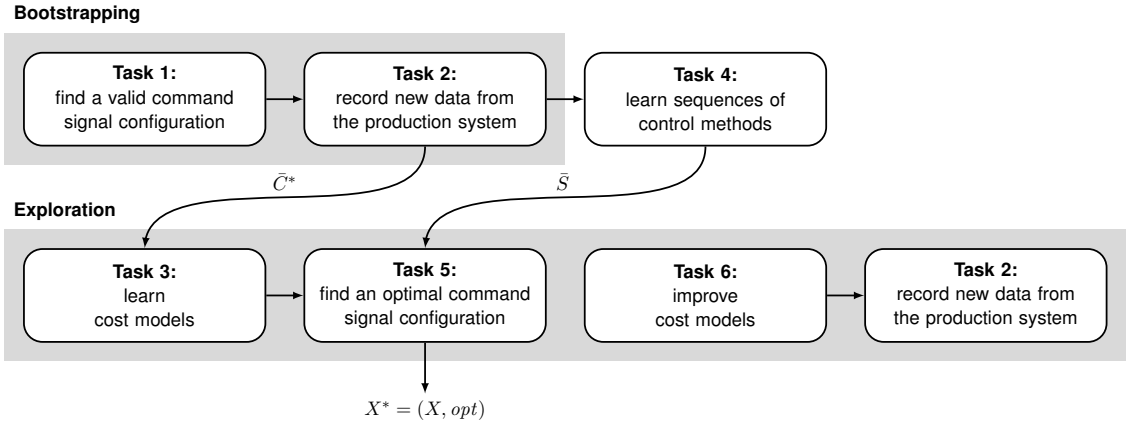


Figure 14 Overview of all tasks to find an optimal command signal configuration: cost model training data $\bar{C}^*$ (described in Section 5.2), sequences of control methods $\bar{S}$ (see Definition 17), and an optimal command signal configuration $X^* = (X, opt)$ (see Definition 18).

A task description is independent of a solution technique. The framework consists of six tasks, input and output concepts: Task 1: find a valid command signal configuration (described in Section 5.1), Task 2: record new data from the production system (described in Section 5.2), Task 3: learn cost models (described in Section 5.3), Task 4: learn sequences of control methods (described in Section 5.4), Task 5: find an optimal command signal configuration (described in Section 5.5), and Task 6: improve cost models (described in Section 5.6). The framework consists of two parts: (1) a bootstrapping part and (2) an exploration part. The goal of the bootstrapping part is to calculate the initial cost model training data $\bar{C}^*$ (see Definition 22) and to learn sequences of control methods $\bar{S}$ (see Definition 17). Therefore, task 1 calculates valid command signal configurations $X = (\bar{P}, B)$ (see Definition 21) from an automation software structure $\bar{A}$ (see Definition 12), command signals $\bar{P}$ (see Definition 13),

decision parameters B (see Definition 14), and command signal constraints G (see Definition 15). The concept valid command signal configuration $X = (\bar{P}, B)$ is described in Definition 21. A valid command signal configuration $X = (\bar{P}, B)$ (see Definition 21) is required to set up the automation software of a production system. When the automation software is set up, then Task 2 records new data from the production system to obtain an event log S^* (see Definition 23) and cost model training data $\bar{C}^*$. The concept event log S^* is described in Definition 23 and the concept cost model training data $\bar{C}^*$ is described in Definition 22. Task 4 learns sequences of control methods $\bar{S}$ (see Definition 17) from the recorded event log S^* (see Definition 23). The goal of the exploration part is to calculate an optimal command signal configuration $X^* = (X, opt)$ (see Definition 18). Task 3 learns cost models C (see Definition 16) from the recorded cost model training data $\bar{C}^*$ (see Definition 22). Task 5 solves the command signal configuration problem. Task 6 improves the quality of cost models C (see Definition 16). A new valid command signal configuration $X = (\bar{P}, B)$ (see Definition 21) is calculated, which should be evaluated in order to obtain more observations of the production system. With more observations, the cost models are more accurate. The number of evaluations of a production system is important. An exhaustive evaluation of all possible command signal configurations is usually not possible since each evaluation of a production system takes time and is therefore expensive. Few evaluations shorten the time in which a production system does not operate optimally. In this work, “optimal” means energy efficient.

5.1. Task 1: Find a Valid Command Signal Configuration

This section describes the task of finding a valid command signal configuration. The idea is to automatically calculate valid command signal configurations that can then be used to record new data from a production system. A valid command signal configuration describes: (1) which control method j is selected for each software component A_i and (2) which timing parameter value and command signal values are used for each selected control method j.

The formal definition of a valid command signal configuration X is described in Definition 21.

Definition 21 (Valid Command Signal Configuration):

A valid command signal configuration is a tuple $X = (\bar{P}, B)$ of command signals $\bar{P}$ (see Definition 13) and decision parameters B (see Definition 14).

Example 10 describes in Equation 5.1 - Equation 5.3 three valid command signal configurations $X^{(1)} - X^{(3)}$ of the example production system shown in Figure 1. For example, the tuple $(1, 1, 0, 1, 0)$ encodes the selected control methods. In this case, the control methods $\mathit{fill}\text{-}B1(P_{11})$, $\mathit{pick}\text{-}C1(P_{21})$, and $\mathit{move}\text{-}D1(P_{31})$ are selected. The timing parameter and

command signals used are $P_{11} = (5s, 1Hz, 10Hz, 20Hz)$, $P_{21} = (7s)$ and $P_{31} = (6s)$.

Example 10 (Valid Command Signal Configuration):

Valid command signal configurations of the example production system (see Figure 1):

$$\begin{aligned} X^{(1)} &= ((p_{11}^{(1)}, p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)}, p_{21}^{(1)}, p_{22}^{(1)}, p_{31}^{(1)}, p_{32}^{(1)}), (b_{11}, b_{21}, b_{22}, b_{31}, b_{32})) \\ &= ((5s, 1Hz, 10Hz, 20Hz, 7s, 8s, 6s, 10s), (1, 1, 0, 1, 0)) \end{aligned} \tag{5.1}$$

$$\begin{aligned} X^{(2)} &= ((p_{11}^{(1)}, p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)}, p_{21}^{(1)}, p_{22}^{(1)}, p_{31}^{(1)}, p_{32}^{(1)}), (b_{11}, b_{21}, b_{22}, b_{31}, b_{32})) \\ &= ((6s, 1Hz, 30Hz, 10Hz, 5s, 10s, 10s, 5s), (1, 0, 1, 1, 0)) \end{aligned} \tag{5.2}$$

$$\begin{aligned} X^{(3)} &= ((p_{11}^{(1)}, p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)}, p_{21}^{(1)}, p_{22}^{(1)}, p_{31}^{(1)}, p_{32}^{(1)}), (b_{11}, b_{21}, b_{22}, b_{31}, b_{32})) \\ &= ((10s, 1Hz, 10Hz, 5Hz, 10s, 6s, 7s, 8s), (1, 0, 1, 1, 0)) \end{aligned} \tag{5.3}$$

The formal definition of the task of finding a valid command signal configuration is given in Task 21.

Task 1 (Find a Valid Command Signal Configuration):

Given an automation software structure $\bar{A}$ (see Definition 12), command signals $\bar{P}$ (see Definition 13), decision parameters B (see Definition 14), and command signal constraints G (see Definition 15), the objective is to find a valid command signal configuration $X = (\bar{P}, B)$ (see Definition 21) under the following constraints:

(i) **Command signal constraints:**

$$\begin{aligned} g_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)}) = 1, \\ \forall\, i \in \{1, \ldots, m\}, j \in A_i, p_{ij}^{(1)} \in D_{ij}^{(1)}, \ldots, p_{ij}^{(k)} \in D_{ij}^{(k)} \end{aligned} \tag{5.4}$$

(ii) **Only one control method constraint:**

$$\begin{aligned} \sum_{j \in A_i} b_{ij} = 1, \\ \forall\, i \in \{1, \ldots, m\}, b_{ij} \in \{0, 1\} \end{aligned} \tag{5.5}$$

Equation 5.4 describes that all command signal constraints must be satisfied. Each command

signal constraint function $g_{ij}, \forall\, i \in \{1, \ldots, m\}$, and $j \in A_i$ must be valid $g_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)}) = 1$, $p_{ij}^{(1)} \in D_{ij}^{(1)}, \ldots, p_{ij}^{(k)} \in D_{ij}^{(k)}$. Equation 5.5 describes that only one control method j can be selected from a software component A_i. The selected control method j is then used to perform the corresponding activity.

5.2. Task 2: Record New Data From the Production System

This section describes the task of evaluating a production system. The problem of evaluating a production system is described by a task description with inputs and outputs, shown in Figure 15. The input of this task is a valid command signal configuration $X = (\bar{P}, B)$ (see Definition 21). The outputs of this task are cost model training data $\bar{C}^*$ (see Definition 22) and an event log S^* (see Definition 23).

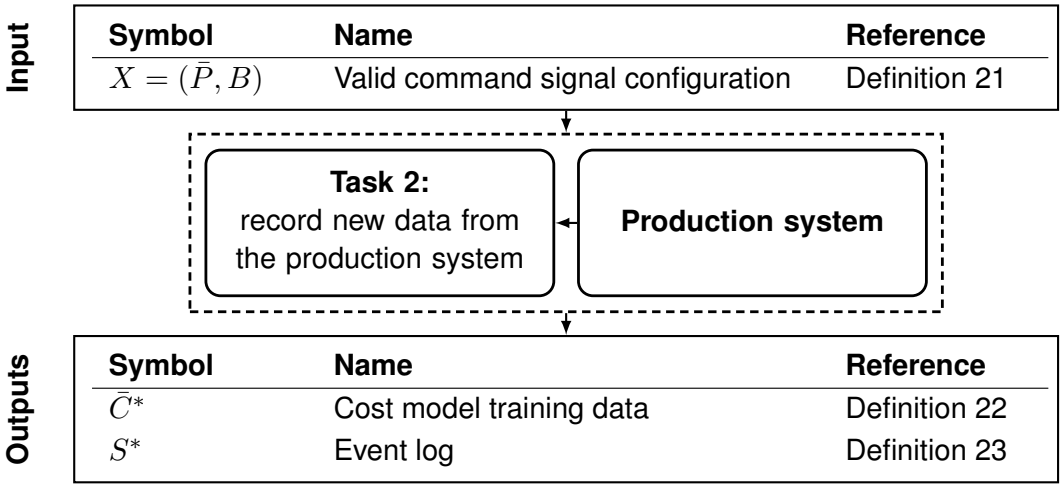


Figure 15 Task 2: Record new data from the production system.

Cost model training data $\bar{C}^*$ (see Definition 22) are needed to learn cost models C (see Definition 16). Cost models are required for comparing control strategies and command signal configurations. A cost model c_{ij} is a mathematical description of measured cost values during the operation of a production system that allows the prediction of expected cost for a command signal configuration. Training datasets are needed to calculate cost models.

The formal definition of cost model training data is described in Definition 22.

Definition 22 (Cost Model Training Data):

Let P_{ij} (see Definition 13) be a command signal configuration for control method j from software component A_i, then $C_{ij}^* = \{(P_{ij}^{(k)}, v^{(k)})\}_{k=1}^{n}, n \in \mathbb{R}^{>1}$ is defined as a tuple of a command signal configuration P_{ij} and a cost value v. It is called a cost model training dataset.

Cost model training data $\bar{C}^* = (C_{ij}^* \mid i \in \{1, \ldots, m\} \wedge j \in A_i)$ is defined as a tuple of cost model training datasets C_{ij}^*.

Example 11 describes a cost model training dataset C_{ij}^{*} and energy consumption values of control method $pick\text{-}C1(P_{21})$ from software component A_1 of the example production system shown in Figure 1.

Example 11 (Cost Model Training Dataset):

Cost model training dataset example of the example production system (see Figure 1):

$$\begin{aligned} C_{11}^{*} = &\{(P_{11}^{(1)}, 21.16J), (P_{11}^{(2)}, 42.33J), (P_{11}^{(3)}, 63.49J) \\ &(P_{11}^{(4)}, 84.65J), (P_{11}^{(5)}, 105.81J), (P_{11}^{(6)}, 126.97J)\} \end{aligned} \tag{5.6}$$

In this example, the unit is Joules.

Event logs of activities are used for learning behavior models. They include information about the number of evaluations of the production system, activities, originator hardware components, and time values.

The formal definition of an event log of activities is described in Definition 23. This definition is based on the definition in [WvDD06]. This concept was originally called transaction log.

Definition 23 (Event Log of Activities):

$S^{*} = ((s, a, o, t_s, t_e)_k)_{k=1}^{n}, n \in \mathbb{R}^{>1}$ is defined as an $n-$tuple of quadruples, where s describes the number of evaluations of the production system, a describes the activity, o describes the originator hardware component, t_s describes the start timestamp, and t_e describes the end timestamp.

Example 12 describes an event log of the example production system. In this example, the evaluations of the production system are recorded.

Example 12 (Event Log):

Event log of the example production system (see Figure 1):

$$\begin{aligned} S^{*} = &((1, A_1, M_1, \text{10:00:00}, \text{10:00:05})_1, \\ &(1, A_2, M_2, \text{10:00:05}, \text{10:00:10})_2, \\ &(1, A_3, M_3, \text{10:00:10}, \text{10:00:15})_3, \\ &(2, A_1, M_1, \text{11:00:00}, \text{10:00:06})_4, \\ &\dots) \end{aligned} \tag{5.7}$$

The formal definition of the task of recording new data from the production system is given in Task 2.

Task 2 (Record New Data From the Production System):

Given a valid command signal configuration $X = (\bar{P}, B)$ (see Definition 21), then:

(i) Set up the automation software with this valid command signal configuration $X = (\bar{P}, B)$ (see Definition 21),

(ii) measure observations during the runtime of the production system (see Section 2.2.1 and Section 2.2.2), and

(iii) transform observations into cost model training data $\bar{C}^*$ (see Definition 22) and into an event log of activities S^* (see Definition 23).

5.3. Task 3: Learn Cost Models

This section describes the task of learning cost models C (see Definition 16) from cost model training data $\bar{C}^*$ (see Definition 22). Unlike simulation-based parameter configuration approaches that use manually predefined optimization models or simulation models, an optimal command signal configuration approach should use machine learning algorithms to learn cost models from observations, reducing manual engineering effort. The idea is to automatically learn cost models using observations of production systems.

The formal definition of the task for learning cost models is described in Task 3.

Task 3 (Learn Cost Models):

Given cost model training data $\bar{C}^* = (C^*_{ij} \mid i \in \{1, \ldots, m\} \wedge j \in A_i)$ (see Definition 22), where $C^*_{ij} = \{(P^{(k)}_{ij}, v^{(k)})\}^n_{k=1}, n \in \mathbb{R}^{>1}$ are cost model training datasets, the objective is to learn a cost model c_{ij} for each control method j from software component A_i.

5.4. Task 4: Learn Sequences of Control

This section describes the task of learning sequences of control methods. The objective is to automatically learn sequences of control methods $\bar{S}$ (see Definition 17) using observations of production systems in the form of an event log S^* (see Definition 23).

The formal definition of the task of learning sequences of control is given in Task 4.

Task 4 (Learn Sequences of Control Methods):

Given an event log $S^* = ((s, a, o, t_s, t_e)_k)_{k=1}^n, n \in \mathbb{R}^{>1}$ (see Definition 23), the objective is to learn sequences of control methods $\bar{S} = (S_1, \ldots, S_h, \ldots, S_r),\ r \in \mathbb{N}_1$ (see Definition 17), where $S_h = \{ij \mid i \in \{1, \ldots, m\} \wedge j \in A_i\}$ describes a path of control methods.

5.5. Task 5: Find an Optimal Command Signal Configuration

This section describes the task of finding an optimal command signal configuration. The goal is to find an optimal command signal configuration $X^* = (X, opt)$ (see Definition 18) by solving a command signal configuration problem (see Definition 20). The command signal configuration problem is described in detail in Section 4.8. The formal definition of the task of finding an optimal command signal configuration is given in Task 5.

Task 5 (Find an optimal command signal configuration):

Solve a command signal configuration problem described in Definition 20 to find an optimal command signal configuration $X^* = (X, opt)$ (see Definition 18) that minimizes the cost. In this work, the cost is represented by energy consumption.

5.6. Task 6: Improve Cost Models

This section describes the task of improving cost models. The formal definition of the task of improving cost models is given in Task 6. Task 6 improves the quality of cost models C (see Definition 16). A new valid command signal configuration $X = (\bar{P}, B)$ (see Definition 21) is calculated, which should be evaluated in order to obtain more observations of the production system. With more observations, cost models are more accurate. The number of evaluations of command signal configurations is important. An exhaustive evaluation of all possible command signal configurations is usually impossible since each evaluation of a command signal configuration takes time and is therefore expensive. Few evaluations shorten the time during which a production system does not operate optimally.

Task 6 (Improve Cost Models):

Given an automation software structure $\bar{A}$ (see Definition 12), command signals $\bar{P}$ (see Definition 13), and cost model training data $\bar{C}^*$ (see Definition 22), the objective is to calculate a valid command signal configuration $X = (\bar{P}, B)$ (see Definition 21) that should be evaluated to improve cost models C (see Definition 16).

6. CyberOpt Algorithm

This chapter describes the CyberOpt algorithm, a novel approach to machine learning and constrained optimization-based command signal configuration. This chapter builds on previous work by the author and extends the published algorithms [ON15, OVN16, OVN18b, OVN18a]. CyberOpt algorithm automates the manual command signal configuration for automation software components in discrete manufacturing. CyberOpt automates the following two tasks:

(i) Selection of optimal control methods from software components of an automation software structure that implement different control strategies and

(ii) calculation of optimal command signal for selected control strategies.

Chapter 4 describes the command signal configuration problem. The problem must be solved by CyberOpt in order to find an optimal command signal configuration. An optimal configuration is a command signal configuration with a minimum cost value. Cost values in this work are energy consumption values. The CyberOpt algorithm uses the following machine learning techniques to reduce manual engineering effort:

T1 - Learn cost models: CyberOpt uses a machine learning technique of regression and polynomial expansion to learn cost models from production system observations (energy consumption values) to reduce manual engineering effort, as opposed to using manually predefined optimization models or simulation models.

T2 - Improve cost models: CyberOpt uses the expected improvement (EI) criterion to calculate new valid command signal configurations that should be evaluated in order to obtain more observations (energy consumption values) of the production system. Black-box optimization approaches use this technique to gather additional data by iterating between fitting cost models and gathering additional observations. With more observations, the cost models become more accurate.

T3 - Learn sequences of control methods: CyberOpt uses a machine learning technique named process mining to learn sequences of control methods from production system observations to reduce manual engineering effort. Manual creation of a behavior model is not necessary, reducing the manual engineering effort and synchronizing the learned behavior model with the current status of the production system.

T4 - Find optimal command signal configurations: CyberOpt uses mixed-integer nonlinear programming to find an optimal command signal configuration. The command signal configuration problem uses learned cost models, which are instances of predefined mathematical models, and learned sequences of control methods, which are instances of predefined be-

havior models, to find the optimal configuration.

Section 1.2 describes the requirements R1 - R6 for an optimal command signal configuration, the requirements M1 - M6 for an optimization model, and the requirements B1 - B4 for a behavior model. Table 11 summarizes the machine learning techniques T1 - T4, requirements R1 - R6 for an optimal command signal configuration, requirements M1 - M6 for a mathematical model, and requirements B1 - B4 for a behavior model. T4 calculates an optimal command signal configuration, and therefore it satisfies requirements R2 - R6 and M1 - M6. T1, T2, and T4 reduce the manual engineering effort R1 by automatically learning cost models, improving cost models, and learning sequences of control methods required to calculate an optimal command signal configuration. T1 also satisfies requirement R3 and requirements B1 - B3, T2 satisfies requirement R2, and T3 satisfies requirements R4, R5, and B4. Therefore, CyberOpt satisfies all the requirements described: R1 - R6, M1 - M6, and B1 - B4.

ID	Requirement	T1	T2	T3	T4
R1	Manual engineering steps	$+$	$+$	$+$	$-$
R2	Global optimum	$-$	$+$	$-$	$+$
R3	Implicit timing parameters	$-$	$-$	$-$	$+$
R4	Sequences of control methods	$-$	$-$	$-$	$+$
R5	Parallel sequences of control methods	$-$	$-$	$-$	$+$
R6	Selection of different control methods	$-$	$-$	$-$	$+$
M1	Linear objective function	$-$	$-$	$-$	$+$
M2	Nonlinear objective function	$-$	$-$	$-$	$+$
M3	Linear constraints	$-$	$-$	$-$	$+$
M4	Nonlinear constraints	$-$	$-$	$-$	$+$
M5	Continuous variables	$-$	$-$	$-$	$+$
M6	Discrete variables	$-$	$-$	$-$	$+$
B1	Time behavior	$+$	$-$	$-$	$-$
B2	Continuous behavior	$+$	$-$	$-$	$-$
B3	Cost behavior	$+$	$-$	$-$	$-$
B4	Parallel behavior	$-$	$-$	$+$	$-$

Table 11 Requirements R1 - R6 for an optimal command signal configuration, requirements M1 - M6 for a mathematical model, requirements B1 - B4 for a behavior model, and the corresponding features ($+$ fulfilled and $-$ not fulfilled).

Chapter 5 describes the CyberOpt framework. The CyberOpt framework describes necessary tasks to find an optimal command signal configuration. A task description is independent of a solution technique. The framework consists of six tasks: Task 1: find a valid command signal configuration, Task 2: record new data from the production system, Task 3: learning cost models, Task 4: learn sequences of control methods, Task 5: find an optimal command signal configuration, and Task 6: improve cost models.

The CyberOpt algorithm consists of five subalgorithms: (1) CyberOpt-SIC subalgorithm described in Section 6.1, (2) CyberOpt-LCM subalgorithm described in Section 6.2, (3) CyberOpt-LSC subalgorithm described in Section 6.3, (5) CyberOpt-SPC subalgorithm de-

scribed in Section 6.4, and (4) CyberOpt-ICM subalgorithm described in Section 6.5. Each subalgorithm realizes a task of the CyberOpt framework. Table 12 summarizes input and output concepts of tasks implemented by CyberOpt subalgorithms.

Task	Subalgorithm	Technique	Inputs	Outputs
Task 1	CyberOpt-SIC	$-$	$\bar{A}$, $\bar{P}$, B, G	$X = (\bar{P}, B)$
Task 2	$-$	$-$	$X = (\bar{P}, B)$	$\bar{C}^*$, S^*
Task 3	CyberOpt-LCM	T1	$\bar{C}^*$	C
Task 4	CyberOpt-LSC	T3	S^*	$\bar{S}$
Task 5	CyberOpt-SPC	T4	$\bar{A}$, $\bar{P}$, B, G, C, $\bar{S}$, obj	$X^* = (X, opt)$
Task 6	CyberOpt-ICM	T2	$\bar{A}$, $\bar{P}$, C	$X = (\bar{P}, B)$

Table 12 Overview of algorithms: automation software structure $\bar{A}$ described in Definition 12, command signals $\bar{P}$ described in Definition 13, decision parameters B described in Definition 14, command signal constraints G described in Definition 15, objective time value obj described in Definition 19, sequences of control methods $\bar{S}$ described in Definition 17, cost models C described in Definition 16, event log of activities S^* described in Definition 23, cost model training data $\bar{C}^*$ described in Definition 22, and valid command signal configuration X described in Definition 21.

The CyberOpt-SIC subalgorithm realizes Task 1. The subalgorithm samples valid command signal configurations $X = (\bar{P}, B)$ (see Definition 21). A valid command signal configuration X is required to set up the automation software of a production system. When the automation software is set up, Task 2 records new data from the production system to obtain an event log of activities S^* (see Definition 23) and cost model training data $\bar{C}^*$ (see Definition 22). The CyberOpt-LSC subalgorithm realizes Task 4. The subalgorithm learns sequences of control methods $\bar{S}$ from an event log of activities S^*. The CyberOpt-LCM subalgorithm realizes Task 3. The subalgorithm learns cost models C (see Definition 16) from cost model training data $\bar{C}^*$ (see Definition 22). Cost model training data is recorded by an evaluation of a valid command signal configuration described in Task 2. The CyberOpt-SPC subalgorithm realizes Task 5. The subalgorithm solves the command signal configuration problem. The CyberOpt-ICM subalgorithm realizes Task 6. The subalgorithm improves the quality of cost models C (see Definition 16). A new valid command signal configuration X (see Definition 21) is calculated, which should be evaluated in order to obtain more observations of the production system.

The CyberOpt algorithm is described in Algorithm 1 and has the following four algorithm parameters:

Objective time: Objective time $obj \in \mathbb{R}^{>0}$ defines the objective time period to execute all control methods of an automation software, see Definition 19.

Bootstrapping steps: Parameter $\delta \in \mathbb{R}^{>2}$ defines the number of recording steps of the production system. Task 2 records new data from the production system to obtain an event log of activities S^* and cost model training data $\bar{C}^*$. At least two recording steps are required to learn a linear cost model for each control method.

Termination criterion: Parameter $\varrho \in \mathbb{R}^{>1}$ defines the maximum allowed number of itera-

tions of the CyberOpt algorithm. It is the termination criterion of the CyberOpt algorithm. The CyberOpt algorithm iterates the subalgorithms, and the production system is evaluated automatically to acquire new data from the production system in each iteration. Each evaluation of a production system takes time and is therefore expensive. A domain expert can define the maximal number of iterations to find an optimal command signal configuration. For example, if an evaluation of a production system takes 5 minutes and the maximal iterations are set to $\varrho = 10$, then the total evaluation time of the production system is 50 minutes.

Quality of command signal configuration: Parameter $\epsilon \in \mathbb{R}^{>0}$ specifies the maximum distance to a new optimal command signal configuration. It defines the quality of the calculated command signal configuration. In each iteration, new data is acquired from the production system, but each evaluation of a production system takes time and is therefore expensive. For example, suppose a domain expert sets the maximum iterations to $\varrho = 10$ and the maximum distance to $\epsilon = 200J$. In the best case, the CyberOpt algorithm terminates before ten evaluations are performed because of the assumption that a better optimal command signal configuration will be found in each iteration. In the worst case, the CyberOpt algorithm terminates after ten evaluations. Then a better optimal command signal configuration is found in each iteration, and the energy consumption of the production system can be improved by more than $200J$. A domain expert can then decide to increase the maximum iterations ϱ to have more time to find an optimal command signal configuration.

The CyberOpt algorithm consists of the following two steps:

Algorithm 1: Step 1 (Bootstrapping): In the first step, sequences of control methods $\bar{S}$ (see Definition 17) are learned from an event log of activities and initial cost model training data $\bar{C}^*$ (see Definition 22) are recorded, represented by function $bootstrapping$. The CyberOpt bootstrapping step is described in Algorithm 2.

Algorithm 1: Step 2 (Exploration): In the second step, an optimal command signal configuration $X^* = (X, opt)$ (see Definition 18) is calculated, represented by the function $exploration$. The CyberOpt exploration step is described in Algorithm 3.

Algorithm 1: CyberOpt algorithm

Input : Bootstrapping steps δ
: Termination criterion ϱ
: Quality of command signal configuration ϵ
: Maximal degree of polynomial features λ
: Objective time obj
: Automation software structure $\bar{A}$
: Command signals $\bar{P}$
: Decision parameters B
: Command signal constraints G
Output : Optimal command signal configuration $X^* = (X, opt)$

▷ **Step 1 (Bootstrapping):**

1 $\bar{S}, \bar{C}^* \leftarrow bootstrapping(\delta, \bar{A}, \bar{P}, B, G)$

▷ **Step 2 (Exploration):**

2 $X, opt \leftarrow exploration(\varrho, \epsilon, \lambda, obj, \bar{S}, \bar{C}^*, \bar{A}, \bar{P}, B, G)$

3 **return** X, opt

The CyberOpt bootstrapping step is described in Algorithm 2. The algorithm consists of the following three steps:

Algorithm 2: Step 1 (Sample a valid command signal configuration): The first step is to calculate a valid command signal configuration X (see Definition 21). A valid command signal configuration $X = (\bar{P}, B)$ is a tuple of command signals $\bar{P}$ (see Definition 13) and decision parameters B (see Definition 14). The sampling of the valid command signal configuration X is realized by the CyberOpt-SIC subalgorithm described in Section 6.1. The CyberOpt-SIC subalgorithm is represented by function SIC.

Algorithm 2: Step 2 (Record new data from the production system): In the second step, the automation software is set up with the valid command signal configuration X and new data from the production system is recorded, represented by function $record$. The function implements Tasks 2 and encapsulates the recording of an event log of activities S^* (see Definition 23) and cost model training data $\bar{C}^*$ (see Definition 22).

Algorithm 2: Step 3 (Learn sequences of control methods): In the third step, the sequences of control methods $\bar{S}$ (see Definition 17) are learned from the event log of activities S^* (see Definition 23). The learning of sequences of control methods $\bar{S}$ is realized by the CyberOpt-LSC subalgorithm, which is described in Section 6.3. The CyberOpt-LSC subalgorithm is represented by function LSC. Sequences of control methods $\bar{S}$ are required to solve the command signal configuration problem, see Definition 20.

Algorithm 2: CyberOpt bootstrapping

Input : Bootstrapping steps δ
: Automation software structure $\bar{A}$
: Command signals $\bar{P}$
: Decision parameters B
: Command signal constraints G
Output : Sequences of control methods $\bar{S}$
: Cost model training data $\bar{C}^*$

1 **for** 1 *to* δ **do**

▷ **Step 1 (Sample a valid command signal configuration):**

2 $X \leftarrow SIC(\bar{A}, \bar{P}, B, G)$

▷ **Step 2 (Record new data from the production system):**

3 $S^*, \bar{C}^* \leftarrow record(X, S^*, \bar{C}^*)$

▷ **Step 3 (Learn sequences of control methods)**

4 $\bar{S} \leftarrow LSC(S^*, \bar{A})$

5 **return** $\bar{S}, \bar{C}^*$

The CyberOpt exploration step is described in Algorithm 3. The algorithm consists of the following five steps:

Algorithm 3: Step 1 (Learn cost models): In the first step, for each control method j from software component A_i, a cost model c_{ij} is learned from cost model training data C^*_{ij} (see Definition 22). The learning of cost models C (see Definition 16) is realized by the CyberOpt-LCM subalgorithm described in Section 6.2. The CyberOpt-LCM subalgorithm is represented by function LCM.

Algorithm 3: Step 2 (Solve the command signal configuration problem): In the second step, the command signal configuration problem (see Definition 20) is solved. The solving is realized by the CyberOpt-SPC subalgorithm described in Section 6.4. The CyberOpt-SPC subalgorithm is represented by the function SPC.

Algorithm 3: Step 3 (Check if the command signal configuration is good enough): In the third step, the termination criterion of CyberOpt algorithm is checked. For this purpose, the Euclidean distance $\|\hat{opt} - opt\| \leq \epsilon$ between the newly calculated optimal energy consumption value opt and the previously calculated optimal energy consumption value $\hat{opt}$ is calculated. If opt is not better than $\hat{opt}$, then the calculation is finished. The CyberOpt algorithm returns an optimal command signal configuration $X^* = (X, opt)$ (see Definition 18). Parameter ϵ is used to describe a deviation between opt and $\hat{opt}$, e.g., $\epsilon = 0.05$.

Algorithm 3: Step 4 (Create a new command signal configuration to improve cost models): In the fourth step, a new valid command signal configuration $X = (\bar{P}, B)$ (see Definition 21) is calculated to be evaluated by the production system to improve the quality of the cost models C (see Definition 16). Therefore, for each control method j from software

component A_i, the CyberOpt-ICM subalgorithm calculates a new command signal configuration P_{ij} (see Definition 13) represented by the function ICM. The implementation of the CyberOpt-ICM subalgorithm is described in Section 6.5.

Algorithm 3: Step 5 (Record new data from the production system): In the fifth step, the automation software is set up with the valid command signal configuration $X = (\bar{P}, B)$ (see Definition 21) and new data from the production system is recorded, represented by function $record$. The function implements Tasks 2 and encapsulates the recording cost model training data $\bar{C}^*$ (see Definition 22).

Algorithm 3: CyberOpt exploration

Input : Objective time obj
: Termination criterion ϱ
: Quality of command signal configuration ϵ
: Maximal degree of polynomial features λ
: Sequences of control methods $\bar{S}$
: Cost model training data $\bar{C}^*$
: Automation software structure $\bar{A}$
: Command signals $\bar{P}$
: Decision parameters B
: Command signal constraints G
Output : Optimal command signal configuration $X^* = (X, opt)$

$\hat{opt} \leftarrow \infty$;
for 1 *to* ϱ **do**

▷ **Step 1 (Learn cost models):**

$C \leftarrow (c_{ij} \mid i \in \{1, \ldots, m\} \ \wedge \ j \in A_i)$ **for** $j \in A_i$ **do**
  for $i \in \{1, \ldots, m\}$ **do**
    $c_{ij} \leftarrow LCM(\lambda, C^*_{ij})$

▷ **Step 2 (Solve the command signal configuration problem):**

$X^* \leftarrow SPC(\bar{A}, \bar{P}, B, G, C, \bar{S}, obj)$

▷ **Step 3 (Check if the command signal configuration is good enough):**

if $\|\hat{opt} - opt\| \leq \epsilon$ **then**
  return X^*
$\hat{opt} \leftarrow opt$

▷ **Step 4 (Create a new command signal configuration):**

$\bar{P} \leftarrow (P_{ij} \mid i \in \{1, \ldots, m\} \ \wedge \ j \in A_i)$ **for** $j \in A_i$ **do**
  for $i \in \{1, \ldots, m\}$ **do**
    $P_{ij} \leftarrow ICM(C^*_{ij})$
$B^* \leftarrow random(B)$ $X \leftarrow (\bar{P}, B^*)$

▷ **Step 5 (Record new data from the production system):**

$\bar{C}^* \leftarrow record(X, \bar{C}^*)$

6.1. CyberOpt-SIC Subalgorithm

This section describes the CyberOpt-SIC subalgorithm. The CyberOpt-SIC subalgorithm realizes Task 1, which is shown in Figure 16. The motivation of the CyberOpt-SIC subalgorithm is to automatically sample a valid command signal configuration X. A valid command signal configuration X is required to set up automation software. When automation software is set up with a valid command signal configuration X, energy consumption values can be measured, and an event log of activities can be recorded during the runtime of a production system, see Task 2.

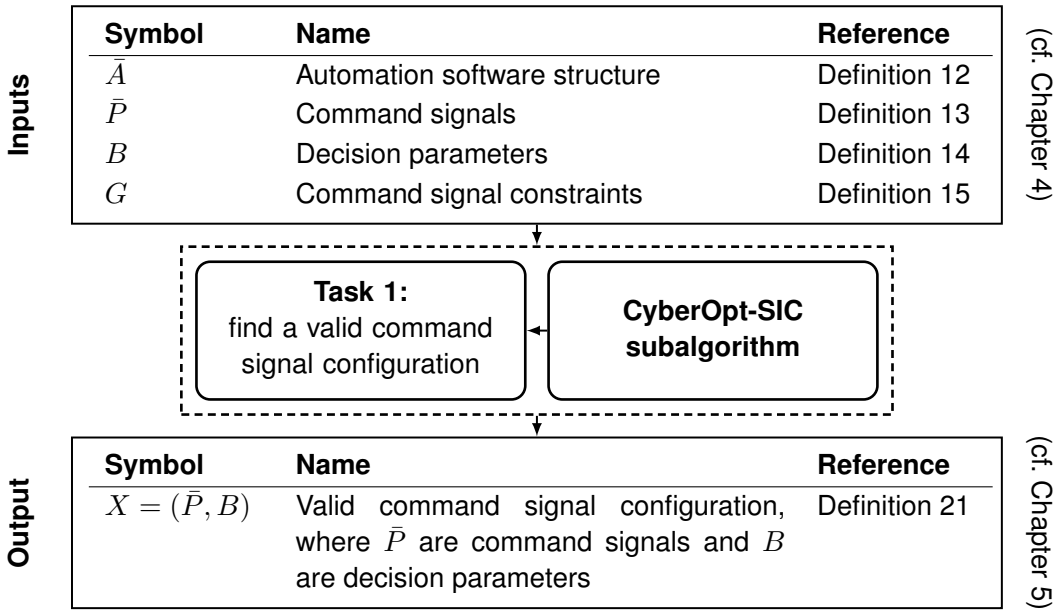


Figure 16 Task 1: Find a valid command signal configuration.

The CyberOpt-SIC subalgorithm realizes Task 1 and is described in Algorithm 4. Command signals $\bar{P}$ are uniformly sampled, and then validated by command signal constraints G, represented by function $random\_parameters$. The decision parameters B are sampled from a discrete uniform distribution, repeated by function $random\_decisions$.

Algorithm 4: The CyberOpt-SIC subalgorithm samples a valid command signal configuration

Input : Automation software structure $\bar{A}$
: Command signals $\bar{P}$
: Decision parameters B
: Command signal constraints G
Output : Valid command signal configuration $X = (\bar{P}, B)$

1 $\bar{P} \leftarrow (P_{ij} \mid i \in \{1, \ldots, m\} \ \wedge \ j \in A_i)$ **for** $j \in A_i$ **do**
2 **for** $i \in \{1, \ldots, m\}$ **do**
3 $P_{ij} \leftarrow random\_parameters(g_{ij})$
4 $B^* \leftarrow random\_decisions(B, \bar{A})$
5 $X \leftarrow (\bar{P}, B^*)$
6 **return** X

6.2. CyberOpt-LCM Subalgorithm

This section describes the CyberOpt-LCM subalgorithm. The CyberOpt-LCM subalgorithm realizes Task 3, shown in Figure 17. The motivation of the CyberOpt-LCM subalgorithm is to automatically learn a cost model c_{ij} for each control method j from a software component A_i. In order to compare different command signal configurations, cost models are required to find an optimal command signal configuration. Manual creation of cost models is not required, reducing the manual engineering effort.

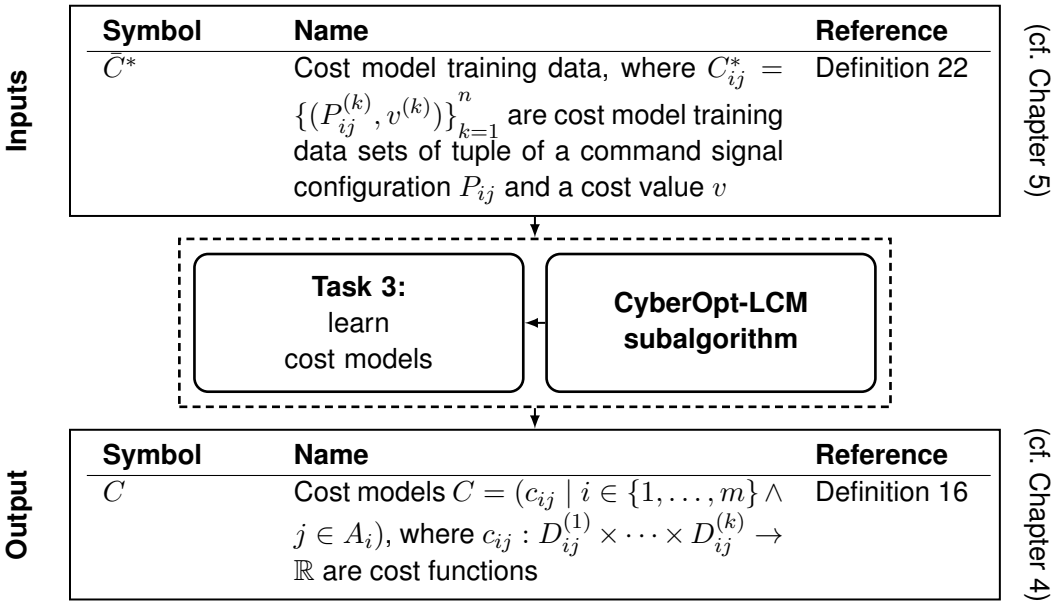


Figure 17 Task 3: Learn cost models.

Regression analysis can be used to learn cost models. Regression analysis describes a statistical process for estimating relationships between dependent and independent variables. In this work, the dependent variable is energy consumption, and the independent variables are command signals. The most common form of regression analysis is linear regression.

Definition 24 describes the concept of a linear cost model. For example, the method of ordinary least squares calculates coefficient values W_{ij} of a linear model $c_{ij}(W_{ij}, P_{ij})$ such that the distances between observed values $C^*_{ij} = \{(P^{(k)}_{ij}, v^{(k)})\}^n_{k=1}, n \in \mathbb{R}^{>1}$ and values of the linear model $c_{ij}(W_{ij}, P_{ij})$ are minimized. The linear model can then be used to predict energy consumption values.

Definition 24 (Linear Cost Models):

The linear cost model $W_{ij} = (w^{(1)}_{ij}, \ldots, w^{(k+1)}_{ij})$ is defined as $(k+1)$-tuple of coefficients, where $w^{(1)}_{ij}$ is defined as intercept. c_{ij} is defined as $c_{ij}(W_{ij}, P_{ij}) = w^{(1)}_{ij} + w^{(2)}_{ij} p^{(1)}_{ij} + \cdots + w^{(k+1)}_{ij} p^{(k)}_{ij}$.

In cases where the energy consumption is not linear, polynomial expansion can be used. Polynomial expansion is a transformation of the feature space [Ger74]. It allows linear models to capture nonlinearities; more precisely, it allows a linear model to learn polynomial relationships between dependent and independent variables [YHL12]. Definition 25 describes the concept of nonlinear cost model.

Definition 25 (Nonlinear Cost Model):

$W_{ij} = (w_{ij}^{(1)}, \ldots, w_{ij}^{(k+1)})$ is defined as $(k+1)$-tuple of coefficients, where $w_{ij}^{(1)}$ is defined as intercept. Let $expand(P_{ij}, d)$ be an algorithm [NH18] that calculates a polynomial expansion of degree d, then the calculated polynomial expansion is called nonlinear cost model.

For example, $expand((p_{ij}^{(1)}, p_{ij}^{(2)}, p_{ij}^{(3)}), 2)$ generates the following nonlinear cost model, see Example 13:

Example 13 (Polynomial Expansion):

$$c(P_{ij}, W_{ij}) = p_{ij}^{(1)} w_{ij}^{(2)} + p_{ij}^{(2)} w_{ij}^{(3)} + p_{ij}^{(3)} w_{ij}^{(4)} + w_{ij}^{(1)} \tag{6.1}$$

For example, $expand((p_{ij}^{(1)}, p_{ij}^{(2)}, p_{ij}^{(3)}), 3)$ generates the following nonlinear cost model, see Example 14:

Example 14 (Polynomial Expansion):

$$\begin{aligned} c(P_{ij}, W_{ij}) = {} & p_{ij}^{(1)} w_{ij}^{(2)} + p_{ij}^{(2)} w_{ij}^{(3)} + p_{ij}^{(3)} w_{ij}^{(4)} && (6.2)\\ & + p_{ij}^{(1)} p_{ij}^{(1)} w_{ij}^{(1)} w_{ij}^{(5)} + p_{ij}^{(1)} p_{ij}^{(2)} w_{ij}^{(6)} && (6.3)\\ & + p_{ij}^{(1)} p_{ij}^{(3)} w_{ij}^{(7)} + p_{ij}^{(1)} w_{ij}^{(2)} && (6.4)\\ & + p_{ij}^{(2)} p_{ij}^{(2)} w_{ij}^{(8)} + p_{ij}^{(2)} p_{ij}^{(3)} w_{ij}^{(9)} && (6.5)\\ & + p_{ij}^{(2)} w_{ij}^{(3)} + p_{ij}^{(3)} p_{ij}^{(3)} w_{ij}^{(10)} + p_{ij}^{(3)} w_{ij}^{(4)} + w_{ij}^{(1)} && (6.6) \end{aligned}$$

Nonlinear cost models are used in this work. Manual approximation with linear cost models is not necessary because MILP problems and MINLP problems are in the NP-hard complexity class. Section 3.2.2 describes various solving techniques for solving convex and nonconvex MINLP problems. In cases where the cost models are nonconvex, any global MINLP solver introduced in Section 3.2.3 automatically uses underestimating or overestimating functions, separable functions, or the problem is rewritten using factorization techniques.

The CyberOpt-LCM subalgorithm is described in Algorithm 5. Given a cost model training dataset $C_{ij}^{*} = \{(P_{ij}^{(k)}, v^{(k)})\}_{k=1}^{n}, n \in \mathbb{R}^{>1}$, the algorithm learns a cost model c_{ij} for control method j from the automation software component A_i. The CyberOpt-LCM subalgorithm consists of the following four steps:

Algorithm 5: Step 1 (Generate polynomial expansion): In the first step, a polynomial expansion for a command signal configuration $P_{ij}^{(1)}$ is generated. The algorithm parameter d describes the degree of the generated polynomial.

Algorithm 5: Step 2 (Fit the generated cost model c to data C_{ij}^{*}): In the second step, the generated nonlinear cost model c is fitted to the cost model training dataset C_{ij}^{*}.

Algorithm 5: Step 3 (Calculate the regression score between data C_{ij}^{*} and cost model c^{*}): In the third step, a score value $s \in [-\infty, 1]$ is calculated by a regression score function. In this case, the "coefficient of determination" is used, where $s = 1$ describes the best possible regression score value.

Algorithm 5: Step 4 (Select cost model with the highest score): In the fourth step, the nonlinear cost model c^{*} with the highest regression score value is selected because this nonlinear cost model has the best fit to the cost model training dataset C_{ij}^{*}.

The first step is repeated until the maximum degree λ is reached.

Algorithm 5: The CyberOpt-LCM subalgorithm learns a cost model c_{ij} from a cost model training dataset C_{ij}^{*}

Input : Maximum degree of polynomial features λ
: Cost model training dataset C_{ij}^{*}
Output : Cost model c_{ij}

$R \leftarrow \emptyset$; ▷ a set of tuple (s, c^{*}) of score regression value s and learned cost model c^{*}

for $d = 1$ *to* λ **do**

▷ **Step 1 (Generate polynomial expansion):**

$c \leftarrow expansion(P_{ij}^{(1)}, d)$

▷ **Step 2 (Fit the generated cost model c to data C_{ij}^{*}):**

$c^{*} \leftarrow fit(c, C_{ij}^{*})$

▷ **Step 3: (Calculate regression score between data C_{ij}^{*} and cost model c^{*}):**

$s \leftarrow score(C_{ij}^{*}, c^{*})$

$R \leftarrow R \cup (s, c^{*})$

▷ **Step 4: (Select cost model with the highest score):**

$c_{ij} \leftarrow \max(R)$

return c_{ij}

6.3. CyberOpt-LSC Subalgorithm

This section describes the CyberOpt-LSC subalgorithm. The CyberOpt-LSC subalgorithm realizes Task 4, shown in Figure 17. The motivation of the CyberOpt-LSC subalgorithm is to learn sequences of control methods $\bar{S}$ from an event log of activities S^*. Manual creation of a behavior model is not required, reducing manual engineering effort and synchronizing the learned behavior model with the current state of the production system.

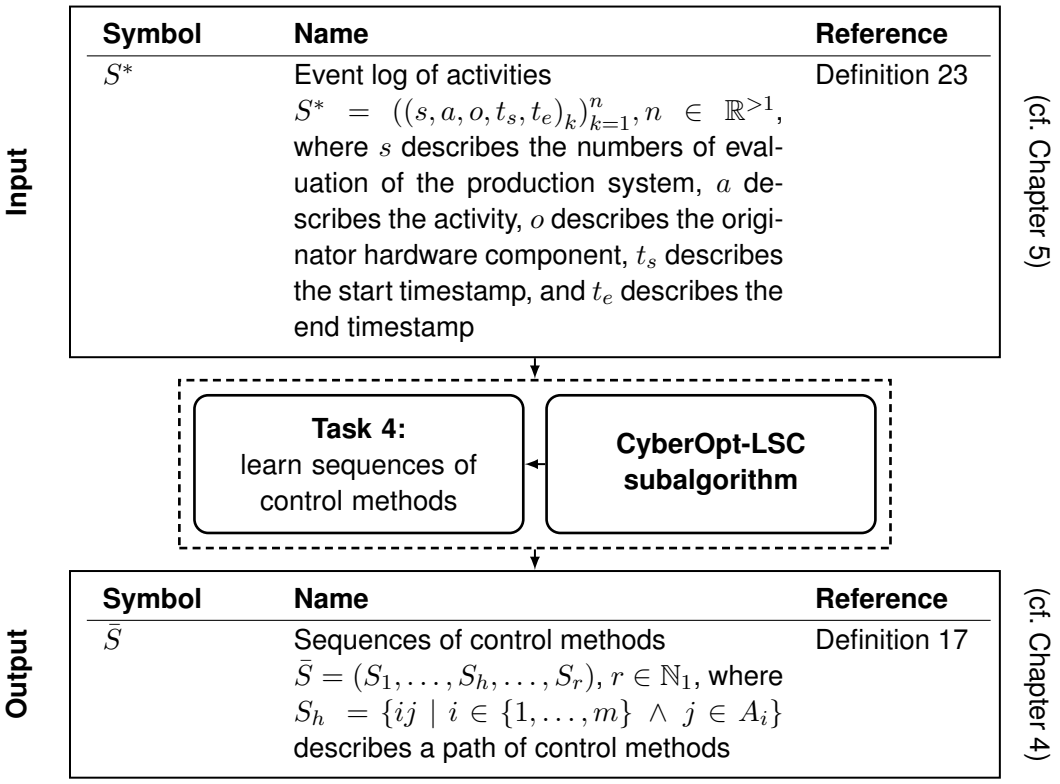


Figure 18 Task 4: Learn sequences of control methods.

In Section 3.3, behavior models are classified into automata-based behavior models and engineering behavior models. These behavior models can be learned by automata learning algorithms or process mining, respectively. Automata learning algorithms learn automata-based behavior models, and the process mining technique learns engineering behavior models. The HyBUTLA algorithm is introduced in [Nig+12]. The algorithm is capable of learning a hybrid timed automata model from synchronized signals and events. In the first step, a prefix-acceptor tree is created. A prefix-acceptor tree is a hybrid timed automaton in the form of a tree. In a prefix-acceptor tree, each sequence of observations results in a path from the root to a leaf. In a second step, the states compatible with the prefix acceptor tree are merged until a smaller automaton is reached that can still predict the system behavior. An algorithm named BUTLA is introduced in [Mai+11] that is able to learn a timed automata model from synchronized signals and events, and in [Mai14], an algorithm named OTALA is introduced that is able to learn online. The basic concept of process mining is to extract information about processes from transaction logs recorded by an information system [vdAal+03]. A transaction log contains events, and each event has a timestamp and refers to an activity and a

case, e.g., a process instance. An activity defines a specific step in the process. Control-flow mining algorithms, e.g., the α algorithm [vdAWM04], create a dependency graph (directed graph) directly from transaction logs. A graph with Petri net semantics is then created from the dependency graph. For example, if a transaction log contains an event E1 followed by an event E2, a relationship is created between a state E1 and a state E2 in the corresponding dependency graph. The problem with these algorithms is that even a single incorrect observation can completely disturb the derivation of a correct conclusion. To solve this problem, the heuristic miner algorithm introduced in [WvDD06] uses a frequency-based metric to calculate how certain a relationship between events is. For example, if the transaction log contains the observation 40 times that event E1 is followed by event E2, it is more likely that a relationship exists between E1 and E2 than if this observation was seen only once. Moreover, the fuzzy mining algorithm described in [GV07] implements adaptive simplifications based on data clustering and graph clustering, e.g., to solve the concurrency problem. The problem occurs when event E1 and event E2 can be observed in any order, e.g., they are on two distinct parallel paths, then the transaction log contains both possible cases: event E1 follows event E2 and vice versa. In this work, process mining is used to learn a behavior model. Section 3.3.3 describes a comparison of behavior models. Timed automata, hybrid timed automata, and priced timed automata are sequential behavior models. Therefore, sequential behavior models cannot be used to satisfy the requirement R5 of an optimal command signal configuration described in Section 1.2. An optimal command signal configuration should be able to incorporate knowledge about parallel sequences of control methods since production systems have many parallel activities.

The CyberOpt-LSC subalgorithm realizes Task 4 and is described in Algorithm 6. Given an event log of activities $S^* = ((s, a, o, t_s, t_e)_k)_{k=1}^n, n \in \mathbb{R}^{>1}$ and an automation software structure $\bar{A}$, the algorithm learns sequences of control methods $\bar{S} = (S_1, \ldots, S_h, \ldots, S_r), r \in \mathbb{N}_1$, where $S_h = \{ij \mid i \in \{1, \ldots, m\} \wedge j \in A_i\}$ describes a path of control methods. The CyberOpt-LSC subalgorithm consists of the following two steps:

Algorithm 6: Step 1 (Learn directed graph of activities): In the first step, the fuzzy miner presented in [GV07] is used to calculate a directed graph of activities from an event log of activities S^*, represented by function $mine$. The directed graph D of activities is defined as quadruple $D = (L, l_0, l_e, T)$. The finite set L describes the nodes of the directed graph of activities, where l_0 describes the start node and l_e describes the end node. $T \subseteq L \times L$ describes the transitions between nodes. Each node has an activity index i that refers to a software component A_i. The directed graph is only required for internal algorithm use to calculate sequences of control methods.

Algorithm 6: Step 2 (Calculate sequences of control): In the second step, a simple deep-first search algorithm and knowledge about the automation software structure $\bar{A}$ are used to calculate sequences of control methods $\bar{S}$, represented by the function $search$. The directed graph of activities is expanded to a directed graph of control methods. The directed graph of

control methods contains every possible path from the start node l_0 to the end node l_e.

Algorithm 6: The CyberOpt-LSC subalgorithm learns sequences of control methods $\bar{S}$ from an event log of activities S^*

Input : Event log S^*
: Automation software structure $\bar{A}$
Output : Sequences of control methods $\bar{S}$

1 $\bar{S} \leftarrow \emptyset$

▷ **Step 1: (Learn directed graph D of activities):**

2 $D \leftarrow mine(S^*)$

▷ **Step 2: (Calculate sequences of control):**

3 $\bar{S} \leftarrow search(D, \bar{A})$

4 **return** $\bar{S}$

6.4. CyberOpt-SPC Subalgorithm

This section describes the CyberOpt-SPC subalgorithm. The CyberOpt-SPC subalgorithm realizes Task 5, shown in Figure 17. The motivation of the CyberOpt-SPC subalgorithm is to automatically find optimal command signal configurations by solving the command signal configuration problem (see Definition 20).

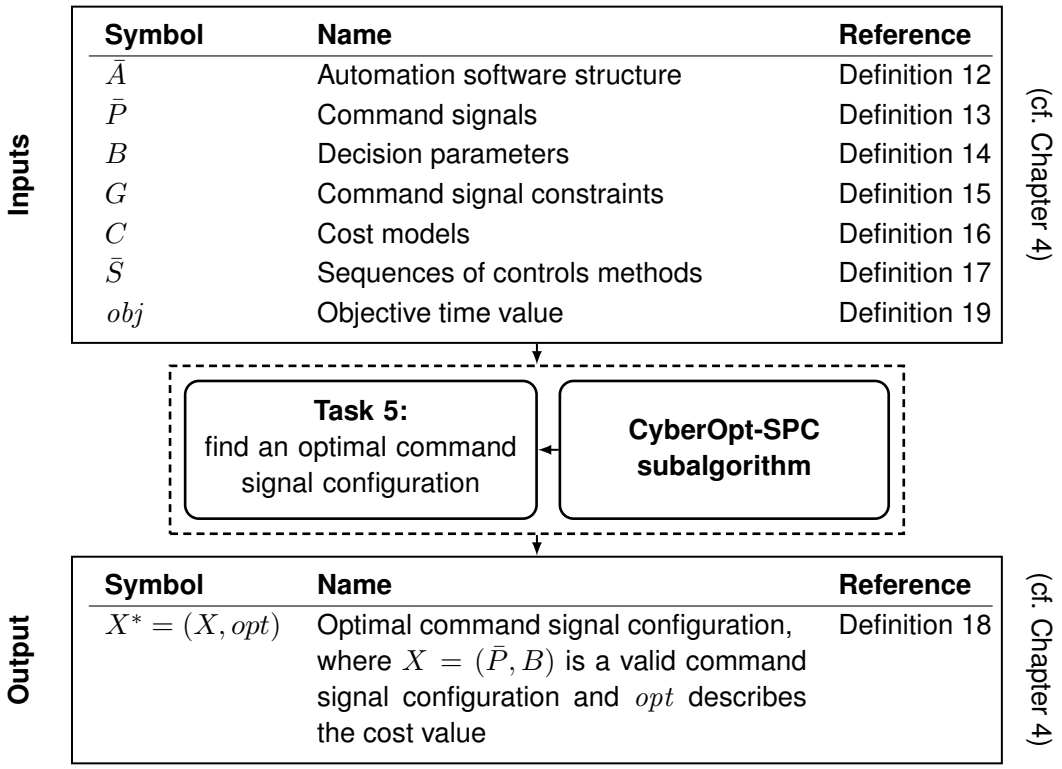


Figure 19 Task 5: Find an optimal command signal configuration.

A command signal configuration problem can be solved with (1) black-box optimization, (2) human-in-the-loop, (3) a sampling and validating approach, and (4) a general mixed-integer nonlinear programming solver. In the case of costly evaluations, e.g., physical simulations,

black-box optimization approaches can be used as a less costly alternative [HHL13, CN14, ON15]. The objective function is unknown, and an evaluation of the simulation model is performed for each command signal configuration. Model-based black-box optimization approaches construct a regression model, also named response surface model or surrogate model, that predicts the costs. The model is then used for optimization. Sequential model-based optimization, described in [HHL11], iterates between fitting a model and gathering additional data. It uses Gaussian process models [Ras06] to build a response surface model to predict which data needs to be evaluated. Gaussian processes are a generic supervised learning method used to solve, for example, regression problems. Command signal configurations that should be evaluated are calculated using an expected improvement criterion based on [JSW98] to improve the quality of surrogate models. Human-in-the-loop describes a process in which a domain expert selects a plausible command signal configuration according to his domain knowledge. For each software component A_i, a control method j must be selected, and timing parameter values and command signal values must be defined for each selected control method j. Sampling and validation describes a process in which a heuristic samples uniform random values to select a control method j for each component A_i and uniform random values for timing parameters and command signal for the selected control methods j. The sampled values are then validated by the command signal configuration problem constraints: objective timing constraint (see Equation 4.36) and command signal constraints (see Equation 5.4). The command signal configuration problem is defined as a mixed-integer nonlinear programming problem with a nonlinear objective function, linear constraints, and a mixture of continuous and binary variables (see Section 4.8). Section 3.2.2 describes various solving techniques for solving convex and nonconvex mixed-integer nonlinear programming problems. The different solving techniques are bundled by a solver described in Section 3.2.3.

This work uses general mixed-integer nonlinear programming solvers to solve the command signal configuration problem. In costly evaluations, black-box optimization approaches are useful because they iterate between fitting a model and gathering additional data to improve the quality of surrogate models that predict costs. Empirical results have shown that black-box approaches, compared to general mixed-integer nonlinear programming solvers, are not suitable for this type of combinatorial constraint optimization [OVN16, OVN18b]. Black-box optimization approaches have the problem of determining constraint valid command signal configurations. The human-in-the-loop technique contradicts the general idea of automating manual command signal configuration for automation software components in discrete manufacturing. In [OVN18b], we have shown that the probability of capturing a valid command signal configuration with a sampling and validating heuristic is near zero.

The CyberOpt-SPC subalgorithm is described in Algorithm 7. The command signal configuration problem is defined as a mixed-integer nonlinear programming problem with a nonlinear objective function, linear constraints, and a mixture of continuous and binary variables. The problem is defined in Algorithm 7: line 1 and function $solve(Problem)$ represents the calcula-

tion of an optimal command signal configuration $X^* = (X, opt)$. Any general mixed-integer nonlinear programming solver described in Section 3.2.3 can be used to solve the command signal configuration problem.

Algorithm 7: The CyberOpt-SPC subalgorithm calculates an optimal command signal configuration X^*

Input : Objective time value obj
: Automation software structure $\bar{A}$
: Command signals $\bar{P}$
: Decision parameters B
: Command signal constraints G
: Sequences of control methods $\bar{S}$
: Cost models C
Output : Optimal command signal configuration $X^* = (X, opt)$

1 $Problem$:

$$\text{minimize:} \quad \sum_{i \in \{1,\ldots,m\}} \sum_{j \in A_i} c_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)})\, b_{ij}, \qquad p_{ij}^{(1)} \in D_{ij}^{(1)}, \ldots, p_{ij}^{(k)} \in D_{ij}^{(k)}, b_{ij} \in \{0,1\} \tag{6.7}$$

$$\text{s.t.:} \quad \sum_{ij \in S_h} p_{ij}^{(1)}\, b_{ij} \leq obj, \qquad \forall\, h \in \{1,\ldots,r\}, p_{ij}^{(1)} \in D_{ij}^{(1)}, b_{ij} \in \{0,1\} \tag{6.8}$$

$$g_{ij}(p_{ij}^{(1)}, \ldots, p_{ij}^{(k)}) = 1, \qquad \forall\, i \in \{1,\ldots,m\}, j \in A_i, p_{ij}^{(1)} \in D_{ij}^{(1)}, \ldots, p_{ij}^{(k)} \in D_{ij}^{(k)} \tag{6.9}$$

$$\sum_{j \in A_i} b_{ij} = 1, \qquad \forall\, i \in \{1,\ldots,m\}, b_{ij} \in \{0,1\} \tag{6.10}$$

$X^* \leftarrow solve(Problem)$

2 **return** X^*

6.5. CyberOpt-ICM Algorithm

This section describes the CyberOpt-ICM subalgorithm. The CyberOpt-ICM subalgorithm realizes Task 20 shown in Figure 17. The motivation of the CyberOpt-ICM subalgorithm is to avoid exhaustive evaluation of all possible command signal configurations. An exhaustive evaluation of all possible command signal configurations is usually impossible since each evaluation of a command signal configuration takes time and is therefore expensive. Few evaluations shorten the time in which a production system does not operate optimally. Optimal in this work means energy efficient. The CyberOpt-ICM subalgorithm is to calculate a new valid command signal configuration $X = (\bar{P}, B)$, which should be evaluated in order to obtain more energy consumption values of the production system. With more energy consumption values, cost models are more accurate.

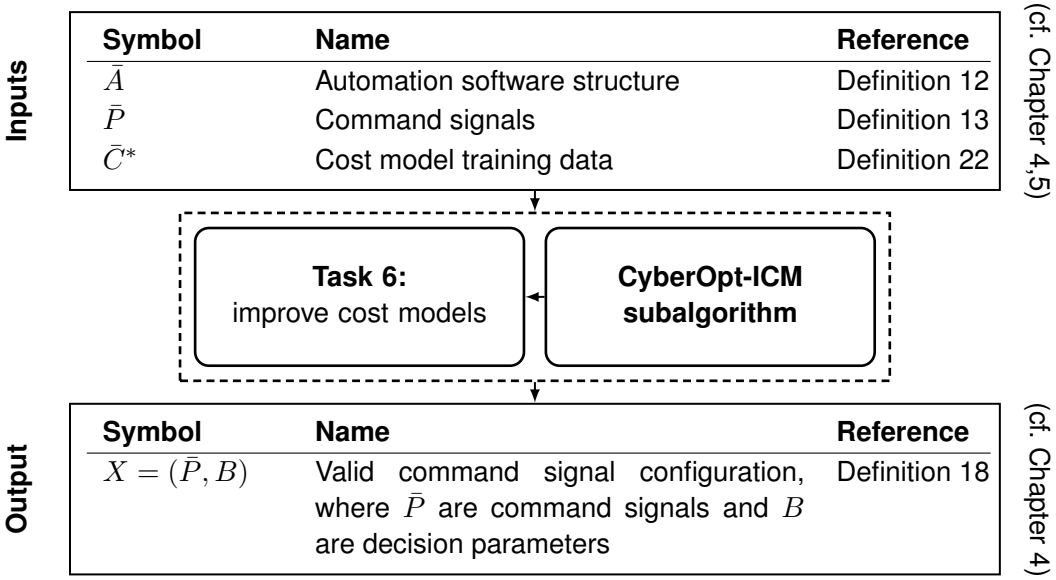


Figure 20 Task 6: Improve cost models.

In this work, the expected improvement (EI) criterion [JSW98] is used to calculate new valid command signal configurations. The EI criterion, described in Definition 26, calculates new command signals for a regression model. A regression model predicts cost values. The dependent variable is energy consumption, and the independent variables are command signals. The calculated command signals should be evaluated to improve the quality of the regression model because the dependent variable for these command signals (independent variables) is unknown. The idea is to calculate acquisition values using the EI criterion for a regression model. The command signal configuration with the highest acquisition value should be evaluated to improve the quality of the cost model c_{ij}.

Figure 21 illustrates the general idea. A reference cost function is known (see Figure 21: reference). The dependent variable is energy consumption (see Figure 21: energy consumption) and the independent variable is a timing parameter (see Figure 21: t [seconds]). In the first iteration, only two energy consumption values are known (Figure 21: evaluations). A

regression model is calculated for these two energy consumption values (Figure 21: predict). The acquisition values (Figure 21: EIC) are calculated for each command signal value between three and ten seconds. The command signal value with the highest acquisition value is selected. The real energy consumption value for this command signal value is calculated by the reference cost function. In the second iteration, three energy consumption values are known. A regression model is calculated for these three energy consumption values, and so on.

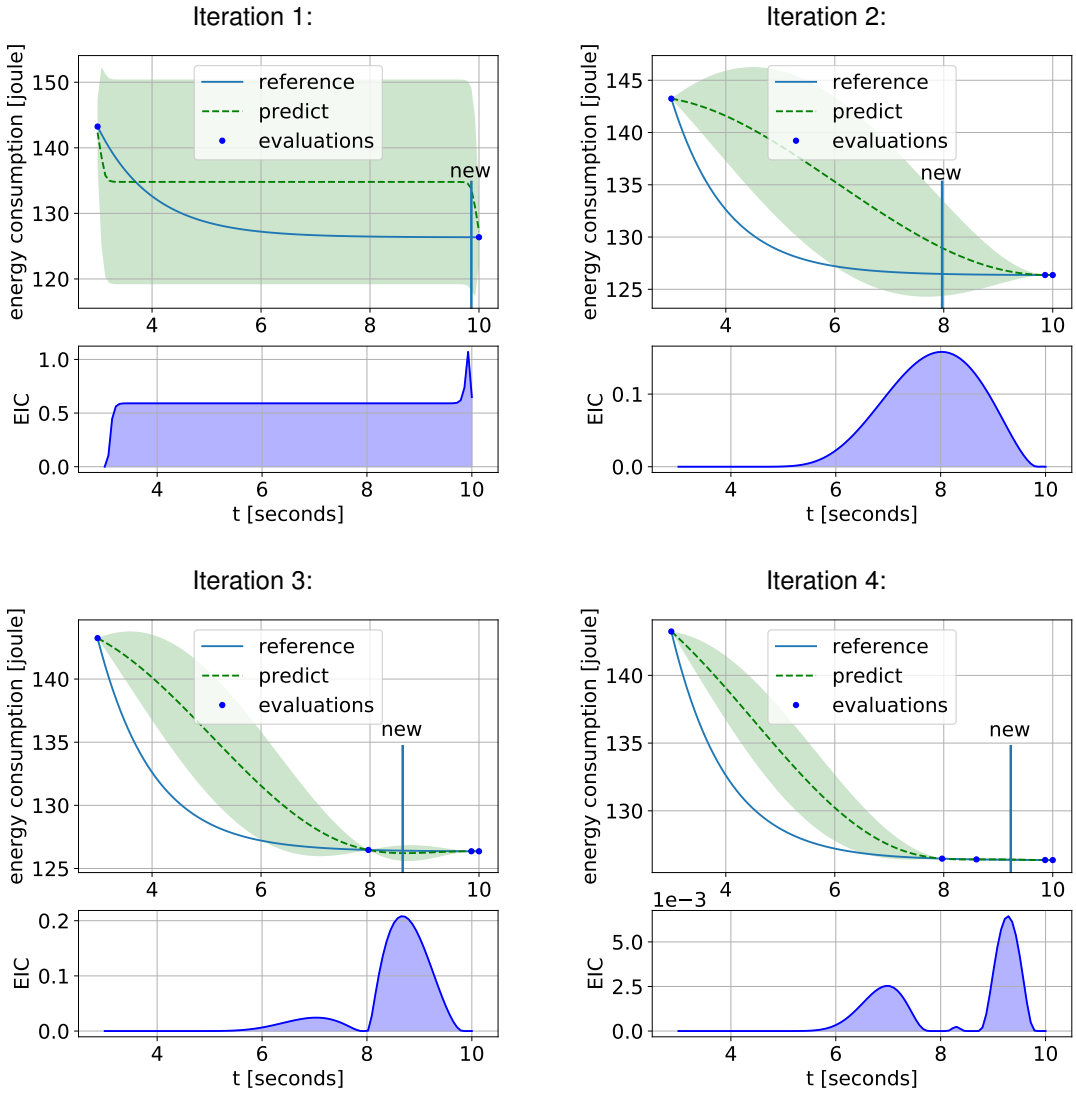


Figure 21 Expected improvement criterion example, four iterations that calculate new command signal configurations (adapted from [OVN18a]).

The expected improvement criterion requires the mean μ and the variance σ for each energy consumption value. The Gaussian process can be used to calculate these values, see Definition 27. The Gaussian process is a generic nonparametric supervised learning method [Ras06, BR13]. The Gaussian process is a generalization of the Gaussian distribution and describes a distribution over functions. It defines a collection of Gaussian distributed random variables. Random variables represent the values of a cost function c_{ij}. The ker-

nel matrix K that represents the collection of Gaussian distributed random variables is constructed by evaluating the covariance function between all pairs of inputs points (command signal configurations).

Definition 26 (Expected Improvement Criterion):

The expected improvement criterion function to calculate acquisition values (cdf is the standard normal cumulative distribution function, pdf is the standard normal probability density function, and c_{min} is the minimal cost value of all predicted cost values of a regression model):

$$EIC(c_{min}, \mu, \sigma) = (c_{min} - \mu)\ cdf(\frac{c_{min} - \mu}{\sigma}) + \sqrt{\sigma}\ pdf(\frac{c_{min} - \mu}{\sigma}) \tag{6.11}$$

Definition 27 (Gaussian Process Regression Model Predict):

Let P be training command signal configurations, $v_{1:n}$ be a vector of training cost values, σ^2 be the variance of measurement noise, P^* be the command signal configurations for which cost value should be predicted, the identity matrix I and the parameter l be the length scale of the kernel.

(i) **Predict function:**

$$GP(P, v_{1:n}, P^*, \sigma^2) = K(P^*, P)\left[K(P, P) + \sigma^2\, I\right]^{-1} v_{1:n} \tag{6.12}$$

(ii) **Kernel matrix:**

$$K = \begin{bmatrix} k(P_{ij}^{(1)}, P_{ij}^{(1)}) & \cdots & k(P_{ij}^{(1)}, P_{ij}^{(k)}) \\ & \ddots & \\ k(P_{ij}^{(k)}, P_{ij}^{(1)}) & \cdots & k(P_{ij}^{(k)}, P_{ij}^{(k)}) \end{bmatrix} \tag{6.13}$$

(iii) **Kernel function (squared exponential):**

$$k(P_{ij}, P_{ij}^*) = \sigma^2\ \exp\left(-\frac{\left\|P_{ij} - P_{ij}^*\right\|^2}{2\, l^2}\right) \tag{6.14}$$

The CyberOpt-ICM subalgorithm realizes Task 6 and is described in Algorithm 8. The algorithm consists of the following four steps:

Algorithm 8: Step 1 (Generate command signal configurations): In the first step, m command signal configurations are generated by a Latin hypercube design [San+]. Latin hypercube sampling can be used to perform computer experiments [Kan+15]. The algorithm

generates n command signal values from a d-dimensional hypercube. The continuous range for each command signal is partitioned into n equally spaced intervals. The command signal configuration values generated by this sampling method are distributed uniformly in the hypercube space [Che+06].

Algorithm 8: Step 2 (Fit and predict Gaussian process): In the second step, regression is used to fit a Gaussian process GP to the cost model training dataset C_{ij}^{*}. Then the cost mean values E_μ and cost variance values E_σ for the generated command signal configurations E_p are predicted using algorithm 2.1 from [RW06]. Fitting and prediction are represented by the function $GP(P, E_p, v_{1:n}, \sigma^2)$. Please note that fitting a Gaussian process can be computationally expensive. Inverting the matrix takes time $O(n^3)$.

Algorithm 8: Step 3 (Calculate expected improvement criterion): In the third step, an expected improvement criterion value is calculated for each generated command signal configuration.

Algorithm 8: Step 4 (Choose command signal configuration): In the last step, the command signal configuration with the highest expected improvement criterion value is nominated to be evaluated to enhance the quality of the cost model c_{ij}.

Algorithm 8: The CyberOpt-ICM subalgorithm calculates a new valid command signal configuration

---

Input : Cost model training dataset $C_{ij}^{*} = \{(P_{ij}^{(k)}, v^{(k)})\}_{k=1}^{n}, n \in \mathbb{R}^{>1}$
Output : Command signal values P_{ij}

▷ **Step 1 (Generate command signal configurations):**

1 $E_p \leftarrow lhd(m)$

▷ **Step 2 (Fit and predict Gaussian process):**

2 $E_\sigma, E_\mu = GP(P_{ij}^{(1)}, \ldots, P_{ij}^{(n)}, v^{(1)}, \ldots, v^{(n)}, E_p, \sigma^2)$; ▷ cost means E_μ, cost variances E_σ and variance of measurement noise σ^2

▷ **Step 3 (Calculate expected improvement criterion):**

3 $c_{min} = \min(E_\mu)$; ▷ minimum predicted energy consumption value

4 $ACQ \leftarrow \emptyset$; ▷ set of acquisition values

5 **for** $k \in \{1, \ldots m\}$ **do**

6 $\quad acq = (c_{min} - e_\mu^{(k)})\ cdf(\frac{c_{min} - e_\mu^{(k)}}{e_\sigma^{(k)}}) + \sqrt{e_\sigma^{(k)}}\ pdf(\frac{c_{min} - e_\mu^{(k)}}{e_\sigma^{(k)}})$

7 $\quad ACQ \leftarrow ACQ \cup (acq, E_p^{(k)})$

▷ **Step 4 (Choose command signal configuration):**

8 $P_{ij} \leftarrow \max(ACQ)$

9 **return** P_{ij}

---

7. Description of Scenarios and Extensions

This chapter describes in detail three common application scenarios and their synthetic extensions. This chapter is based on previous work by the author and extends the published scenarios [ON15, OVN16, OVN18b, OVN18a]. The application scenarios, briefly introduced in Section 1.2 to derive requirements for an optimal command signal configuration, are used to systematically analyze the introduced CyberOpt approach in Chapter 8.

The application scenarios show that CyberOpt can be used for applications in discrete manufacturing. There are three ways to show that an algorithm can solve different classes of problems:

Data from real-world applications: Data from real-world applications are good for showing that an algorithm is capable of solving real-world problems and are not just so-called “toy models.” However, the problem of using data from real applications is that it is impossible to show that these data cover all possible cases, e.g., hardware combinations in discrete manufacturing.

Synthetic data: Synthetic data is good for showing that an algorithm is capable of handling various problems and the expected results are known. However, the problem of using synthetic data is that they could be seen as so-called “toy models.” There is a lack of belief that these data correspond to real-world applications.

Synthetic extensions: In this work, synthetic extensions are introduced. The idea is to extend common real-world applications with synthetic data to overcome the above-described drawbacks of real-world applications and synthetic data.

Section 7.1 describes the transport scenario. The transport scenario describes a process for transporting a product between locations, such as transporting a product between different production modules or between production modules and storage solutions. Section 7.2 describes the storage scenario. The storage scenario describes a process to stock products, e.g., produced products that need to be stored temporarily. Section 7.3 describes the pick-and-place scenario. The pick-and-place scenario describes a process for picking and placing products, e.g., to pick a product from a conveyor system and place it in a production module. The automation software structure, the command signals, the command signal constraints, and the sequences of control methods are described for each scenario. Section 7.4 describes the synthetic extensions.

7.1. Transport Scenario

This section describes the transport scenario. Transport scenarios are used to transport products between locations, e.g., between different production modules or between production modules and storage solutions. The goal of the transport scenario, shown in Figure 22, is to move a product from position A to position B. The automation software controls a conveyor system, represented by hardware component M_1.

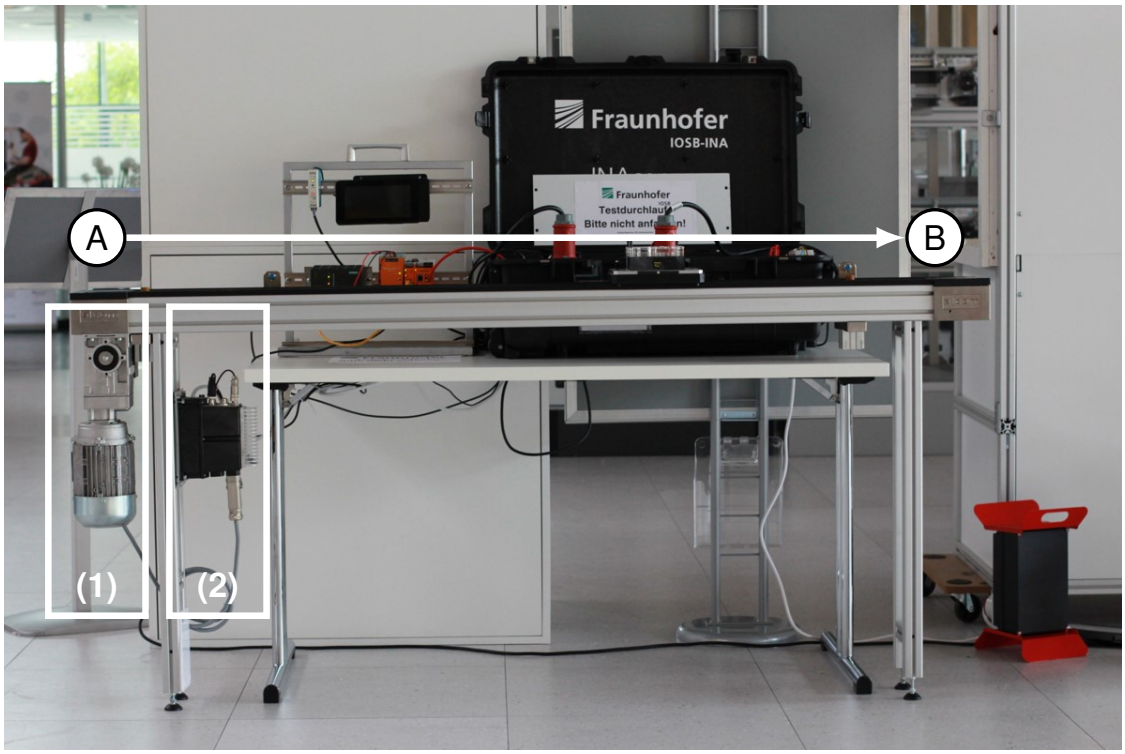


Figure 22 A conveyor system that realizes the transport scenario: M_1 labels the hardware component, A and B label predefined positions, (1) labels the electrical motor and (2) labels the frequency converter.

The hardware component M_1 consists of five automation components: an electric motor, a frequency converter, a conveyor belt, a light barrier, and a programmable logic controller. The electrical motor is controlled by the frequency converter. The frequency converter is controlled by a control strategy implementation that runs on the programmable logic controller. A product is moved from position A to position B by the conveyor belt during a single drive. The distance d between position A and position B is $120cm$.

The automation software structure of the transport scenario is described in Example 15. The automation software structure of the conveyor hardware component M_1 has a software component A_1 with a control strategy $move\text{-}B(P_{11})$, cf. Equation 7.1, Equation 7.2 and Equation 7.3.

Example 15 (Automation Software Structure):

The automation software structure of the transportation scenario:

SWC: $\boldsymbol{A_1}$
(1) $move\text{-}B(P_{11})$

$$\bar{A} = (A_1), \qquad m = 1 \tag{7.1}$$

$$A_1 = \{1\}, \qquad n_1 = 1 \tag{7.2}$$

$$n = n_1 = 1 \tag{7.3}$$

The frequency converter used has tree different modes: (1) an acceleration mode, (2) a constant mode, and (3) a deceleration mode, illustrated in Figure 23. The control method $move\text{-}B(P_{11})$ implements the logic to switch between these modes. The frequency converter works with angles and not with distances. The area under the curve describes the total angle $\phi = \phi_a + \phi_c + \phi_c$. When the product reaches the light barrier at position B, the motor stops.

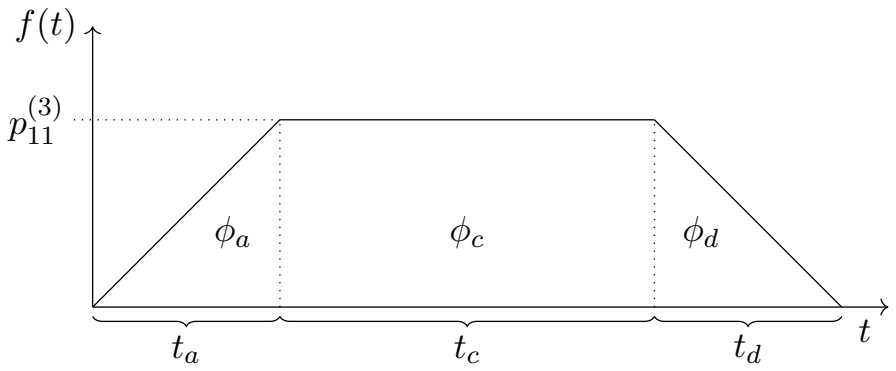


Figure 23 Acceleration and deceleration ramp: t_a describes the time in the acceleration mode, t_c describes the time in the constant mode and t_d describes the time in the deceleration mode.

The command signals of the transport scenario are described in Example 16 and summarized in Table 13.

Command Signals	Description	Minimum	Maximum	Unit
$p_{11}^{(1)}$	time	7.7	26.25	s
$p_{11}^{(2)}$	acceleration	1	20	Hz/s
$p_{11}^{(3)}$	constant	10	50	Hz
$p_{11}^{(4)}$	deceleration	1	20	Hz/s

Table 13 Description of timing parameter $p_{11}^{(1)}$ and command signals $p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)}$

The control strategy $move\text{-}B(P_{11})$ implements the logic to switch between acceleration mode, constant mode, and deceleration mode. It has a timing parameter $p_{11}^{(1)}$ and three command signals $p_{11}^{(2)}$, $p_{11}^{(3)}$, $p_{11}^{(4)}$, see Equation 7.5. Each mode of the frequency converter is described by a command signal. The acceleration mode is controlled by command signal $p_{11}^{(2)} \in$

$[1Hz/s, 20Hz/s]$, the constant mode is controlled by command signal $p_{11}^{(3)} \in [10Hz, 50Hz]$, and the deceleration mode is controlled by command signal $p_{11}^{(4)} \in [1Hz/s, 20Hz/s]$.

Example 16 (Command Signals):

The command signals of the transportation scenario:

SWC: A_1
(1) $move\text{-}B(\boldsymbol{P_{11}})$

$$\bar{P} = (P_{11}) \tag{7.4}$$

$$P_{11} = (p_{11}^{(1)}, p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)}) \tag{7.5}$$

$$D_{11}^{(1)} = \mathbb{R}^{>0} \tag{7.6}$$

The control strategy $move\text{-}D1\,(P_{31})$ calculates from timing parameters $p_{11}^{(1)}$, $p_{11}^{(2)}$ and $p_{11}^{(3)}$ and command signals t_a, t_c and t_d. The command signals are used to switch between the different modes. The command signal t_a describes the time of the acceleration phase, the command signal t_c describes the time of the constant phase, and the command signal t_d describes the time of the deceleration phase.

Equation 7.9 - Equation 7.11 describes the geometric derivation of command signal t_c, which is shown in Figure 23. Equation 7.12 describes the calculation from timing parameters $p_{11}^{(1)}$, $p_{11}^{(2)}$, $p_{11}^{(3)}$, and $p_{11}^{(4)}$ to the command signal t_c. Equation 7.8 describes the calculation from $p_{11}^{(2)}$ and $p_{11}^{(3)}$ to t_a. Equation 7.13 describes the calculation from $p_{11}^{(3)}$ and $p_{11}^{(4)}$ to t_d. The timing parameter $p_{11}^{(1)}$ describes the time to perform the activity $move$ and is described in Equation 7.7.

(i) **Timing parameter** $p_{11}^{(1)}$:

$$p_{11}^{(1)} = t_a + t_c + t_d \tag{7.7}$$

(ii) **Command signal** t_a:

$$t_a = \frac{p_{11}^{(3)}}{p_{11}^{(2)}} \quad \text{maximal frequency / acceleration} \tag{7.8}$$

(iii) **Command signal** t_c:

$$\phi = \frac{1}{2}\, p_{11}^{(3)}\, t_a + p_{11}^{(3)}\, t_c + \frac{1}{2}\, p_{11}^{(3)}\, t_d \tag{7.9}$$

$$\phi - \frac{1}{2}\, p_{11}^{(3)}\, t_a - \frac{1}{2}\, p_{11}^{(3)}\, t_d = p_{11}^{(3)}\, t_c \tag{7.10}$$

$$\frac{\phi}{p_{11}^{(3)}} - \frac{1}{2}\, t_a - \frac{1}{2}\, t_d = t_c \tag{7.11}$$

$$t_c = \frac{\phi}{p_{11}^{(3)}} - \frac{1}{2}\,(t_a + t_d) \tag{7.12}$$

(iv) **Command signal** t_d:

$$t_d = \frac{p_{11}^{(3)}}{p_{11}^{(4)}} \quad \text{maximal frequency / deceleration} \tag{7.13}$$

The command signal constraints of the transport scenario are described in Example 17. The command signal constraint function $g_{21}(p_{11}^{(1)}, p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)})$ defines a timing constraint and command signal constraints, see Equation 7.15. The constraint $p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)} > 0$ ensures that every mode is used.

Example 17 (Command Signal Constraints):

The command signal constraints of the transportation scenario:

$$G = (g_{11}) \tag{7.14}$$

$$g_{21}(p_{11}^{(1)}, p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)}) = \begin{cases} 1 & 7.7s \leq p_{11}^{(1)} \leq 26.25s \\ 1 & 1Hz/s \leq p_{11}^{(2)} \leq 20Hz/s \\ 1 & 10Hz \leq p_{11}^{(3)} \leq 50Hz \\ 1 & 1Hz/s \leq p_{11}^{(4)} \leq 20Hz/s \\ 1 & p_{11}^{(2)}, p_{11}^{(3)}, p_{11}^{(4)} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{7.15}$$

The transport scenario has only one sequence of control methods $\bar{S} = ((11)_1)$.

7.2. Storage Scenario

This section describes the storage scenario. Storage scenarios are used to stock products, e.g., produced products that need to be stored temporarily. The modular storage system,

shown in Figure 24, consists of four hardware components $M_1 - M_4$ and realizes two transports: (1) from position G to position A, via position F and position C, or via position F, position D, and position C.

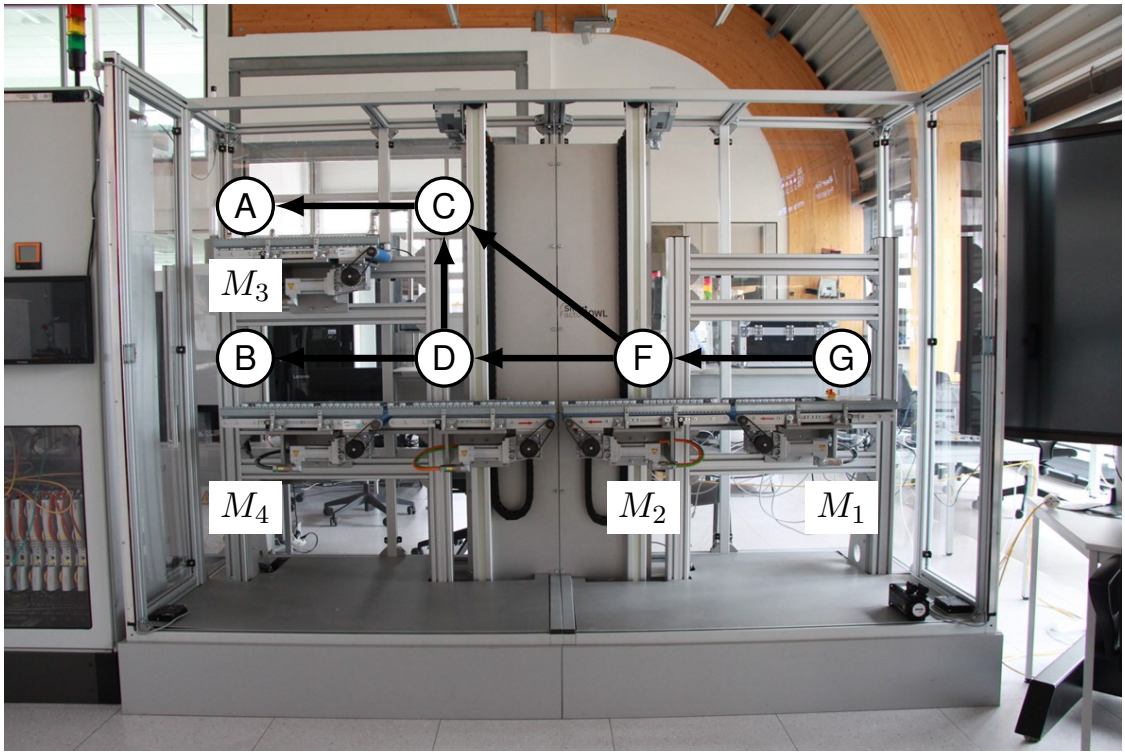


Figure 24 Modular storage system that realizes a storage scenario: $M_1 - M_3$ are transport components and $A - G$ are predefined positions (adapted from [OVN18b, OVN18a]).

The hardware component M_1, hardware component M_3, and hardware component M_4 are capable of moving horizontally from position G to position F, from position C to position A, and from position D to position B. The hardware component M_2 can move vertically from position D to position C, or from position F to position C, as well as horizontally from position F to position D. Each component has a programmable logic controller connected to a central programmable logic controller.

The automation software structure of the storage scenario is described in Example 18. The automation software structure of the modular storage system consists of three software components A_1, A_2, and A_3, see Equation 7.16, and four control methods, see Equation 7.20. The software component A_1 has one control method $drive\text{-}F(P_{11})$ that implements the movement from position G to position F, see Equation 7.17. The software component A_2 has two control methods $drive\text{-}C1(P_{21})$ and $drive\text{-}C2(P_{22})$, see Equation 7.18. The control method $drive\text{-}C1(P_{21})$ implements the movement from position F to position C via position D, and the control method $drive\text{-}C2(P_{22})$ implements the direct movement from position C to position D. The software component A_3 has one control method $drive\text{-}A(P_{31})$ that implements the movement from position C to position A, see Equation 7.19.

Example 18 (Automation Software Structure):

The automation software structure of the storage scenario:

SWC: $\boldsymbol{A_1}$
(1) $drive\text{-}F(P_{11})$

SWC: $\boldsymbol{A_2}$
(1) $drive\text{-}C1(P_{21})$
(2) $drive\text{-}C2(P_{22})$

SWC: $\boldsymbol{A_3}$
(1) $drive\text{-}A(P_{31})$

$$\bar{A} = (A_1, A_2, A_3), \qquad m = 3 \tag{7.16}$$
$$A_1 = \{1\}, \qquad n_1 = 1 \tag{7.17}$$
$$A_2 = \{1, 2\}, \qquad n_2 = 2 \tag{7.18}$$
$$A_3 = \{1\}, \qquad n_3 = 1 \tag{7.19}$$
$$n = n_1 + n_2 + n_3 = 4 \tag{7.20}$$

The command signals of the storage scenario are described in Example 19. The control methods $drive\text{-}F(P_{11})$, $drive\text{-}C1(P_{21})$, $drive\text{-}C2(P_{22})$, and $drive\text{-}A(P_{31})$ use the same model predictive control approach that is described in [WNS15]. The model predictive control approach minimizes energy consumption by adjusting movements of electric drives during runtime to find optimal sequences of movement parameters so that all control strategies have only one timing parameter, see Equation 7.22 - Equation 7.25.

Example 19 (Command Signals):

The command signals of the storage scenario:

SWC: A_1
(1) $drive\text{-}F(\boldsymbol{P_{11}})$

SWC: A_2
(1) $drive\text{-}C1(\boldsymbol{P_{21}})$
(2) $drive\text{-}C2(\boldsymbol{P_{22}})$

SWC: A_3
(1) $drive\text{-}A(\boldsymbol{P_{31}})$

$$\bar{P} = (P_{11}, P_{21}, P_{22}, P_{31}) \tag{7.21}$$
$$P_{11} = (p_{11}^{(1)}) \tag{7.22}$$
$$P_{21} = (p_{21}^{(1)}) \tag{7.23}$$
$$P_{22} = (p_{22}^{(1)}) \tag{7.24}$$
$$P_{31} = (p_{31}^{(1)}) \tag{7.25}$$
$$D_{11}^{(1)}, D_{21}^{(1)}, D_{22}^{(1)}, D_{31}^{(1)} = \mathbb{R}^{>0} \tag{7.26}$$

The command signal constraints of the transport scenario are described in Example 20. For each control method $drive\text{-}F(P_{11})$, $drive\text{-}C1(P_{21})$, $drive\text{-}C2(P_{22})$, and $drive\text{-}A(P_{31})$, a command signal constraint function is defined $G = (g_{11}, g_{21}, g_{22}, g_{31})$, see Equation 7.27. Each constraint function, see Equation 7.28 - Equation 7.31, describes a timing constraint.

Example 20 (Command Signal Constraints):

The command signal constraints of the storage scenario:

$$G = (g_{11}, g_{21}, g_{22}, g_{31}) \tag{7.27}$$

$$g_{11}(p_{11}^{(1)}) = \begin{cases} 1 & 3s \leq p_{11}^{(1)} \leq 10s \\ 0 & \text{otherwise.} \end{cases} \tag{7.28}$$

$$g_{21}(p_{21}^{(1)}) = \begin{cases} 1 & 2s \leq p_{21}^{(1)} \leq 6s \\ 0 & \text{otherwise.} \end{cases} \tag{7.29}$$

$$g_{22}(p_{22}^{(1)}) = \begin{cases} 1 & 6s \leq p_{22}^{(1)} \leq 10s \\ 0 & \text{otherwise.} \end{cases} \tag{7.30}$$

$$g_{31}(p_{31}^{(1)}) = \begin{cases} 1 & 3s \leq p_{31}^{(1)} \leq 10s \\ 0 & \text{otherwise.} \end{cases} \tag{7.31}$$

The activity diagram, shown in Figure 25, describes software components and control methods required to achieve the goal of the storage scenario.

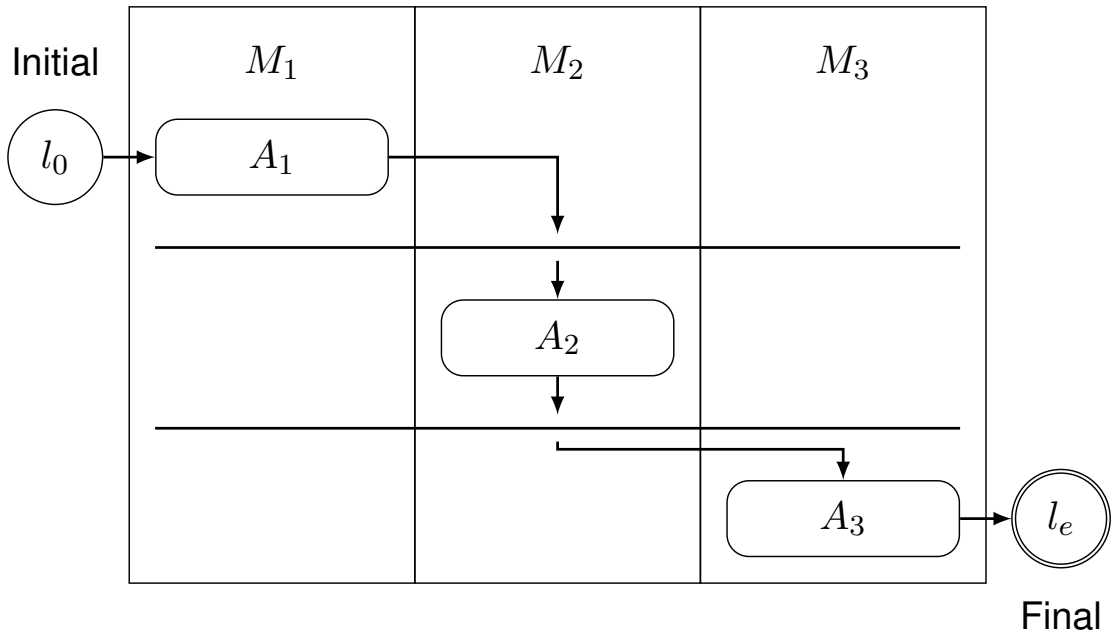


Figure 25 Activity diagram of the storage scenario (adapted from [OVN18b, OVN18a]).

The sequences of control methods $\bar{S}$ are defined as:

$$\bar{S} = (S_1, S_2) \tag{7.32}$$

$$S_1 = \{11, 21, 31\} \tag{7.33}$$

$$S_2 = \{11, 22, 31\} \tag{7.34}$$

7.3. Pick-and-Place Scenario

This section describes the pick-and-place scenario. Pick-and-place scenarios are used to pick and place products, e.g., to pick a product from a conveyor system and place it in a production module. The goal of the process step pick-and-place, shown in Figure 26, is to take a customer product from position B and place it in a box at position C.

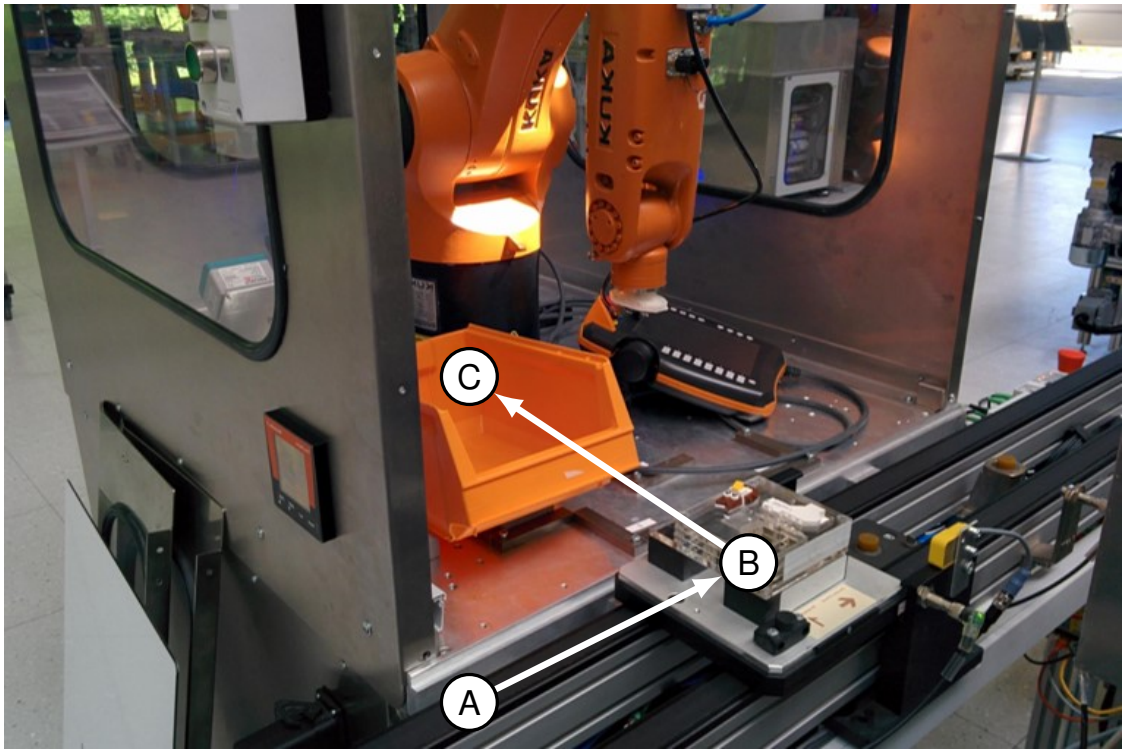


Figure 26 A robot system that realizes the pick-and-place scenario (adapted from [OVN18b]).

The automation software controls three heterogeneous hardware components: a conveyor system M_1, a robot M_2, and a compressor M_3. The robot is equipped with a vacuum gripper. The conveyor system consists of transportation components. Each transport component has a frequency converter to adjust the speed continuously. The compressor generates pressure, which is buffered by a pressure accumulator with a volume of 350 liters. Each hardware component has a programmable logic controller for implementing reusable software components. Each controller is connected to a central programmable logic controller via the real-time communication protocol PROFINET, which implements the automation software that controls the hardware components $M_1 - M_3$.

The automation software structure of the pick-and-place scenario is described in Example 21. The automation software structure of the modular storage system has seven software components A_1, A_2, A_3, A_4, A_5, A_6, and A_7, see Equation 7.35, and eight control methods, see Equation 7.43. The software component A_1 has a control method $drive\text{-}B(P_{11})$ that implements the movement of the conveyor hardware component M_1 to position B, see Equation 7.36. The software component A_2 has a control method $drive\text{-}B(P_{21})$ that implements

the movement of the robot hardware component M_2 to position F, see Equation 7.37. The software component A_3 has a control method $produce\text{-}P(P_{31})$ that implements the logic of the compressor hardware component M_3 to generate compressed air, see Equation 7.38. The software component A_4 has a control method $take\text{-}B(P_{41})$ that implements the logic of picking the hardware component M_2 at position B, see Equation 7.39. The software component A_5 has a control method $drive\text{-}A(P_{51})$ that implements the movement of the hardware component M_1 M_1 from position B to position A, see Equation 7.40. The software component A_6 has two control methods $drive\text{-}C1(P_{61})$ and $drive\text{-}C2(P_{62})$. They implement the movement of the hardware component M_2 from position B to position C, see Equation 7.41. The software component A_7 has a control method $drive\text{-}F(P_{11})$ that implements the logic of placing the hardware component M_2 at position C, see Equation 7.42.

Example 21 (Automation Software Structure):

The automation software structure of the pick-and-place scenario:

SWC: $\boldsymbol{A_1}$
(1) $drive\text{-}B(P_{11})$

SWC: $\boldsymbol{A_2}$
(1) $drive\text{-}B(P_{21})$

SWC: $\boldsymbol{A_3}$
(1) $produce\text{-}P(P_{31})$

SWC: $\boldsymbol{A_4}$
(1) $take\text{-}B(P_{41})$

SWC: $\boldsymbol{A_5}$
(1) $drive\text{-}A(P_{51})$

SWC: $\boldsymbol{A_6}$
(1) $drive\text{-}C1(P_{61})$
(2) $drive\text{-}C2(P_{62})$

SWC: $\boldsymbol{A_7}$
(1) $drop\text{-}C(P_{71})$

$$\bar{A} = (A_1, A_2, A_3, A_4, A_5, A_6, A_7), \qquad m = 7 \tag{7.35}$$
$$A_1 = \{1\}, \qquad n_1 = 1 \tag{7.36}$$
$$A_2 = \{1\}, \qquad n_2 = 1 \tag{7.37}$$
$$A_3 = \{1\}, \qquad n_3 = 1 \tag{7.38}$$
$$A_4 = \{1\}, \qquad n_4 = 1 \tag{7.39}$$
$$A_5 = \{1\}, \qquad n_5 = 1 \tag{7.40}$$
$$A_6 = \{1, 2\}, \qquad n_6 = 2 \tag{7.41}$$
$$A_7 = \{1\}, \qquad n_7 = 1 \tag{7.42}$$
$$n = n_1 + n_2 + n_3 + n_4 + n_5 + n_6 + n7 = 8 \tag{7.43}$$

The command signals of the pick-and-place scenario are described in Example 22. The control methods $drive\text{-}B(P_{11})$ and $drive\text{-}A(P_{51})$ use the model predictive control approach described in [WNS15]. They have only one timing parameter, see Equations 7.45 and Equation 7.49. The robot hardware component M_2 has a vendor-specific black box control strategy implementation. The control methods $drive\text{-}B(P_{21})$, $take\text{-}B(P_{41})$, $drive\text{-}C1(P_{61})$, $drive\text{-}C2(P_{62})$, and $drop\text{-}C(P_{72})$ have one timing parameter, see Equation 7.46, 7.48, 7.50, 7.51, and 7.52. The compressor hardware component M_3 also has a vendor-specific black box control strategy implementation. The control method $produce\text{-}P(P_{31})$ has one timing

parameter, see Equation 7.49.

Example 22 (Command Signals):

Automation software structure

SWC: A_1	SWC: A_2	SWC: A_3
(1) $drive\text{-}B1(\boldsymbol{P_{11}})$	(1) $drive\text{-}B1(\boldsymbol{P_{21}})$	(1) $produce\text{-}P1(\boldsymbol{P_{31}})$

SWC: $\boldsymbol{A_4}$	SWC: A_5	SWC: A_6	SWC: A_7
(1) $take\text{-}B1(\boldsymbol{P_{41}})$	(1) $drive\text{-}A1(\boldsymbol{P_{51}})$	(1) $drive\text{-}C1(\boldsymbol{P_{61}})$ (2) $drive\text{-}C2(\boldsymbol{P_{62}})$	(1) $drop\text{-}C1(\boldsymbol{P_{71}})$

The command signals of the example production system:

$$\bar{P} = (P_{11}, P_{21}, P_{31}, P_{41}, P_{51}, P_{61}, P_{62}, P_{71}) \tag{7.44}$$

$$P_{11} = (p_{11}^{(1)}) \tag{7.45}$$

$$P_{21} = (p_{21}^{(1)}) \tag{7.46}$$

$$P_{31} = (p_{31}^{(1)}) \tag{7.47}$$

$$P_{41} = (p_{41}^{(1)}) \tag{7.48}$$

$$P_{51} = (p_{51}^{(1)}) \tag{7.49}$$

$$P_{61} = (p_{61}^{(1)}) \tag{7.50}$$

$$P_{62} = (p_{62}^{(1)}) \tag{7.51}$$

$$P_{71} = (p_{71}^{(1)}) \tag{7.52}$$

$$D_{11}^{(1)}, D_{21}^{(1)}, D_{31}^{(1)}, D_{41}^{(1)} = \mathbb{R}^{>0} \tag{7.53}$$

$$D_{51}^{(1)}, D_{61}^{(1)}, D_{62}^{(1)}, D_{71}^{(1)} = \mathbb{R}^{>0} \tag{7.54}$$

The command signal constraints of the pick-and-place scenario are described in Example 23. For each control method $drive\text{-}B1(P_{11})$, $drive\text{-}B1(P_{21})$, $produce\text{-}P1(P_{31})$, $take\text{-}B1(P_{41})$, $drive\text{-}A1(P_{51})$, $drive\text{-}C1(P_{61})$, $drive\text{-}C2(P_{62})$, and $drop\text{-}C1(P_{71})$ a command signal constraint function is defined $G = (g_{11}, g_{21}, g_{31}, g_{41}, g_{51}, g_{61}, g_{62}, g_{71})$, see Equation 7.55. Each constraint function, see Equation 7.56 - Equation 7.63, describes a timing constraint.

Example 23 (Command Signal Constraints):

Command signal constraints of the pick-and-place scenario:

$$G = (g_{11}, g_{21}, g_{31}, g_{41}, g_{51}, g_{61}, g_{62}, g_{71}) \tag{7.55}$$

$$g_{11}(p_{11}^{(1)}) = \begin{cases} 1 & 3s \leq p_{11}^{(1)} \leq 10s \\ 0 & \text{otherwise.} \end{cases} \tag{7.56}$$

$$g_{21}(p_{21}^{(1)}) = \begin{cases} 1 & 5s \leq p_{21}^{(1)} \leq 10s \\ 0 & \text{otherwise.} \end{cases} \tag{7.57}$$

$$g_{31}(p_{31}^{(1)}) = \begin{cases} 1 & 6s \leq p_{31}^{(1)} \leq 12s \\ 0 & \text{otherwise.} \end{cases} \tag{7.58}$$

$$g_{41}(p_{41}^{(1)}) = \begin{cases} 1 & 2s \leq p_{41}^{(1)} \leq 6s \\ 0 & \text{otherwise.} \end{cases} \tag{7.59}$$

$$g_{51}(p_{51}^{(1)}) = \begin{cases} 1 & 3s \leq p_{51}^{(1)} \leq 10s \\ 0 & \text{otherwise.} \end{cases} \tag{7.60}$$

$$g_{61}(p_{61}^{(1)}) = \begin{cases} 1 & 2s \leq p_{61}^{(1)} \leq 8s \\ 0 & \text{otherwise.} \end{cases} \tag{7.61}$$

$$g_{62}(p_{62}^{(1)}) = \begin{cases} 1 & 8s \leq p_{62}^{(1)} \leq 10s \\ 0 & \text{otherwise.} \end{cases} \tag{7.62}$$

$$g_{71}(p_{71}^{(1)}) = \begin{cases} 1 & 2s \leq p_{71}^{(1)} \leq 6s \\ 0 & \text{otherwise.} \end{cases} \tag{7.63}$$

The activity diagram, shown in Figure 27, describes software components and control methods required to achieve the goal of the pick-and-place scenario.

The sequences of control methods $\bar{S}$ are defined as:

$$\bar{S} = (S_1, S_2) \tag{7.64}$$

$$S_1 = \{11, 21, 31, 41, 51, 61, 71\} \tag{7.65}$$

$$S_2 = \{11, 21, 31, 41, 51, 62, 71\} \tag{7.66}$$

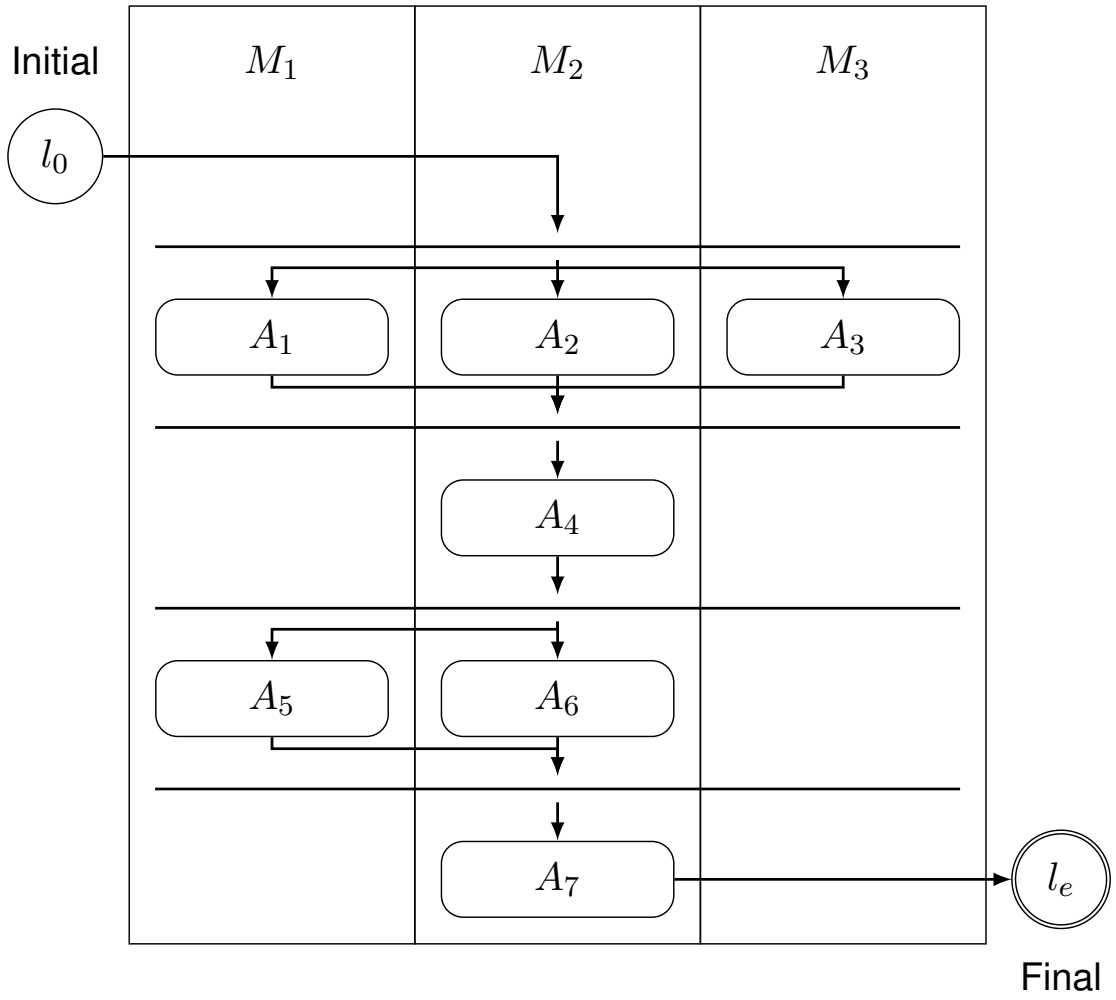


Figure 27 Activity diagram of the pick-and-place scenario (adapted from [OVN18b]).

7.4. Synthetic Extensions

This section describes synthetic extensions of the common application scenarios described above. The idea is to extend the common application scenarios with synthetic data to eliminate the disadvantage of data from real-world applications and synthetic data.

Reference Cost Functions: The reference cost functions defined in Section 7.4.1 extend the transport scenario. The evaluation can therefore be carried out with four reference functions rather than just one function. The defined reference cost functions f_{ackley}, $f_{rastrigin}$, f_{sphere}, and $f_{griewangk}$ are well-known benchmark functions in the domain of optimization.

Behavior Model Generator: The behavior models of the storage scenario and pick-and-place scenario are relatively "small." Therefore, a behavior model generator is defined to generate "larger" behavior models.

7.4.1. Extension 1: Reference Cost Functions

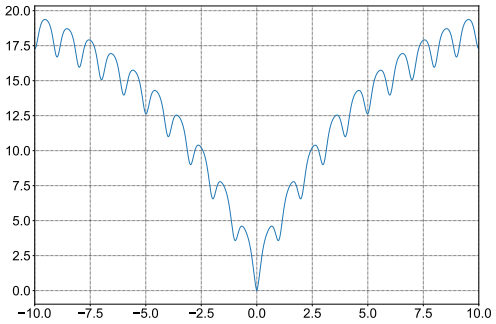
This section describes four canonical 10-dimensional reference cost functions. For each of these cost functions, the global optimum is $x^* = [0, \ldots, 0]$ with function value $f(x^*) = 0$.

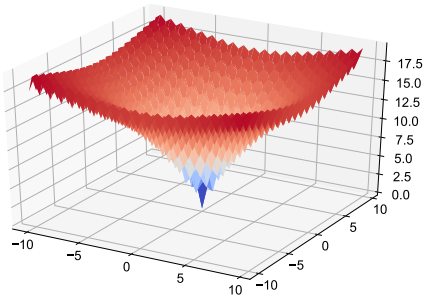
Ackley Reference Cost Function:

Definition 28 defines the f_{ackley} reference cost function and illustrates the reference cost function in 1D and 2D.

Definition 28 (Ackley Reference Cost Function):

$$f_{ackley}(x) = 20 - 20 * \exp\left[-0.2\sqrt{\sum_{i=1}^{n}\frac{x_i^2}{n}}\right] + \exp(1) - \exp\left[\sum_{i=1}^{n}\frac{\cos(2\pi x_i)}{n}\right]$$



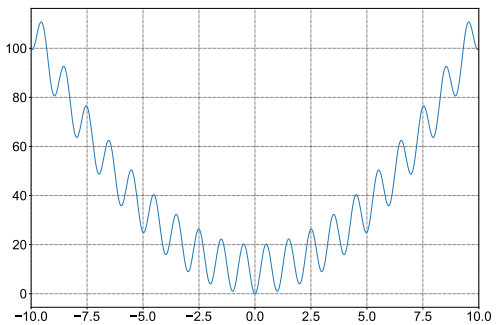


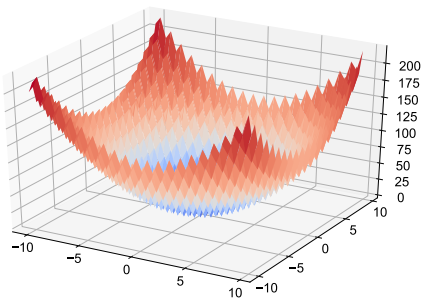
Rastrigin Reference Cost Function:

Definition 29 defines the $f_{rastrigin}$ reference cost function and illustrates the reference cost function in 1D and 2D.

Definition 29 (Rastrigin Reference Cost Function):

$$f_{rastrigin}(x) = 10 \cdot \left[n - \sum_{i=1}^{n}\cos(2\pi x_i)\right] + \sum_{i=1}^{n} x_i^2$$



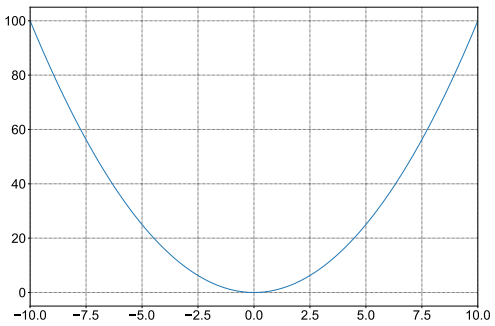


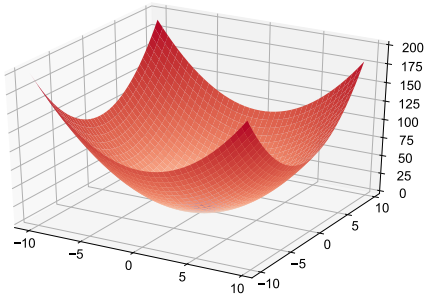
Sphere Reference Cost Function:

Definition 30 defines the f_{sphere} reference cost function and illustrates the reference cost function in 1D and 2D.

Definition 30 (Sphere Reference Cost Function):

$$f_{sphere}(x) = \sum_{i=1}^{n} x_i^2$$



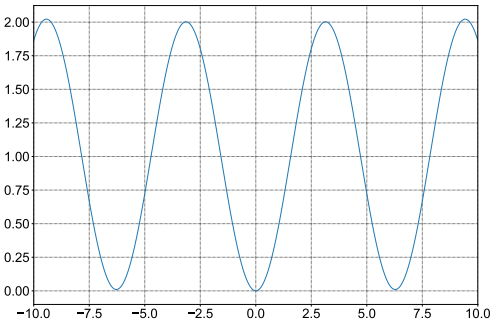


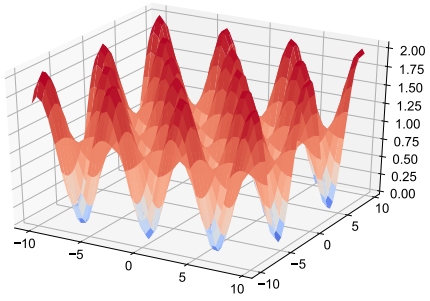
Griewangk Reference Cost Function:

Definition 31 defines the $f_{griewangk}$ reference cost function and illustrates the reference cost function in 1D and 2D.

Definition 31 (Griewangk Reference Cost Function):

$$f_{griewangk}(x) = 1 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{(i)}}) + \sum_{i=1}^{n} \frac{x_i^2}{4000}$$





7.4.2. Extension 2: Behavior Model Generator

This section describes the behavior model generator in the following steps:

Step 1 (Generate paths, activities, and control methods): In the first step, a behavior model generator generates $n_p \in \mathbb{N}_1$ paths with $n_a \in \mathbb{N}_1$ activities and $n_c \in \mathbb{N}_1$ control methods.

Step 2 (Sample command signal configurations): In the second step, for each control method j from the software component A_i, the command signals $P_{ij} = (p_{ij}^{(1)}, \ldots, p_{ij}^{(k)})$, $k \in \mathbb{N}^{\geq 1}$ are uniformly sampled. Decision parameters $B = (b_{ij} \mid i \in \{1, \ldots, m\} \wedge j \in A_i \wedge b_{ij} \in \{0, 1\})$ are sampled from a discrete uniform distribution.

Step 3 (Validate constraints): In the third step, each path is validated against the objective time constraint, see Definition 20. Valid command signal configurations are counted to calculate a success rate.

Figure 28 shows a generated behavior model with $n_p = 3$ paths and $n_a = 2$ activities. Generated methods are not visualized.

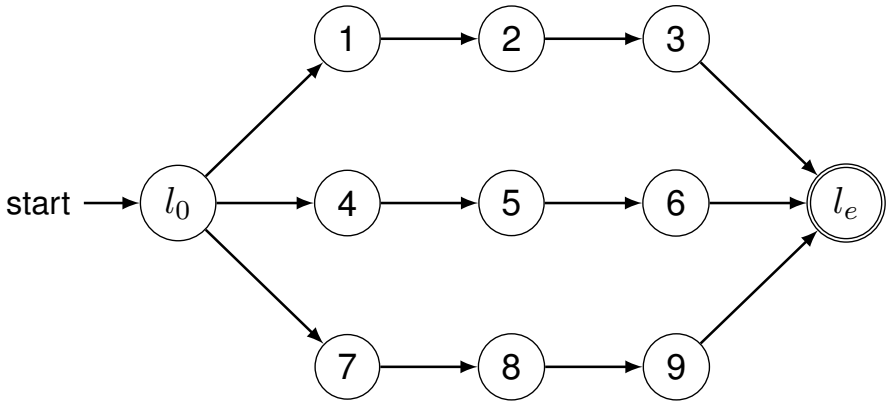


Figure 28 Example result of the behavior model generator: $n_p = 3$ and $n_a = 2$ (methods are not visualized).

8. Evaluation of CyberOpt

This chapter evaluates the approach to machine learning and constrained optimization of command signal configuration named CyberOpt, presented in Chapter 6. This chapter builds on previous work by the author and extends the published results [ON15, OVN16, OVN18b, OVN18a]. The following experiments are defined for a systematic analysis of the CyberOpt approach:

Experiment 1 (Ground truth): The results from the ground truth experiment are used as reference results. The results from the random walk experiment, the black-box optimization experiment results, the CyberOpt approach experiment, and the results from the CyberOpt MEM experiment are compared to these results. In [OVN18b], an evaluation of the storage scenario and the pick-and-place scenario showed that the cost per control method can be estimated by the following linear $c_{ij}^{lin}(p_{ij}^{(1)})$ and nonlinear $c_{ij}^{exp}(p_{ij}^{(1)})$ exponential functions with coefficients $\varepsilon_{ij}, \varrho_{ij}, \varsigma_{ij} \in \mathbb{R}$:

$$c_{ij}^{lin}(p_{ij}^{(1)}) = \varrho_{ij}\, p_{ij}^{(1)} + \varsigma_{ij} \tag{8.1a}$$

$$c_{ij}^{exp}(p_{ij}^{(1)}) = \varepsilon_{ij}\, e^{-\varrho_{ij}\, p_{ij}^{(1)}} + \varsigma_{ij} \tag{8.1b}$$

Experiment 2 (Random walk): Experiment 2 tries to find an optimal command signal configuration by random sampling. For each control method j from the software component A_i, the command signals $P_{ij} = (p_{ij}^{(1)}, \ldots, p_{ij}^{(k)})$, $k \in \mathbb{N}^{\geq 1}$ are uniformed sampled. Decision parameters $B = (b_{ij} \mid i \in \{1, \ldots, m\} \ \wedge \ j \in A_i \wedge b_{ij} \in \{0, 1\})$ are sampled from a discrete uniform distribution. For each objective time value obj, 1000 command signal configurations are generated. The sampled values are then validated by the command signal configuration problem constraints. If these are valid, then they are evaluated by the reference objective function. The best command signal configuration is selected.

Experiment 3 (Black-box optimization): Experiment 3 tries to find an optimal command signal configuration using a black-box optimization approach. The RBFOpt algorithm described in [CN14] is used for this experiment.

Experiment 4 (CyberOpt approach): Experiment 4 tries to find an optimal command signal configuration using the CyberOpt approach.

Experiment 5 (CyberOpt MEM): Experiment 5 tries to find an optimal command signal configuration using the CyberOpt approach. The memory operation mode MEM uses all energy consumption values from previous calculations, e.g., the results from a calculation for

$obj = 13s$ are used to calculate an optimal solution for $obj = 14s$ and so on.

8.1. Storage Scenario

This section describes the results of the storage scenario. Figure 29 illustrates the resulting energy consumption relative to the objective time value of the reference (Experiment 1). The results are published in [OVN18b] and are calculated by finding an optimal solution for each objective time value obj in range $[8s, 30s]$, with step size 0.5, such that 46 optimal solutions are calculated. The minimum energy consumption is $702.72J$ and the maximum energy consumption is $786.48J$ when the scenario performs its behavior in $30s$ or $8s$. If the objective time value $obj \leq 16.5s$, then the control method $drive\text{-}C1(P_{21})$ is used; otherwise, if the objective time value $obj > 16.5s$, then the control method $drive\text{-}C2(P_{22})$ is used.

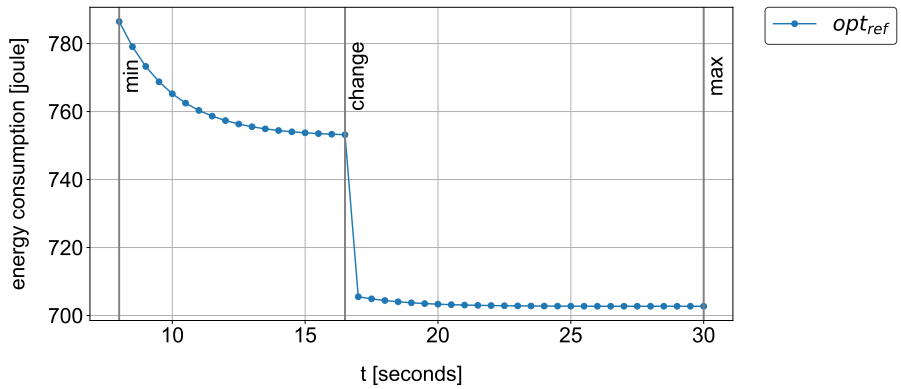


Figure 29 Ground truth of the storage scenario, x-axis objective time value obj in range $[8s, 30s]$ in seconds, y-axis energy consumption in Joules.

The following cost functions and coefficients are used:

$$c_{11}^{exp}, c_{31}^{exp}, c_{21}^{lin}, c_{22}^{lin} \tag{8.2a}$$

$$\varepsilon_{11}, \varepsilon_{31} = 33.03, \varrho_{11}, \varrho_{31} = -0.99, \varsigma_{11}, \varsigma_{31} = 12.63, \tag{8.2b}$$

$$\varrho_{21} = 500, \varrho_{22} = 387.5, \varsigma_{21} = -500, \varsigma_{22} = -1875 \tag{8.2c}$$

Section 8.1.1 describes the results of the random walk experiment. Section 8.1.2 describes the results of the black-box experiment. Section 8.1.3 describes the results of the CyberOpt experiment. Section 8.1.4 describes the results of the CyberOpt MEM mode experiment.

8.1.1. Storage Scenario: Random Walk

This section describes the results of the random walk of the storage scenario. The results shown in Figure 30 are calculated by finding an optimal solution for each objective time value obj in range $[8s, 30s]$ with step size 1, so that 22 solutions are calculated. For each obj in range $[8s, 30s]$, 1000 command signal configurations are uniformly sampled and evaluated with the reference cost functions. The command signal configuration with the minimal cost value is selected. Each evaluation is repeated 20 times. Table 14 summarizes the results of the random walk. In nine out of 22 cases, the distance between the ground truth opt_{ref} and

the mean optimal value opt_{mean} is greater than 10. The sampling approach R1000 is unable to find a solution for $obj = 8$.

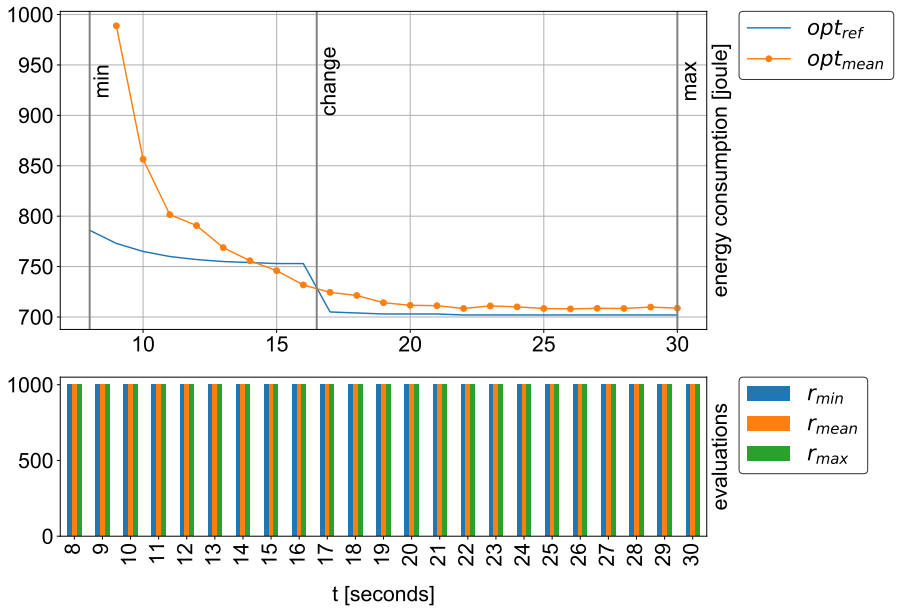


Figure 30 Results of the random walk (storage scenario), x-axis objective time value obj in seconds and in range $[8s, 30s]$ with step size 1, y-axis energy consumption in Joules and number of evaluations.

obj	opt_{ref}	opt_{min}	opt_{mean}	opt_{max}	opt_{std}	opt_{dist}
8	786	NSF	NSF	NSF	NSF	NSF
9	773	900.9279	988.8169	1111.7279	72.7583	-215.8169
10	765	767.9404	856.4873	1056.1030	78.2048	-91.4873
11	760	765.8132	801.5464	885.9590	27.2687	-41.5464
12	757	764.1548	790.7006	848.9165	21.8777	-33.7006
13	755	724.5657	768.8227	793.4232	14.8617	-13.8227
14	754	718.4507	755.8107	793.0004	22.2644	-1.8107
15	753	716.9081	745.9851	775.5675	17.0244	7.0149
16	753	712.0425	731.8156	775.8639	18.7404	21.1844
17	705	710.1256	724.4573	754.3771	10.2411	-19.4573
18	704	710.4492	721.3627	746.0513	9.9821	-17.3627
19	703	704.7446	714.1975	726.6075	4.9335	-11.1975
20	703	704.7477	711.6047	720.9809	5.3991	-8.6047
21	703	703.8890	711.2303	723.0519	5.2034	-8.2303
22	702	703.6426	708.4655	719.8426	3.6600	-6.4655
23	702	703.1116	711.0285	721.3317	5.2410	-9.0285
24	702	703.5296	710.0481	719.7134	4.3503	-8.0481
25	702	703.4374	708.4664	718.1470	4.4060	-6.4664
26	702	703.7324	708.0408	716.4453	3.0146	-6.0408
27	702	703.0879	708.6443	714.1509	3.4084	-6.6443
28	702	704.0457	708.4751	715.5699	3.5653	-6.4751
29	702	703.4226	709.8340	722.0039	4.5399	-7.8340
30	702	703.3426	708.7500	720.3681	4.5243	-6.7500

Table 14 Optimal solutions of the random walk (storage scenario), NSF=no solution found, $opt_{dist} \geq \pm 10$ are highlighted.

8.1.2. Storage Scenario: Black-Box Optimization

This section describes the results of the black-box optimization of the storage scenario. The results shown in Figure 31 are calculated by finding an optimal solution for each objective time value obj in range $[8s, 30s]$ with step size 1, so that 22 solutions are calculated. For

each obj in range $[8s, 30s]$, the black-box optimization can sample a maximum of 100 values of the reference cost function. Each evaluation is repeated 20 times. Table 15 summarizes the results of the black-box optimization. In four out of 22 cases, the distance between the ground truth opt_{ref} and the mean optimal value opt_{mean} is greater than 10. The black-box optimization unable to find a solution for $obj = 8$ and for $obj = 9$.

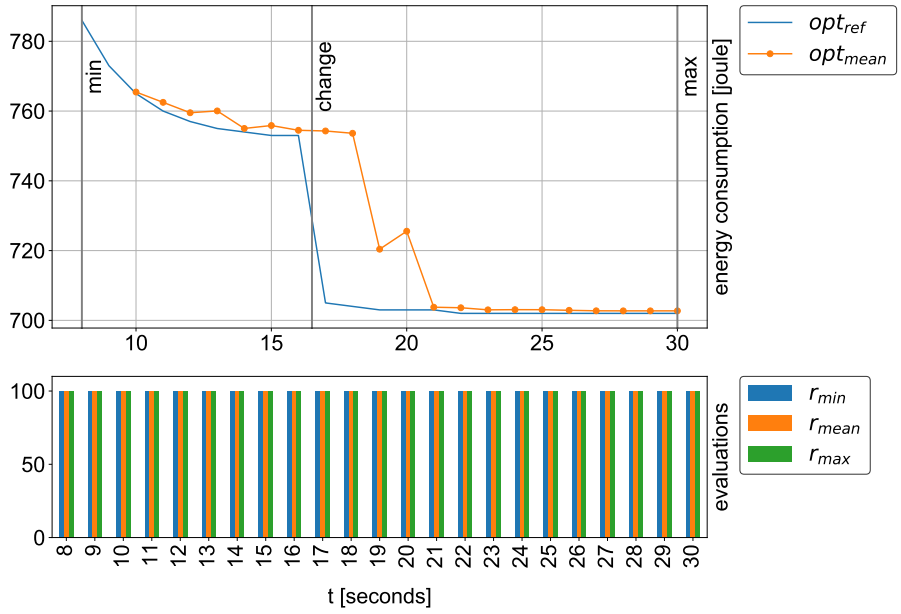


Figure 31 Results of the black-box optimization (storage scenario), x-axis objective time value obj in seconds and in range $[8s, 30s]$ with step size 1, y-axis energy consumption in Joules and number of evaluations.

obj	opt_{ref}	opt_{min}	opt_{mean}	opt_{max}	opt_{std}	opt_{dist}
8	786	NSF	NSF	NSF	NSF	NSF
9	773	NSF	NSF	NSF	NSF	NSF
10	765	765.4542	765.4542	765.4542	0.0000	-0.4542
11	760	762.2708	762.4995	762.6227	0.1679	-2.4995
12	757	759.0478	759.5213	759.9949	0.4736	-2.5213
13	755	759.9253	760.0390	760.6833	0.2707	-5.0390
14	754	754.7234	755.0171	755.3107	0.2936	-1.0171
15	753	754.3213	755.8524	756.0338	0.5104	-2.8524
16	753	754.4359	754.4872	755.4626	0.2238	-1.4872
17	705	753.7574	754.2909	754.9109	0.4573	-49.2909
18	704	753.4523	753.6317	753.8111	0.1794	-49.6317
19	703	708.5320	720.4006	721.0252	2.7228	-17.4006
20	703	715.9417	725.5553	727.9587	4.8068	-22.5553
21	703	703.7221	703.7677	704.6346	0.1989	-0.7677
22	702	703.3808	703.6154	703.9020	0.2593	-1.6154
23	702	702.9805	703.0162	703.6947	0.1557	-1.0162
24	702	702.9805	703.0665	703.1954	0.1053	-1.0665
25	702	702.9805	703.0557	703.1954	0.1025	-1.0557
26	702	702.7247	702.8800	703.0353	0.1553	-0.8800
27	702	702.7247	702.7403	703.0353	0.0677	-0.7403
28	702	702.7247	702.7247	702.7247	0.0000	-0.7247
29	702	702.7247	702.7247	702.7247	0.0000	-0.7247
30	702	702.7247	702.7247	702.7247	0.0000	-0.7247

Table 15 Optimal solutions of the black-box optimization (storage scenario), NSF=no solution found, $opt_{dist} \geq \pm 10$ rows highlighted.

8.1.3. Storage Scenario: CyberOpt Approach

This section describes the results of the CyberOpt approach of the storage scenario. The results shown in Figure 32 are calculated by finding an optimal solution for each objective time value obj in range $[8s, 30s]$ with step size 1, so that 22 solutions are calculated. For each obj in range $[8s, 30s]$, the CyberOpt approach can sample a maximum of 100 values of the reference cost function. Each evaluation is repeated 20 times.

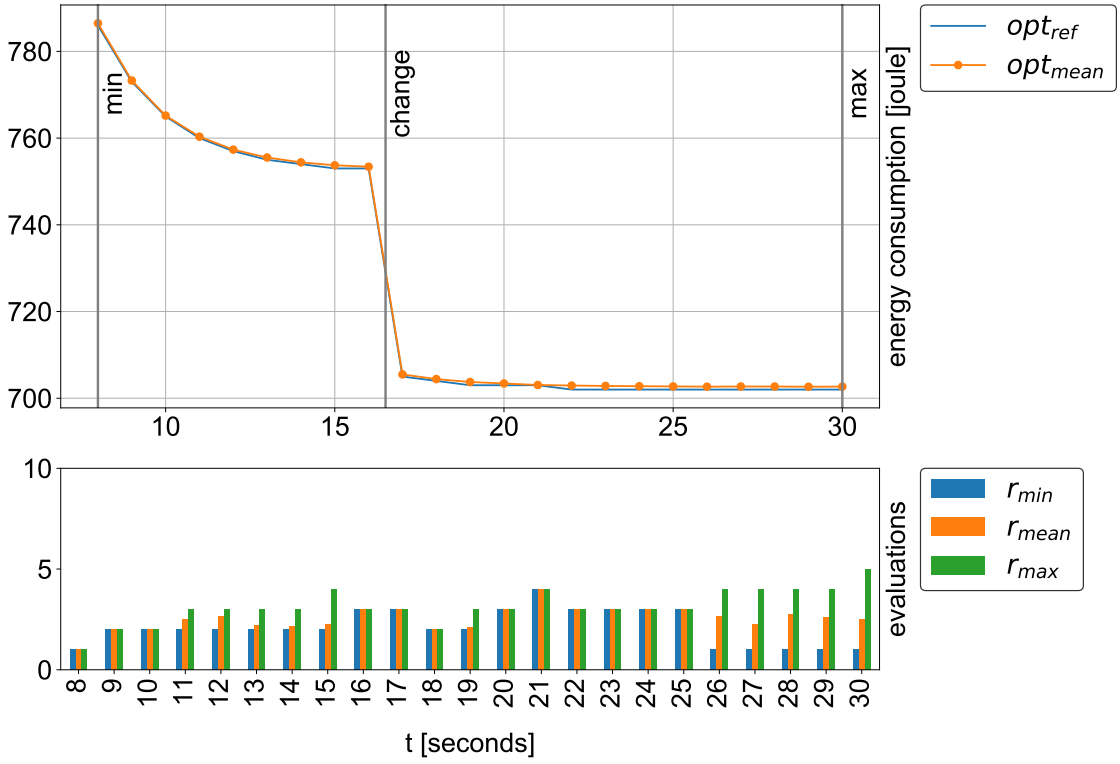


Figure 32 Results of the CyberOpt approach (storage scenario), x-axis objective time value obj in seconds and in range $[8s, 30s]$ with step size 1, y-axis energy consumption in Joules and number of evaluations.

Table 16 summarizes the results of the CyberOpt approach. The maximum distance between the ground truth opt_{ref} and the mean optimal value opt_{mean} is $opt_{dist} = -0.9300$. The CyberOpt approach is able to find an optimal solution for each obj in range $[8s, 30s]$ in each of the 20 repetitions. Table 17 summarizes the number of cost function evaluations. The maximum number of evaluations is $r_{max} = 5$.

obj	opt_{ref}	opt_{min}	opt_{mean}	opt_{max}	opt_{std}	opt_{dist}
8	786	786.4803	786.4803	786.4803	0.0000	-0.4803
9	773	773.2778	773.2778	773.2778	0.0000	-0.2778
10	765	765.1567	765.1925	765.2340	0.0376	-0.1925
11	760	760.3327	760.3330	760.3333	0.0003	-0.3330
12	757	757.3464	757.3468	757.3475	0.0005	-0.3468
13	755	755.4549	755.5214	755.5284	0.0196	-0.5214
14	754	754.3591	754.4108	754.4200	0.0214	-0.4108
15	753	753.6037	753.7349	753.7708	0.0358	-0.7349
16	753	753.4052	753.4052	753.4052	0.0000	-0.4052
17	705	705.4906	705.4906	705.4906	0.0000	-0.4906
18	704	704.4196	704.4198	704.4200	0.0002	-0.4198
19	703	703.7429	703.7440	703.7448	0.0008	-0.7440
20	703	703.4052	703.4052	703.4052	0.0000	-0.4052
21	703	703.0539	703.0539	703.0543	0.0001	-0.0539
22	702	702.9300	702.9300	702.9300	0.0000	-0.9300
23	702	702.8368	702.8368	702.8368	0.0000	-0.8368
24	702	702.7751	702.7751	702.7751	0.0000	-0.7751
25	702	702.7296	702.7296	702.7296	0.0000	-0.7296
26	702	702.3838	702.6665	702.8432	0.1658	-0.6665
27	702	702.3951	702.7099	702.8248	0.1098	-0.7099
28	702	702.3838	702.7031	702.8432	0.1222	-0.7031
29	702	702.3766	702.6521	702.8432	0.1509	-0.6521
30	702	702.3951	702.6866	702.8320	0.1246	-0.6866

Table 16 Optimal solutions of the CyberOpt approach (storage scenario); the maximum distance between opt_{ref} and opt_{mean} is highlighted.

obj	r_{min}	r_{mean}	r_{std}	r_{max}	**success**
8	1	1.0000	0.0000	1	20/20
9	2	2.0000	0.0000	2	20/20
10	2	2.0000	0.0000	2	20/20
11	2	2.5000	0.5000	3	20/20
12	2	2.6500	0.4770	3	20/20
13	2	2.2000	0.4000	3	20/20
14	2	2.1500	0.3571	3	20/20
15	2	2.2500	0.6225	4	20/20
16	3	3.0000	0.0000	3	20/20
17	3	3.0000	0.0000	3	20/20
18	2	2.0000	0.0000	2	20/20
19	2	2.1000	0.3000	3	20/20
20	3	3.0000	0.0000	3	20/20
21	4	4.0000	0.0000	4	20/20
22	3	3.0000	0.0000	3	20/20
23	3	3.0000	0.0000	3	20/20
24	3	3.0000	0.0000	3	20/20
25	3	3.0000	0.0000	3	20/20
26	1	2.6500	1.1522	4	20/20
27	1	2.2500	1.2990	4	20/20
28	1	2.7500	1.2600	4	20/20
29	1	2.6000	1.2000	4	20/20
30	1	2.5000	1.4318	5	20/20

Table 17 Number of evaluations of the CyberOpt approach (storage scenario); the maximum value of r_{max} is highlighted.

8.1.4. Storage Scenario: CyberOpt MEM

This section describes the results of the CyberOpt MEM of the storage scenario. The results shown in Figure 33 are calculated by finding an optimal solution for each objective time value obj in range $[8s, 30s]$ with step size 1, so that 22 solutions are calculated. For each obj in range $[8s, 30s]$, the CyberOpt MEM can sample a maximum of 100 values of the reference cost function. Each evaluation is repeated 20 times.

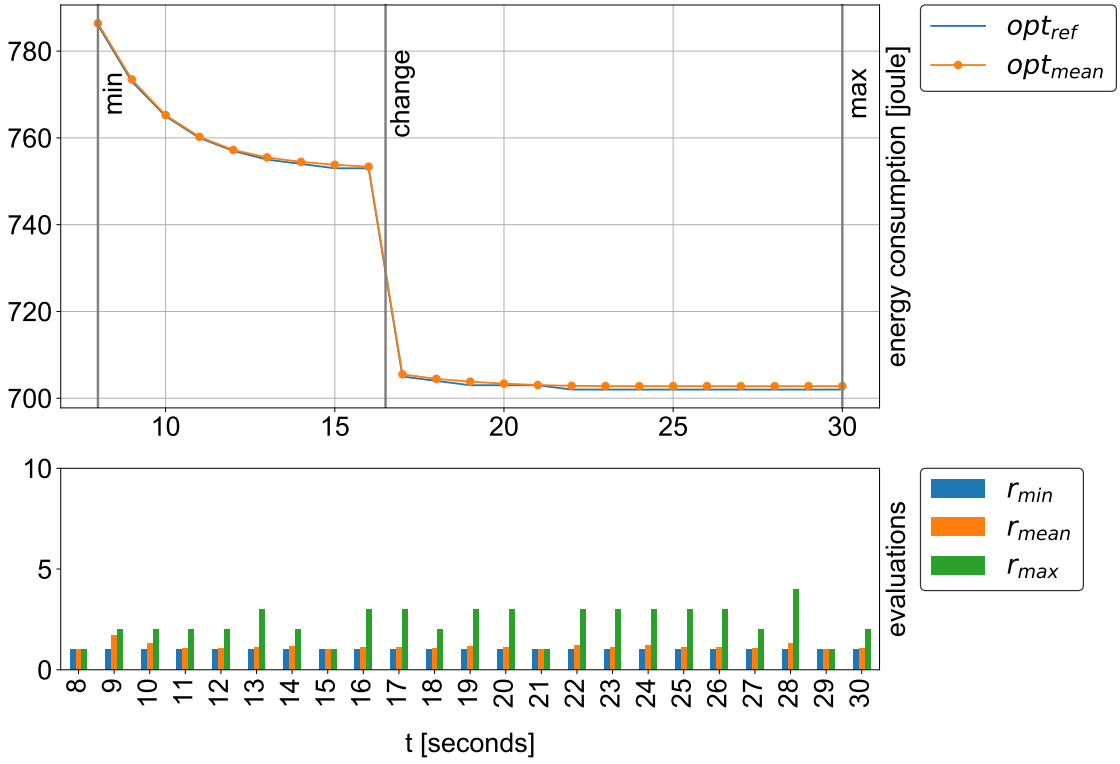


Figure 33 Results of the CyberOpt MEM (storage scenario), x-axis objective time value obj in seconds and in range $[8s, 30s]$ with step size 1, y-axis energy consumption in Joules and number of evaluations.

Table 18 summarizes the results of the CyberOpt MEM. The maximum distance between the ground truth opt_{ref} and the mean optimal value opt_{mean} is $opt_{dist} = -0.8630$. The CyberOpt approach is able to find an optimal solution for each obj in range $[8s, 30s]$ in each of the 20 repetitions. Table 17 summarizes the number of cost function evaluations. The maximum number of evaluations is $r_{max} = 4$.

obj	opt_{ref}	opt_{min}	opt_{mean}	opt_{max}	opt_{std}	opt_{dist}
8	786	786.3382	786.4147	786.4803	0.0512	-0.4147
9	773	773.2778	773.4816	773.5824	0.0751	-0.4816
10	765	765.1274	765.2622	765.3882	0.0915	-0.2622
11	760	760.1605	760.2424	760.4292	0.0771	-0.2424
12	757	757.1763	757.2302	757.3475	0.0365	-0.2302
13	755	755.4215	755.5005	755.6007	0.0418	-0.5005
14	754	754.4196	754.4750	754.5284	0.0355	-0.4750
15	753	753.7748	753.8127	753.8619	0.0243	-0.8127
16	753	753.2395	753.3665	753.4179	0.0380	-0.3665
17	705	705.4379	705.5026	705.6171	0.0399	-0.5026
18	704	704.3819	704.4683	704.5365	0.0419	-0.4683
19	703	703.7862	703.8263	703.8663	0.0231	-0.8263
20	703	703.3270	703.3680	703.4105	0.0247	-0.3680
21	703	702.9595	703.0437	703.1029	0.0366	-0.0437
22	702	702.7831	702.8630	702.9300	0.0406	-0.8630
23	702	702.7342	702.8037	702.8590	0.0260	-0.8037
24	702	702.7620	702.7940	702.8563	0.0239	-0.7940
25	702	702.7296	702.8002	702.8608	0.0283	-0.8002
26	702	702.7139	702.7962	702.8460	0.0310	-0.7962
27	702	702.4904	702.7834	702.8495	0.0715	-0.7834
28	702	702.3951	702.7846	702.8447	0.0931	-0.7846
29	702	702.7190	702.7970	702.8701	0.0357	-0.7970
30	702	702.7606	702.8038	702.8491	0.0227	-0.8038

Table 18 Optimal solutions of the CyberOpt MEM (storage scenario), the maximum distance between opt_{ref} and opt_{mean} is highlighted.

obj	r_{min}	r_{mean}	r_{std}	r_{max}	**success**
8	1	1.0000	0.0000	1	20/20
9	1	1.7000	0.4583	2	20/20
10	1	1.3000	0.4583	2	20/20
11	1	1.0500	0.2179	2	20/20
12	1	1.0500	0.2179	2	20/20
13	1	1.1000	0.4359	3	20/20
14	1	1.1500	0.3571	2	20/20
15	1	1.0000	0.0000	1	20/20
16	1	1.1000	0.4359	3	20/20
17	1	1.1000	0.4359	3	20/20
18	1	1.0500	0.2179	2	20/20
19	1	1.1500	0.4770	3	20/20
20	1	1.1000	0.4359	3	20/20
21	1	1.0000	0.0000	1	20/20
22	1	1.2000	0.6000	3	20/20
23	1	1.1000	0.4359	3	20/20
24	1	1.2000	0.6000	3	20/20
25	1	1.1000	0.4359	3	20/20
26	1	1.1000	0.4359	3	20/20
27	1	1.0500	0.2179	2	20/20
28	1	1.3000	0.7810	4	20/20
29	1	1.0000	0.0000	1	20/20
30	1	1.0500	0.2179	2	20/20

Table 19 Number of evaluations of the CyberOpt MEM (storage scenario), the maximum value of r_{max} is highlighted.

8.1.5. Storage Scenario: Comparison

This section compares the results of the above approaches. Figure 34 illustrates the results of the reference opt_{ref} (ground truth), the results of the R1000 sampling, the results of the black-box optimization, the results of the CyberOpt approach, and the results of CyberOpt MEM.

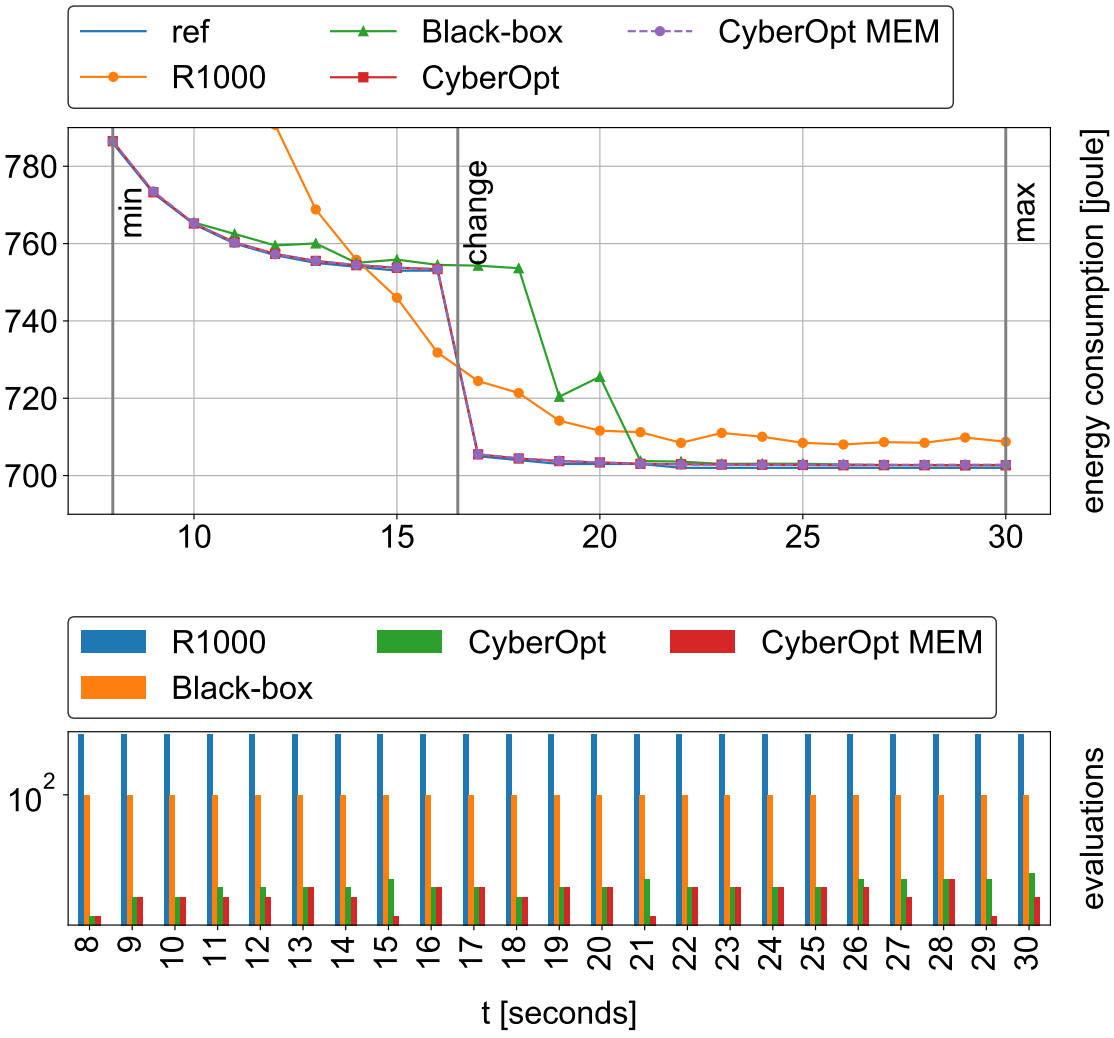


Figure 34 Comparison of approaches (storage scenario), x-axis objective time value obj in seconds and in range $[8s, 30s]$ with step size 1, y-axis energy consumption in Joules and number of evaluations.

Table 20 summarizes the distances between the ground truth and the found optimal solutions of the different approaches. For each objective time value obj, the CyberOpt approach and the CyberOpt MEM calculate more optimal solutions than the R1000 sampling and the black-box optimization. They are able to find an optimal solution with fewer cost function evaluations in every case and every repetition.

obj	opt_{ref}	R1000	Black-box	CyberOpt	MEM
8	786	NSF	NSF	-0.4803	**-0.4147**
9	773	-215.8169	NSF	**-0.2778**	-0.4816
10	765	-91.4873	-0.4542	**-0.1925**	-0.2622
11	760	-41.5464	-2.4995	-0.3330	**-0.2424**
12	757	-33.7006	-2.5213	-0.3468	**-0.2302**
13	755	-13.8227	-5.0390	-0.5214	**-0.5005**
14	754	-1.8107	-1.0171	**-0.4108**	-0.4750
15	753	7.0149	-2.8524	**-0.7349**	-0.8127
16	753	21.1844	-1.4872	-0.4052	**-0.3665**
17	705	-19.4573	-49.2909	**-0.4906**	-0.5026
18	704	-17.3627	-49.6317	**-0.4198**	-0.4683
19	703	-11.1975	-17.4006	**-0.7440**	-0.8263
20	703	-8.6047	-22.5553	-0.4052	**-0.3680**
21	703	-8.2303	-0.7677	-0.0539	**-0.0437**
22	702	-6.4655	-1.6154	-0.9300	**-0.8630**
23	702	-9.0285	-1.0162	-0.8368	**-0.8037**
24	702	-8.0481	-1.0665	**-0.7751**	-0.7940
25	702	-6.4664	-1.0557	**-0.7296**	-0.8002
26	702	-6.0408	-0.8800	**-0.6665**	-0.7962
27	702	-6.6443	-0.7403	**-0.7099**	-0.7834
28	702	-6.4751	-0.7247	**-0.7031**	-0.7846
29	702	-7.8340	-0.7247	**-0.6521**	-0.7970
30	702	-6.7500	-0.7247	**-0.6866**	-0.8038

Table 20 Distances between the ground truth and the found optimal solutions of the different approaches (storage scenario), NSF=no solution found.

8.2. Pick-and-Place Scenario

This section describes the results of the pick-and-place scenario. Figure 35 illustrates the resulting energy consumption relative to the objective time value of reference.

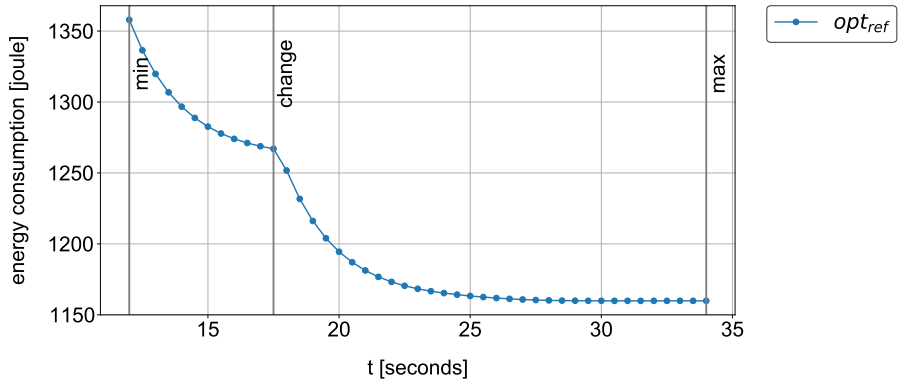


Figure 35 Ground truth of the pick-and-place scenario, x-axis objective time value obj in range $[12s, 34s]$ in seconds, y-axis energy consumption in Joules.

The results are published in [OVN18b] and are calculated by finding an optimal solution for each objective time value obj in range $[12s, 34s]$, with step size 0.5, so that 46 optimal solutions are calculated. The minimum energy consumption is $1159.85J$ and the maximum energy consumption $1357.96J$ when the pick-and-place scenario performs its behavior in $34s$ or $12s$. If the objective time value $obj \leq 17.5s$, then the control method $drive\text{-}C1(P_{61})$ is used; otherwise, if the objective time value $obj > 17.5s$, then the control method $drive\text{-}C2(P_{62})$ is

used. The following cost functions and coefficients are used:

$$c_{11}^{exp}, c_{21}^{exp}, c_{31}^{exp}, c_{41}^{exp}, c_{51}^{exp}, c_{71}^{exp}, c_{61}^{lin}, c_{62}^{lin} \tag{8.3a}$$

$$\varepsilon_{11}, \varepsilon_{21}, \varepsilon_{31}, \varepsilon_{41}, \varepsilon_{51}, \varepsilon_{71} = 33.03, \tag{8.3b}$$

$$\varrho_{11}, \varrho_{21}, \varrho_{31}, \varrho_{41}, \varrho_{51}, \varrho_{71} = -0.99, \tag{8.3c}$$

$$\varsigma_{11}, \varsigma_{21}, \varsigma_{31}, \varsigma_{41}, \varsigma_{51}, \varsigma_{71} = 12.63, \tag{8.3d}$$

$$\varrho_{61} = 333.3, \varrho_{62} = 800, \varsigma_{61} = -166.7, \varsigma_{62} = -6000 \tag{8.3e}$$

Section 8.2.1 describes the results of the random walk experiment. Section 8.2.2 describes the results of the black-box experiment. Section 8.2.3 describes the results of the CyberOpt experiment. Section 8.2.4 describes the results of the CyberOpt MEM mode experiment.

8.2.1. Pick-and-Place: Random Walk

This section describes the results of the random walk of the pick-and-place scenario. The results shown in Figure 36 are calculated by finding an optimal solution for each objective time value obj in range $[12s, 34s]$ with step size 1, so that 22 solutions are calculated. For each obj in range $[12s, 34s]$, 1000 command signal configurations are uniformly sampled and evaluated by the reference cost functions. The command signal configuration with the minimum cost value is selected. Each evaluation is repeated 20 times.

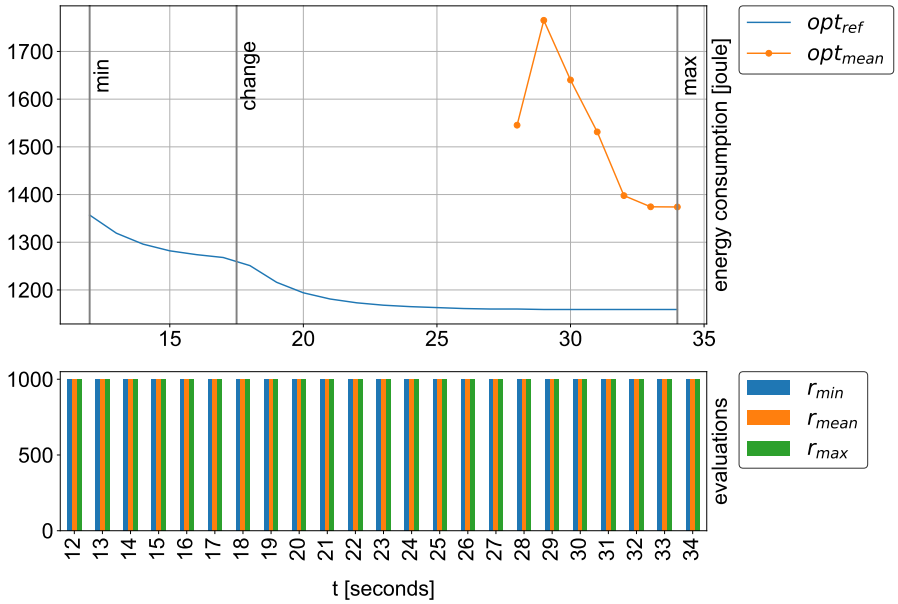


Figure 36 Results of the random walk (pick-and-place scenario), x-axis objective time value obj in seconds and in range $[12s, 34s]$ with step size 1, y-axis energy consumption in Joules and number of evaluations.

Table 21 summarizes the results of the random walk. In seven out of 22 cases, the distance between the ground truth opt_{ref} and the mean optimal value opt_{mean} is greater than 10. The sampling approach R1000 is only able to find a solution for $obj \in \{28, 29, 30, 31, 32, 33, 34\}$.

obj	opt_{ref}	opt_{min}	opt_{mean}	opt_{max}	opt_{std}	opt_{dist}
12	1357	NSF	NSF	NSF	NSF	NSF
13	1319	NSF	NSF	NSF	NSF	NSF
14	1296	NSF	NSF	NSF	NSF	NSF
15	1282	NSF	NSF	NSF	NSF	NSF
16	1274	NSF	NSF	NSF	NSF	NSF
17	1268	NSF	NSF	NSF	NSF	NSF
18	1251	NSF	NSF	NSF	NSF	NSF
19	1216	NSF	NSF	NSF	NSF	NSF
20	1194	NSF	NSF	NSF	NSF	NSF
21	1181	NSF	NSF	NSF	NSF	NSF
22	1173	NSF	NSF	NSF	NSF	NSF
23	1168	NSF	NSF	NSF	NSF	NSF
24	1165	NSF	NSF	NSF	NSF	NSF
25	1163	NSF	NSF	NSF	NSF	NSF
26	1161	NSF	NSF	NSF	NSF	NSF
27	1160	NSF	NSF	NSF	NSF	NSF
28	1160	1406.6618	1545.4134	1684.1650	138.7516	-385.4134
29	1159	1576.8717	1765.2713	2118.9721	250.2820	-606.2713
30	1159	1368.4387	1640.1907	1988.1840	210.6989	-481.1907
31	1159	1318.7656	1531.3286	1908.1532	176.2895	-372.3286
32	1159	1307.1587	1397.7711	1567.9231	72.7594	-238.7711
33	1159	1299.7244	1374.3007	1498.1282	61.9761	-215.3007
34	1159	1287.7350	1373.9593	1614.0599	73.2958	-214.9593

Table 21 Optimal solutions of the random walk (pick-and-place scenario), NSF=no solution found, $opt_{dist} \geq \pm 10$ are highlighted.

8.2.2. Pick-and-Place: Black-Box Optimization

This section describes the results of the black-box optimization of the pick-and-place scenario. The results shown in Figure 37 are calculated by finding an optimal solution for each objective time value obj in range $[12s, 34s]$ with step size 1, so that 22 solutions are calculated. For each obj in range $[12s, 34s]$, the black-box optimization can sample a maximum of 100 values of the reference cost function. Each evaluation is repeated 20 times.

Table 22 summarizes the results of the black-box optimization. In four out of 22 cases, the distance between the ground truth opt_{ref} and the mean optimal value opt_{mean} is greater than 10. The black-box optimization is only able to find a solution for $obj \in \{31, 32, 33, 34\}$.

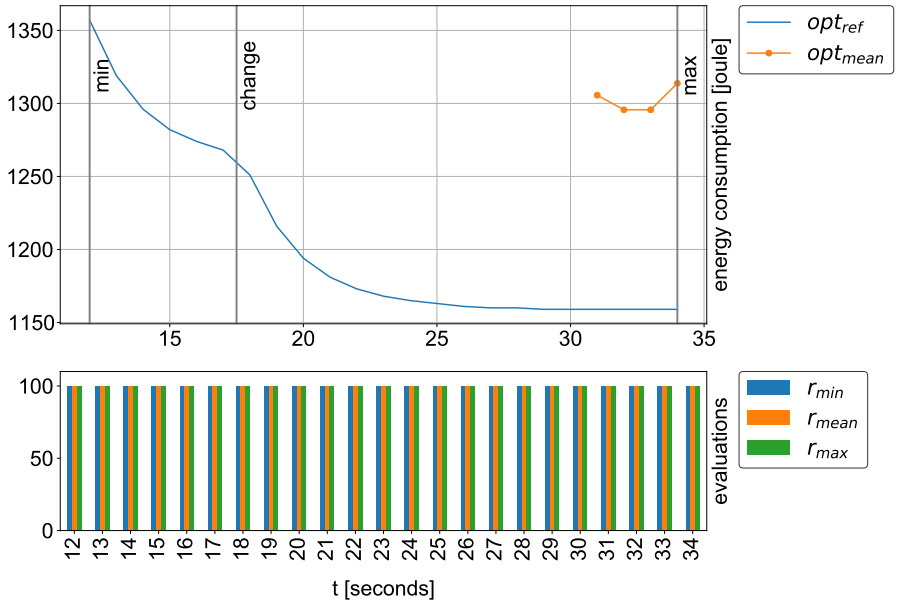


Figure 37 Results of the black-box optimization (pick-and-place scenario), x-axis objective time value obj in seconds and in range $[12s, 32s]$ with step size 1, y-axis energy consumption in Joules and number of evaluations.

obj	opt_{ref}	opt_{min}	opt_{mean}	opt_{max}	opt_{std}	opt_{dist}
12	1357	NSF	NSF	NSF	NSF	NSF
13	1319	NSF	NSF	NSF	NSF	NSF
14	1296	NSF	NSF	NSF	NSF	NSF
15	1282	NSF	NSF	NSF	NSF	NSF
16	1274	NSF	NSF	NSF	NSF	NSF
17	1268	NSF	NSF	NSF	NSF	NSF
18	1251	NSF	NSF	NSF	NSF	NSF
19	1216	NSF	NSF	NSF	NSF	NSF
20	1194	NSF	NSF	NSF	NSF	NSF
21	1181	NSF	NSF	NSF	NSF	NSF
22	1173	NSF	NSF	NSF	NSF	NSF
23	1168	NSF	NSF	NSF	NSF	NSF
24	1165	NSF	NSF	NSF	NSF	NSF
25	1163	NSF	NSF	NSF	NSF	NSF
26	1161	NSF	NSF	NSF	NSF	NSF
27	1160	NSF	NSF	NSF	NSF	NSF
28	1160	NSF	NSF	NSF	NSF	NSF
29	1159	NSF	NSF	NSF	NSF	NSF
30	1159	NSF	NSF	NSF	NSF	NSF
31	1159	1305.6218	1305.6218	1305.6218	0.0000	-146.6218
32	1159	1295.6014	1295.6014	1295.6014	0.0000	-136.6014
33	1159	1295.6014	1295.6014	1295.6014	0.0000	-136.6014
34	1159	1313.7720	1313.7720	1313.7720	0.0000	-154.7720

Table 22 Optimal solutions of the black-box optimization (pick-and-place scenario), NSF=no solution found, $opt_{dist} \geq \pm 10$ are highlighted.

8.2.3. Pick-and-Place: CyberOpt Approach

This section describes the results of the CyberOpt approach of the pick-and-place scenario. The results shown in Figure 38 are calculated by finding an optimal solution for each objective time value obj in range $[12s, 34s]$ with step size 1, so that 22 solutions are calculated. For each obj in range $[12s, 34s]$, the CyberOpt approach can sample a maximum of 100 values of the reference cost function. Each evaluation is repeated 20 times.

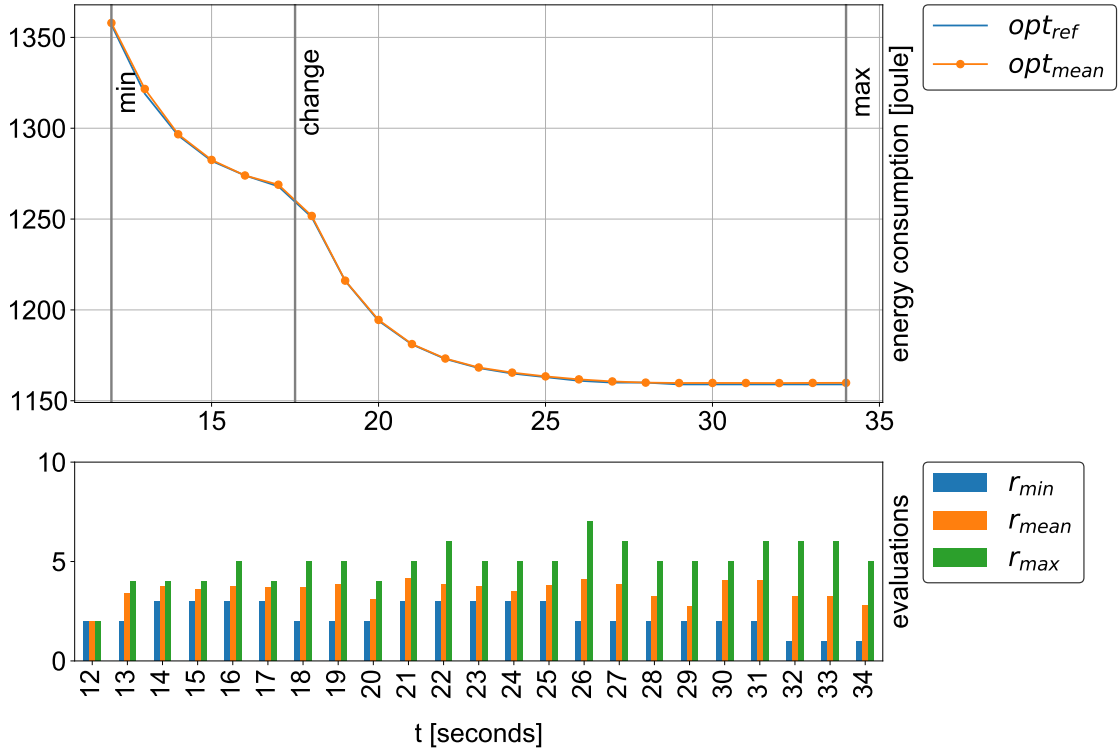


Figure 38 Results of the CyberOpt approach (pick-and-place scenario), x-axis objective time value obj in seconds and in range $[12s, 32s]$ with step size 1, y-axis energy consumption in Joules and number of evaluations.

Table 23 summarizes the results of the CyberOpt approach. The maximum distance between the ground truth opt_{ref} and the mean optimal value opt_{mean} is $opt_{dist} = -2.6159$. The CyberOpt approach is able to find an optimal solution for each obj in range $[12s, 34s]$ in each of the 20 repetitions. Table 24 summarizes the number of cost function evaluations. The maximum number of evaluations is $r_{max} = 7$.

obj	opt_{ref}	opt_{min}	opt_{mean}	opt_{max}	opt_{std}	opt_{dist}
12	1357	1357.9647	1357.9647	1357.9647	0.0000	-0.9647
13	1319	1319.9359	1321.6159	1325.4027	2.4791	-2.6159
14	1296	1296.6055	1296.6746	1296.7711	0.0430	-0.6746
15	1282	1282.4866	1282.5553	1282.6008	0.0291	-0.5553
16	1274	1273.8540	1273.9928	1274.0613	0.0661	0.0072
17	1268	1268.8523	1268.9180	1269.4377	0.1214	-0.9180
18	1251	1251.7048	1251.7235	1251.7734	0.0225	-0.7235
19	1216	1216.0343	1216.1147	1216.2291	0.0445	-0.1147
20	1194	1194.1199	1194.4920	1194.6610	0.1436	-0.4920
21	1181	1180.8715	1181.2014	1181.3027	0.1104	-0.2014
22	1173	1173.2065	1173.2396	1173.2833	0.0279	-0.2396
23	1168	1168.2492	1168.3079	1168.3884	0.0454	-0.3079
24	1165	1165.1689	1165.4881	1165.7457	0.1965	-0.4881
25	1163	1163.3803	1163.4152	1163.4605	0.0241	-0.4152
26	1161	1161.2000	1161.7664	1162.4717	0.3131	-0.7664
27	1160	1160.1831	1160.6246	1160.8189	0.1683	-0.6246
28	1160	1159.7810	1159.9860	1160.2367	0.1680	0.0140
29	1159	1159.4983	1159.7631	1160.0287	0.2063	-0.7631
30	1159	1159.3787	1159.7961	1159.9493	0.1808	-0.7961
31	1159	1159.3743	1159.7789	1159.8946	0.1521	-0.7789
32	1159	1159.0657	1159.7322	1159.9054	0.2376	-0.7322
33	1159	1159.5255	1159.8187	1159.9706	0.1449	-0.8187
34	1159	1159.5183	1159.8524	1159.9664	0.0825	-0.8524

Table 23 Optimal solutions of the CyberOpt approach (pick-and-place scenario), the maximum distance between opt_{ref} and opt_{mean} is highlighted.

obj	r_{min}	r_{mean}	r_{std}	r_{max}	**success**
12	2	2.0000	0.0000	2	20/20
13	2	3.4000	0.9165	4	20/20
14	3	3.7500	0.4330	4	20/20
15	3	3.6000	0.4899	4	20/20
16	3	3.7500	0.6225	5	20/20
17	3	3.7000	0.4583	4	20/20
18	2	3.7000	0.8426	5	20/20
19	2	3.8500	1.0137	5	20/20
20	2	3.1000	0.8307	4	20/20
21	3	4.1500	0.5723	5	20/20
22	3	3.8500	0.7921	6	20/20
23	3	3.7500	0.8292	5	20/20
24	3	3.5000	0.5916	5	20/20
25	3	3.8000	0.8718	5	20/20
26	2	4.1000	1.5780	7	20/20
27	2	3.8500	1.1079	6	20/20
28	2	3.2500	0.8292	5	20/20
29	2	2.7500	0.8874	5	20/20
30	2	4.0500	0.9206	5	20/20
31	2	4.0500	1.0235	6	20/20
32	1	3.2500	1.5452	6	20/20
33	1	3.2500	1.5452	6	20/20
34	1	2.8000	1.5684	5	20/20

Table 24 Number of evaluations of the CyberOpt approach (pick-and-place scenario), the maximum value of r_{max} is highlighted.

8.2.4. Pick-and-Place: CyberOpt MEM

This section describes the results of the CyberOpt MEM of the pick-and-place scenario. The results shown in Figure 39 are calculated by finding an optimal solution for each objective time value obj in range $[12s, 34s]$ with step size 1, so that 22 solutions are calculated. For each obj in range $[12s, 34s]$, the CyberOpt MEM can sample a maximum of 100 values of the reference cost function. Each evaluation is repeated 20 times.

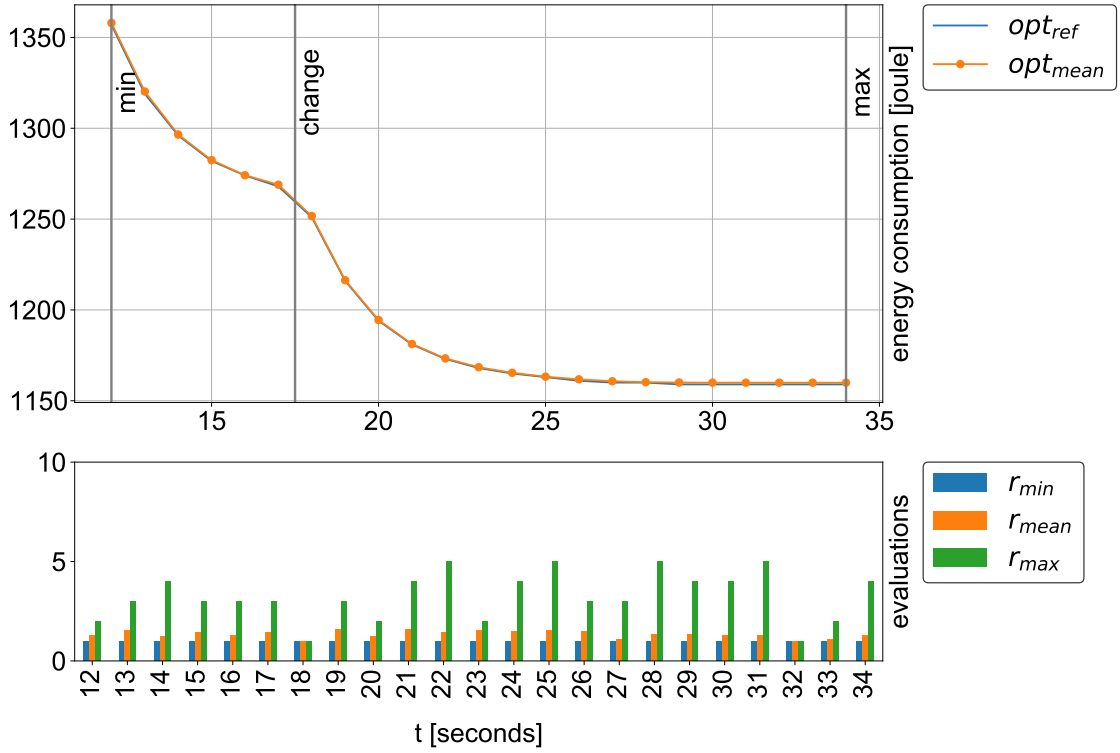


Figure 39 Results of the CyberOpt MEM (pick-and-place scenario), x-axis objective time value obj in seconds and in range $[12s, 34s]$ with step size 1, y-axis energy consumption in Joules and number of evaluations.

Table 25 summarizes the results of the CyberOpt approach. The maximum distance between the ground truth opt_{ref} and the mean optimal value opt_{mean} is $opt_{dist} = -1.2250$. The CyberOpt MEM is able to find an optimal solution for each obj in range $[12s, 34s]$ in each of the 20 repetitions. Table 26 summarizes the number of cost function evaluations. The maximum number of evaluations is $r_{max} = 4$.

obj	opt_{ref}	opt_{min}	opt_{mean}	opt_{max}	opt_{std}	opt_{dist}
12	1357	1357.8397	1357.9868	1358.0968	0.0757	-0.9868
13	1319	1319.7898	1320.2250	1325.4027	1.1935	-1.2250
14	1296	1296.4194	1296.5563	1296.6937	0.0846	-0.5563
15	1282	1282.2882	1282.4702	1282.6982	0.1206	-0.4702
16	1274	1273.9421	1274.1323	1274.3234	0.0883	-0.1323
17	1268	1268.6725	1268.9452	1269.2116	0.1348	-0.9452
18	1251	1251.5421	1251.6707	1251.7582	0.0789	-0.6707
19	1216	1216.2019	1216.3340	1216.5284	0.0862	-0.3340
20	1194	1194.2888	1194.4419	1194.6623	0.1028	-0.4419
21	1181	1181.0715	1181.2513	1181.3607	0.0794	-0.2513
22	1173	1173.1591	1173.3006	1173.5633	0.0879	-0.3006
23	1168	1168.2945	1168.4606	1168.6624	0.1310	-0.4606
24	1165	1165.2911	1165.3995	1165.6163	0.0869	-0.3995
25	1163	1162.9379	1163.2657	1163.4129	0.1087	-0.2657
26	1161	1161.5582	1161.7919	1162.0356	0.1473	-0.7919
27	1160	1160.5950	1160.7496	1160.8779	0.0736	-0.7496
28	1160	1160.1531	1160.2189	1160.3382	0.0633	-0.2189
29	1159	1159.7839	1160.0408	1160.1919	0.0977	-1.0408
30	1159	1159.5410	1159.9636	1160.1683	0.1284	-0.9636
31	1159	1159.7859	1159.9581	1160.1516	0.0822	-0.9581
32	1159	1159.8536	1159.9498	1160.1307	0.0757	-0.9498
33	1159	1159.5862	1159.9547	1160.1038	0.1137	-0.9547
34	1159	1159.7278	1159.9292	1160.1333	0.0930	-0.9292

Table 25 Optimal solutions of the CyberOpt MEM (pick-and-place scenario), the maximum distance between opt_{ref} and opt_{mean} is highlighted.

obj	r_{min}	r_{mean}	r_{std}	r_{max}	**success**
8	1	1.0000	0.0000	1	20/20
9	1	1.7000	0.4583	2	20/20
10	1	1.3000	0.4583	2	20/20
11	1	1.0500	0.2179	2	20/20
12	1	1.0500	0.2179	2	20/20
13	1	1.1000	0.4359	3	20/20
14	1	1.1500	0.3571	2	20/20
15	1	1.0000	0.0000	1	20/20
16	1	1.1000	0.4359	3	20/20
17	1	1.1000	0.4359	3	20/20
18	1	1.0500	0.2179	2	20/20
19	1	1.1500	0.4770	3	20/20
20	1	1.1000	0.4359	3	20/20
21	1	1.0000	0.0000	1	20/20
22	1	1.2000	0.6000	3	20/20
23	1	1.1000	0.4359	3	20/20
24	1	1.2000	0.6000	3	20/20
25	1	1.1000	0.4359	3	20/20
26	1	1.1000	0.4359	3	20/20
27	1	1.0500	0.2179	2	20/20
28	1	1.3000	0.7810	4	20/20
29	1	1.0000	0.0000	1	20/20
30	1	1.0500	0.2179	2	20/20

Table 26 Number of evaluations of the CyberOpt MEM (pick-and-place scenario), the maximum value of r_{max} is highlighted.

8.2.5. Pick-and-Place: Comparison

This section compares the results of the above approaches. Figure 40 illustrates the results of the reference opt_{ref} (ground truth), the results of the R1000 sampling, the results of the black-box optimization, the results of the CyberOpt approach, and the results of CyberOpt MEM.

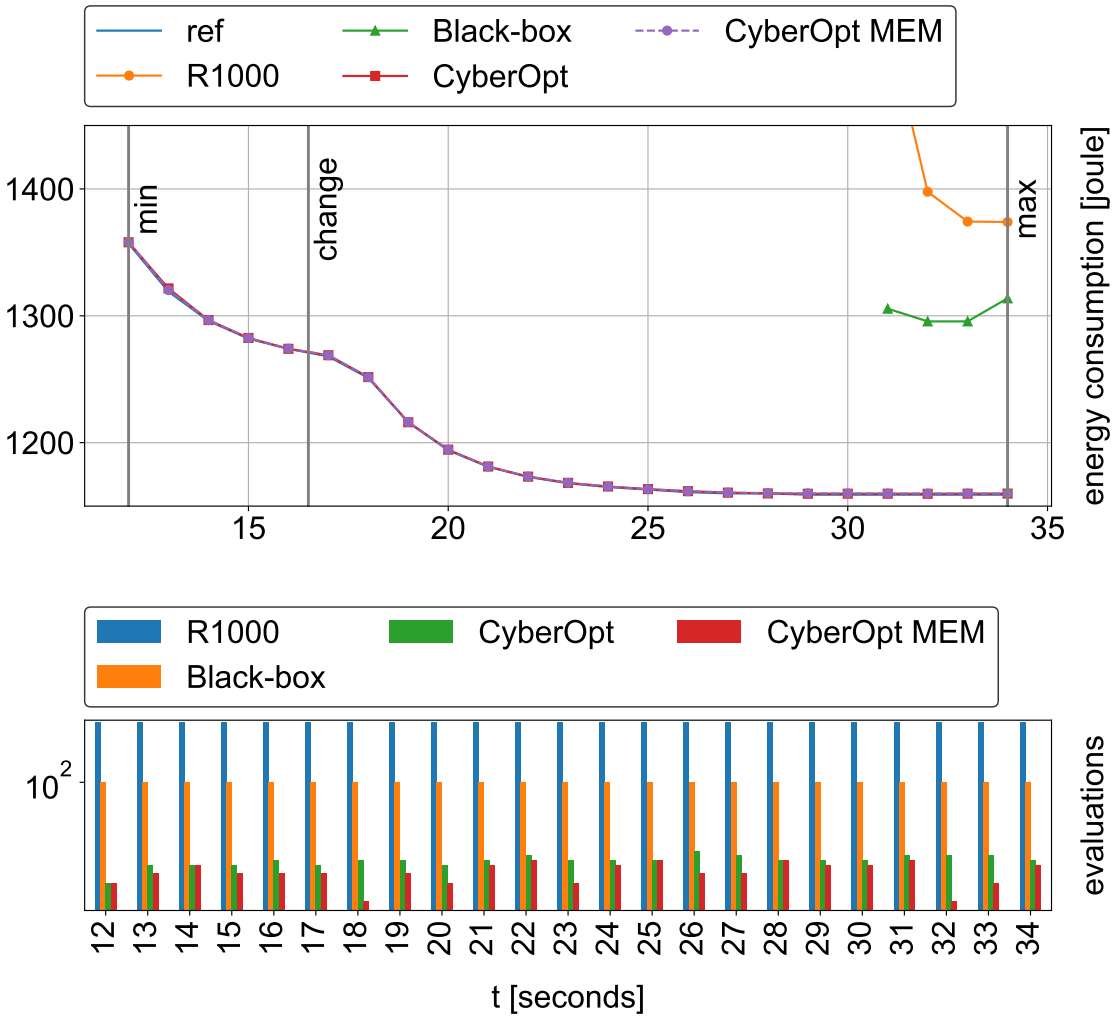


Figure 40 Comparison of approaches (pick-and-place scenario), x-axis objective time value obj in seconds and in range $[12s, 32s]$ with step size 1, y-axis energy consumption in Joules and number of evaluations.

Table 27 summarizes the distances between the ground truth and the found optimal solutions of the different approaches. For each objective time value obj, the CyberOpt approach and the CyberOpt MEM calculate more optimal solutions than the R1000 sampling and the black-box optimization. They are able to find an optimal solution with fewer cost function evaluations in every case and every repetition.

obj	opt_{ref}	**R1000**	**Black Box**	**CyberOpt**	**MEM**
12	1357	NSF	NSF	**-0.9647**	-0.9868
13	1319	NSF	NSF	-2.6159	**-1.2250**
14	1296	NSF	NSF	-0.6746	**-0.5563**
15	1282	NSF	NSF	-0.5553	**-0.4702**
16	1274	NSF	NSF	**0.0072**	-0.1323
17	1268	NSF	NSF	**-0.9180**	-0.9452
18	1251	NSF	NSF	-0.7235	**-0.6707**
19	1216	NSF	NSF	**-0.1147**	-0.3340
20	1194	NSF	NSF	-0.4920	**-0.4419**
21	1181	NSF	NSF	**-0.2014**	-0.2513
22	1173	NSF	NSF	**-0.2396**	-0.3006
23	1168	NSF	NSF	**-0.3079**	-0.4606
24	1165	NSF	NSF	-0.4881	**-0.3995**
25	1163	NSF	NSF	-0.4152	**-0.2657**
26	1161	NSF	NSF	**-0.7664**	-0.7919
27	1160	NSF	NSF	**-0.6246**	-0.7496
28	1160	-385.4134	NSF	**0.0140**	-0.2189
29	1159	-606.2713	NSF	**-0.7631**	-1.0408
30	1159	-481.1907	NSF	**-0.7961**	-0.9636
31	1159	-372.3286	-146.6218	**-0.7789**	-0.9581
32	1159	-238.7711	-136.6014	**-0.7322**	-0.9498
33	1159	-215.3007	-136.6014	**-0.8187**	-0.9547
34	1159	-214.9593	-154.7720	**-0.8524**	-0.9292

Table 27 Distances between the ground truth and the found optimal solutions of the different approaches (pick-and-place scenario), NSF=no solution found.

8.3. Summary of the Evaluation

This section summarizes the above described four experiments: (1) random walk, (2) black-box optimization, (3) CyberOpt and (4) CyberOpt MEM. The minimum pointwise distance between the reference optimal energy consumption and the optimal energy consumption found by the algorithms is calculated for all obj in range $[8s, 30s]$ (storage scenario) and in range $[12s, 34s]$ (pick-and-place scenario), see Table 14, Table 15, Table 16, Table 18, Table 21, Table 22, Table 23, and Table 25. The evaluations that are required to find a solution for each obj are summarized in Table 17, Table 19, Table 24 and Table 26. The random walk R1000 always requires 1000 evaluations and the black-box optimization always requires 100 evaluations. The best case summarizes the minimum absolute mean distance value d_{bc} of each experiment and the minimum mean evaluations e_{bc} required to find a solution. The average case summarizes the mean absolute mean distance value d_{ac} of each experiment and the mean evaluations e_{ac} required to find a solution. The worst case summarizes the maximum absolute mean distance value d_{wc} of each experiment and the mean evaluations e_{wc} required to find a solution.

In the best case, in the average case, and the worst case, CyberOpt finds optimal solutions for the storage scenario and the pick-and-place scenario with fewer evaluations than the random walk R1000 and the black-box optimization.

The Best Case:

Table 28 summarizes the minimum absolute mean distance value d_{bc} of each experiment. CyberOpt finds the best optimal solutions for the storage scenario (CyberOpt: $d_{bc} = 0.0539$ and CyberOpt MEM: $d_{bc} = 0.0437$) and the pick-and-place scenario (CyberOpt: $d_{bc} = 0.0072$ and CyberOpt MEM: $d_{bc} = 0.1323$). In this case, the best means the one with the smallest distance from the reference. The random walk R1000 finds the worst optimal solutions (storage scenario $d_{bc} = 1.8107$, pick-and-place scenario $d_{bc} = 214.9593$). The black-box optimization finds a good optimal solution for the storage scenario ($d_{bc} = 0.4542$), but the worst optimal solution for the pick-and-place scenario ($d_{bc} = 136.6014$).

Scenario	R1000	Black Box	CyberOpt	MEM
Storage	1.8107	0.4542	0.0539	**0.0437**
Pick-and-place	214.9593	136.6014	**0.0072**	0.1323

Table 28 The best case: minimum absolute mean distance value d_{bc} of each experiment.

Table 29 summarizes the minimum mean evaluations e_{bc} that are required to find a solution for each experiment. The random walk R1000 samples 1000 command signal configurations and the black-box optimization samples a maximum of 100 values of the reference cost function. In the best case, CyberOpt only needs $e_{bc} = 2.1304$ evaluations for the storage scenario and $e_{bc} = 2.2609$ evaluations for the pick-and-place scenario to find an optimal solution. CyberOpt MEM requires only one evaluation to find an optimal solution.

Scenario	R1000	Black Box	CyberOpt	MEM
Storage	1000	100	2.1304	**1.0**
Pick-and-place	1000	100	2.2609	**1.0**

Table 29 The best case: minimum mean evaluations e_{bc} that are required to find a solution.

The Average Case:

Table 30 summarizes the average absolute mean distance value d_{ac} of each experiment. CyberOpt finds the best optimal solutions for the storage scenario (CyberOpt: $d_{ac} = 0.5437$ and CyberOpt MEM: $d_{ac} = 0.5748$) and the pick-and-place scenario (CyberOpt: $d_{ac} = 0.6463$ and CyberOpt MEM: $d_{ac} = 0.6520$). In this case, the best means the one with the smallest mean distance to the reference. The random walk R1000 finds the worst solutions (storage scenario $d_{ac} = 25.2268$, pick-and-place scenario $d_{ac} = 359.1764$). The black-box optimization finds a good solution for the storage scenario ($d_{ac} = 7.8126$), but the second worst solution for the pick-and-place scenario ($d_{ac} = 143.6491$).

Scenario	R1000	Black Box	CyberOpt	MEM
Storage	25.2268	7.8126	**0.5437**	0.5748
Pick-and-place	359.1764	143.6491	**0.6463**	0.6520

Table 30 The average case: mean absolute mean distance value d_{ac} of each experiment.

Table 31 summarizes the mean evaluations e_{ac} that are required to find a solution for each experiment. The random walk R1000 samples 1000 command signal configurations and the

black-box optimization samples a maximum of 100 values of the reference cost function. In the average case, CyberOpt only needs $e_{ac} = 2.5478$ evaluations for the storage scenario and $e = 3.5326$ evaluations for the pick-and-place scenario to find an optimal solution. CyberOpt MEM requires only $e_{ac} = 1.1283$ evaluations for the storage scenario and $e_{ac} = 1.3522$ evaluations for the pick-and-place scenario to find an optimal solution.

Scenario	R1000	Black Box	CyberOpt	MEM
Storage	1000	100	2.5478	**1.1283**
Pick-and-place	1000	100	3.5326	**1.3522**

Table 31 The average case: mean evaluations that are required to find a solution.

The Worst Case:

Table 30 summarizes the maximum absolute mean distance value d_{wc} of each experiment. CyberOpt finds the best optimal solutions for the storage scenario (CyberOpt: $d_{wc} = 0.9300$ and CyberOpt MEM: $d_{wc} = 0.8630$) and the pick-and-place scenario (CyberOpt: $d_{wc} = 2.6159$ and CyberOpt MEM: $d_{wc} = 1.2250$). In this case, the best means the one with the smallest mean distance to the reference. The random walk R1000 finds the worst solutions (storage scenario $d_{wc} = 215.8169$, pick-and-place scenario $d_{wc} = 606.2713$). The black-box optimization finds a good solution for the storage scenario ($d_{wc} = 49.6317$), but the second worst solution for the pick-and-place scenario ($d_{wc} = 154.7720$).

Scenario	R1000	Black Box	CyberOpt	MEM
Storage	215.8169	49.6317	0.9300	**0.8630**
Pick-and-place	606.2713	154.7720	2.6159	**1.2250**

Table 32 The worst case: maximum absolute mean distance value d_{wc} of each experiment.

Table 33 summarizes the maximum mean evaluations e_{wc} that are required to find a solution for each experiment. The random walk R1000 samples 1000 command signal configurations and the black-box optimization samples a maximum of 100 values of the reference cost function. In the worst case, CyberOpt requires only $e_{wc} = 3.1304$ evaluations of the storage scenario and $e_{wc} = 4.9565$ evaluations of the pick-and-place scenario to find an optimal solution. CyberOpt MEM requires only $e_{wc} = 2.3478$ evaluations of the storage scenario and $e_{wc} = 3.2609$ evaluations of the pick-and-place scenario to find an optimal solution.

Scenario	R1000	Black Box	CyberOpt	MEM
Storage	1000	100	3.1304	**2.3478**
Pick-and-place	1000	100	4.9565	**3.2609**

Table 33 The worst case: maximum mean evaluations e_{wc} that are required to find a solution.

9. Proof of Hypotheses

The evaluation of the CyberOpt subalgorithms aims to prove that the machine learning techniques are used to reduce the manual engineering effort. This chapter builds on previous work by the author and extends the published results [ON15, OVN16, OVN18b, OVN18a]. Hypothesis H1 describes that an optimal command signal configuration solution does not require manual engineering steps. The CyberOpt algorithm consists of five subalgorithms, see Chapter 6. Each subalgorithm implements a task of the CyberOpt framework, described in Chapter 5, and uses a particular machine learning technique. In order to prove that hypothesis H1 is valid, hypotheses H2, H3, and H4 must be valid. Hypothesis H1 describes that an optimal command signal configuration solution does not require manual engineering steps if:

H2: a machine learning algorithm exists that learns cost models that can then be used by a command signal configuration algorithm to find optimal configurations,

H3: a machine learning algorithm exists that learns sequences of control methods that can then be used by a command signal configuration algorithm to find optimal command signal configurations, and

H4: an algorithm exists that finds optimal command signal configurations.

In order to prove hypotheses H2, H3, and H4, nine sub-hypotheses are defined, which are summarized in Table 34.

ID	Description	Subalgorithm
H2.1	Regression analysis can be used to learn cost models.	CyberOpt-LCM
H2.2	Linear cost models are not sufficient to predict energy consumption values. Nonlinear cost models should be used.	CyberOpt-LCM
H2.3	Polynomial expansion can be used to model nonlinear cost models.	CyberOpt-LCM
H2.4	The expected improvement criterion can be used to enhance the quality of cost models.	CyberOpt-ICM
H3.1	Event logs of activities can be used to learn a behavior model.	CyberOpt-LSC
H3.2	The fuzzy mining algorithm can be used to learn a behavior model.	CyberOpt-LSC
H4.1	Human-in-the-loop and sampling is not suitable for finding an optimal command signal configuration.	CyberOpt-SIC
H4.2	A general mixed-integer nonlinear programming solver can be used to find an optimal command signal configuration.	CyberOpt-SIC
H4.3	The command signal configuration problem is in the complexity class NP-hard.	CyberOpt-SPC

Table 34 Sub-hypotheses to prove hypotheses H2 - H4.

Proof idea of hypothesis H2: In order to prove hypothesis H2, it can be shown that re-

gression analysis can be used to learn cost models (see sub-hypothesis H2.1), a nonlinear cost model should be used (see sub-hypothesis H2.2), polynomial expansion can be used to model nonlinear cost models (see sub-hypothesis H2.3), and the expected improvement criterion can be used to enhance the quality of cost models (see sub-hypothesis H2.4).

Proof idea of hypothesis H3: In order to prove hypothesis H3, it can be shown that event logs of activities can be used to learn a behavior model (see sub-hypothesis H3.1), and the fuzzy mining algorithm can be used to learn a behavior model (see sub-hypothesis H3.2).

Proof idea of hypothesis H4: In order to prove hypothesis H4, it can be shown that human-in-the-loop and sampling are not suitable for finding an optimal command signal configuration (see sub-hypothesis H4.1), that a general mixed-integer nonlinear programming solver can be used to find an optimal command signal configuration (see sub-hypothesis H4.2) and the command signal configuration problem is in the complexity class NP-hard (see sub-hypothesis H4.3).

9.1. Learning Cost Models

This section describes the evaluation of the CyberOpt-LCM subalgorithm, see Section 6.2. The motivation of the CyberOpt-LCM subalgorithm is to automatically learn a cost model c_{ij} for each control method j from a software component A_i. To find an optimal command signal configuration, cost models are required to compare different command signal configurations. Manual creation of cost models is not required, reducing the manual engineering effort. Sub-hypotheses H2.1, H2.2, and H2.3 should be proven to prove hypothesis H2. Sub-hypothesis H2.1 describes that regression analysis can be used to learn cost models. Sub-hypothesis H2.2 describes that a linear model is not sufficient to predict energy consumption values. A nonlinear cost model should be used. Sub-hypothesis H2.3 describes that polynomial expansion can be used to model nonlinear cost models.

The evaluation of the CyberOpt-LCM subalgorithm is performed by the following experimental setup. The f_{ackley} reference cost function described in Definition 28, the $f_{rastrigin}$ reference cost function described in Definition 29, the f_{sphere} reference cost function described in Definition 30, and the $f_{griewangk}$ reference cost function described in Definition 31 are used to generate synthetic cost values C^*_{ij}. Then, regression analysis is used to learn linear and nonlinear cost models, which are then compared using a calculated regression score value. Regression analysis can be used to learn cost models. Regression analysis describes a statistical process for estimating relationships between dependent and independent variables. The ordinary least squares method used calculates the coefficient values W_{ij} of a linear model $c_{ij}(W_{ij}, P_{ij})$, such that the distances between observed values C^*_{ij} and values of the linear model $c_{ij}(W_{ij}, P_{ij})$ are minimized, see Definition 24. In the case of a nonlinear model, polynomial expansion can be used. It allows linear models to capture nonlinearities; more precisely, it allows a linear model to learn polynomial relationships between dependent and

independent variables, see Definition 25. In the following evaluations, regression analysis is used to calculate linear and nonlinear cost models.

The experiments are described by the following list of steps:

Step 1 (Generating command signal configurations): In the first step, 100000 command signal configurations are generated for each reference function by a Latin hypercube design. From a hypercube with dimension d_f, the algorithm generates n command signal values. The continuous range for each command signal is partitioned into n equally spaced intervals. The command signal configuration values generated by this sampling method are uniformly distributed in the hypercube space.

Step 2 (Generating a polynomial expansion): In the second step, a polynomial expansion is generated (nonlinear cost model c_{ij}). The parameter d_p describes the degree of the generated polynomial.

Step 3 (Fitting the generated cost model c_{ij} to data C_{ij}^*): In the third step, the generated nonlinear cost model c_{ij} is fitted to the cost model training dataset C_{ij}^* (the known function values of the reference cost functions).

Step 4 (Calculating the regression score between data C_{ij}^* and cost model c^*): In the fourth step, a score value $s \in [-\infty, 1]$ is calculated by standard k-fold cross validation and a regression score function. In this case, the coefficient of determination is used, where $s = 1$ describes the best possible score value.

9.1.1. Evidence of Sub-Hypothesis H2.1

The proof 1 examines sub-hypothesis H2.1, which describes that regression analysis can be used to learn cost models. Note that this proof is based on empirical results.

Proof 1 (Evidence of H2.1):

Proof. Regression analysis can be used to learn cost models.

1. The results in Table 35 show that regression analysis can be used to calculate a linear cost model for the reference cost functions f_{ackley}, $f_{rastrigin}$, f_{sphere}, and $f_{griewangk}$.

2. The results in Table 36, Table 37, Table 38, and Table 39 show that regression analysis can be used to calculate a nonlinear cost model for the reference cost functions f_{ackley}, $f_{rastrigin}$, f_{sphere}, and $f_{griewangk}$.

From Description 1 and Description 2, it follows that the sub-hypothesis H2.1, which describes that regression analysis can be used to learn cost models, is valid. □

9.1.2. Evidence of Sub-Hypothesis H2.2

Table 35 summarizes the mean and standard deviation values of the calculated regression scores. The linear cost models were learned from data generated by the reference cost functions. For the reference cost functions f_{ackley}, $f_{rastrigin}$, f_{sphere}, and $f_{griewangk}$, a polynomial of degree $d_p = 1$ is not sufficient to predict cost values. This statement is valid for the function dimensions $d_f = 1, d_f = 2, d_f = 3, d_f = 4$, and $d_f = 5$.

	$d_f = 1$	$d_f = 2$	$d_f = 3$	$d_f = 4$	$d_f = 5$
f_{ackley}	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0002)
$f_{rastrigin}$	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0000)	-0.0001 (+/- 0.0001)	-0.0002 (+/- 0.0001)	-0.0002 (+/- 0.0001)
f_{sphere}	-0.0001 (+/- 0.0000)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0002)	-0.0001 (+/- 0.0001)
$f_{griewangk}$	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)

Table 35 Mean and standard deviation of the regression score values $s \in [-\infty, 1]$ of linear cost models learned from reference cost function data ($s = 1$ describes the best possible score value).

Proof 2 examines sub-hypothesis H2.2. Sub-hypothesis H2.2 describes that a linear cost model is not sufficient to predict energy consumption values. A nonlinear cost model should be used. Please note that this proof is based on empirical results.

Proof 2 (Evidence of H2.2):

Proof. Linear cost models are not sufficient to predict energy consumption values.

The results in Table 35 show that:

1. A linear cost model c_{ij} learned from data of the f_{ackley} reference cost function is not sufficient to predict energy consumption values.
2. A linear cost model c_{ij} learned from data of the $f_{rastrigin}$ reference cost function is not sufficient to predict energy consumption values.
3. A linear cost model c_{ij} learned from data of the f_{sphere} reference cost function is not sufficient to predict energy consumption values.
4. A linear cost model c_{ij} learned from data of the $f_{griewangk}$ reference cost function is not sufficient to predict energy consumption values.

From Descriptions 1 - 4, it follows that sub-hypothesis H2.2, which describes that linear cost models are not sufficient to predict energy consumption values, is valid. □

9.1.3. Evidence of Sub-Hypothesis H2.3

Results of the CyberOpt-LCM subalgorithm are compared with the reference cost functions f_{ackley}, $f_{rastrigin}$, f_{sphere}, and $f_{griewangk}$.

Table 36 summarizes the mean and standard deviation values of the regression scores calculated for the cost models learned from the data generated by the reference cost function f_{ackley}. The parameter $d_f \in \{1, 2, 4, 8, 12\}$ describes the degree of polynomial expansion used. The scoring method used is the coefficient of determination. A polynomial cost model c_{ij} of degree $d_p = 12$ has the best regression score value for the dimensions $d_f \in \{1, 2, 3, 4\}$: $d_f = 1 : s =$ 0.9670 (+/- 0.0002), $d_f = 2 : s =$ 0.9678 (+/- 0.0003), $d_f = 3 : s =$ 0.9657 (+/- 0.0004), and $d_f = 4 : s =$ 0.9633 (+/- 0.0005). A polynomial cost model c_{ij} of degree $d_p = 8$ has the best regression score value of $s =$ 0.9602 (+/- 0.0005) for the dimension $d_f = 5$.

	$d_f = 1$	$d_f = 2$	$d_f = 3$	$d_f = 4$	$d_f = 5$
$d_p = 1$	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0002)
$d_p = 2$	0.7469 (+/- 0.0018)	0.7808 (+/- 0.0030)	0.8144 (+/- 0.0026)	0.8423 (+/- 0.0013)	0.8635 (+/- 0.0022)
$d_p = 4$	0.8992 (+/- 0.0006)	0.9194 (+/- 0.0014)	0.9314 (+/- 0.0013)	0.9397 (+/- 0.0008)	0.9451 (+/- 0.0009)
$d_p = 8$	0.9565 (+/- 0.0003)	0.9618 (+/- 0.0005)	0.9621 (+/- 0.0003)	0.9615 (+/- 0.0004)	**0.9602 (+/- 0.0005)**
$d_p = 12$	**0.9670 (+/- 0.0002)**	**0.9678 (+/- 0.0003)**	**0.9657 (+/- 0.0004)**	**0.9633 (+/- 0.0005)**	0.9581 (+/- 0.0006)

Table 36 Mean and standard deviation of regression score values $s \in [-\infty, 1]$ of cost models learned from data generated by the f_{ackley} reference cost function ($s = 1$ describes the best possible score value): $d_f \in \{1, 2, 3, 4, 5\}$ describes the dimension of the f_{ackley} reference cost function, and $d_f \in \{1, 2, 4, 8, 12\}$ describes the degree of polynomial expansion used.

Table 37 summarizes the mean and the standard deviation values of the regression scores calculated for the cost models learned from the data generated by the reference cost function $f_{rastrigin}$. The parameter $d_f \in \{1, 2, 4, 8, 12\}$ describes the degree of polynomial expansion used. The scoring method used is the coefficient of determination. For the dimensions $d_f \in \{1, 2, 3, 4\}$ a polynomial cost model c_{ij} of degree $d_f = 12$ has the best regression score value and for the dimension $d_f = 5$ a polynomial cost model of degree $d_f = 2$. A polynomial cost model c_{ij} of degree $d_p = 12$ has the best regression score value for the dimensions $d_f \in \{1, 2, 3, 4\}$: $d_f = 1 : s =$ 0.9670 (+/- 0.0002), $d_f = 2 : s =$ 0.9678 (+/- 0.0003), $d_f = 3 : s =$ 0.9657 (+/- 0.0004), and $d_f = 4 : s =$ 0.9633 (+/- 0.0005). A polynomial cost model c_{ij} of degree $d_p = 2$ has the best regression score value of $s =$ 0.9602 (+/- 0.0005) for the dimension $d_f = 5$.

Table 38 summarizes the mean and the standard deviation values of the regression scores calculated for the cost models learned from the data generated by the reference cost function f_{sphere}. The parameter $d_f \in \{1, 2, 4, 8, 12\}$ describes the degree of polynomial expansion used. The scoring method used is the coefficient of determination. A polynomial

	$d_f = 1$	$d_f = 2$	$d_f = 3$	$d_f = 4$	$d_f = 5$
$d_p = 1$	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0000)	-0.0001 (+/- 0.0001)	-0.0002 (+/- 0.0001)	-0.0002 (+/- 0.0001)
$d_p = 2$	0.9467 (+/- 0.0007)	0.9469 (+/- 0.0004)	0.9470 (+/- 0.0006)	0.9469 (+/- 0.0008)	**0.9466 (+/- 0.0007)**
$d_p = 4$	0.9467 (+/- 0.0007)	0.9469 (+/- 0.0004)	0.9470 (+/- 0.0006)	0.9469 (+/- 0.0008)	0.9465 (+/- 0.0007)
$d_p = 8$	0.9468 (+/- 0.0007)	0.9471 (+/- 0.0004)	0.9470 (+/- 0.0006)	0.9468 (+/- 0.0008)	0.9459 (+/- 0.0008)
$d_p = 12$	**0.9478 (+/- 0.0007)**	**0.9480 (+/- 0.0003)**	**0.9478 (+/- 0.0006)**	**0.9469 (+/- 0.0007)**	0.9420 (+/- 0.0006)

Table 37 Mean and standard deviation of regression score values $s \in [-\infty, 1]$ of cost models learned from data generated by the $f_{rastrigin}$ reference cost function ($s = 1$ describes the best possible score value): $d_f \in \{1, 2, 3, 4, 5\}$ describes the dimension of the $f_{rastrigin}$ reference cost function, and $d_f \in \{1, 2, 4, 8, 12\}$ describes the degree of polynomial expansion used.

cost model c_{ij} of degree $d_p = 2$ has the best regression score value for the dimensions $d_f \in \{1, 2, 3, 4, 5\}$: $d_f = 1 : s =$ 1, $d_f = 2 : s =$ 1, $d_f = 3 : s =$ 1, $d_f = 4 : s =$ 1, and $d_f = 5 : s =$ 1.

	$d_f = 1$	$d_f = 2$	$d_f = 3$	$d_f = 4$	$d_f = 5$
$d_p = 1$	-0.0001 (+/- 0.0000)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0002)	-0.0001 (+/- 0.0001)
$d_p = 2$	**1.0000 (+/- 0.0000)**	**1.0000 (+/- 0.0000)**	**1.0000 (+/- 0.0000)**	**1.0000 (+/- 0.0000)**	**1.0000 (+/- 0.0000)**
$d_p = 4$	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)
$d_p = 8$	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)
$d_p = 12$	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)	1.0000 (+/- 0.0000)

Table 38 Mean and standard deviation of regression score values $s \in [-\infty, 1]$ of cost models learned from data generated by the f_{sphere} reference cost function ($s = 1$ describes the best possible score value): $d_f \in \{1, 2, 3, 4, 5\}$ describes the dimension of the f_{sphere} reference cost function, and $d_f \in \{1, 2, 4, 8, 12\}$ describes the degree of polynomial expansion used.

Table 39 summarizes the mean and the standard deviation values of the regression scores calculated for the cost models learned from the data generated by the reference cost function $f_{griewangk}$. The parameter $d_f \in \{1, 2, 4, 8, 12\}$ describes the degree of polynomial expansion used. The scoring method used is the coefficient of determination. A polynomial cost model c_{ij} of degree $d_p = 12$ has the best regression score value for the dimensions $d_f \in \{1, 2, 3, 4\}$: $d_f = 1 : s =$ 0.9995 (+/- 0.0000), $d_f = 2 : s =$ 0.5691(+/- 0.0037), $d_f = 3 : s =$ 0.1467(+/- 0.0048), and $d_f = 4 : s =$ 0.0130(+/- 0.0039). A polynomial cost model c_{ij} of degree $d_p = 2$ has the best regression score value of $s =$ 0.0076(+/- 0.0004) for the dimension $d_f = 5$.

Proof 3 examines sub-hypothesis H2.3. Sub-hypothesis H2.3 describes that polynomial ex-

	$d_f = 1$	$d_f = 2$	$d_f = 3$	$d_f = 4$	$d_f = 5$
$d_p = 1$	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)	-0.0001 (+/- 0.0001)
$d_p = 2$	0.0631 (+/- 0.0012)	0.0033 (+/- 0.0010)	0.0008 (+/- 0.0005)	0.0037 (+/- 0.0008)	**0.0076 (+/- 0.0004)**
$d_p = 4$	0.2552 (+/- 0.0038)	0.0173 (+/- 0.0019)	0.0007 (+/- 0.0006)	0.0031 (+/- 0.0006)	0.0061 (+/- 0.0005)
$d_p = 8$	0.8175 (+/- 0.0005)	0.1692 (+/- 0.0043)	0.0149 (+/- 0.0013)	-0.0010 (+/- 0.0016)	-0.0070 (+/- 0.0025)
$d_p = 12$	**0.9995 (+/- 0.0000)**	**0.5691 (+/- 0.0037)**	**0.1467 (+/- 0.0048)**	**0.0130 (+/- 0.0039)**	-0.0965 (+/- 0.0057)

Table 39 Mean and standard deviation regression score values $s \in [-\infty, 1]$ of cost models learned from data generated by the $f_{griewangk}$ reference cost function ($s = 1$ describes the best possible score value): $d_f \in \{1, 2, 3, 4, 5\}$ describes the dimension of the $f_{griewangk}$ reference cost function, and $d_f \in \{1, 2, 4, 8, 12\}$ describes the degree of polynomial expansion used.

pansion can be used to model nonlinear cost models. Note that this proof is based on empirical results.

Proof 3 (Evidence of H2.3):

Proof. Polynomial expansion can be used to model nonlinear cost models.

1. The results in Table 36 show that polynomial expansion can be used to create a nonlinear cost model of data from the f_{ackley} reference cost function.

2. The results in Table 37 show that polynomial expansion can be used to create a nonlinear cost model of data from the $f_{rastrigin}$ reference cost function.

3. The results in Table 38 show that polynomial expansion can be used to create a nonlinear cost model of data from the f_{sphere} reference cost function.

4. The results in Table 39 show that polynomial expansion can be used to create a nonlinear cost model of data from the $f_{griewangk}$ reference cost function.

From Descriptions 1 - 4, it follows that the sub-hypothesis H2.3, which describes that polynomial expansion can be used to model nonlinear cost models, is valid. □

9.2. Improving Cost Models

This section describes the evaluation of the CyberOpt-ICM subalgorithm, see Section 6.5. The motivation of the CyberOpt-ICM subalgorithm is to calculate a new valid command signal configuration $X = (\bar{P}, B)$, which should be evaluated to obtain more observations of the production system. The more observations, the more accurate cost models. Black-box optimization approaches use this technique to gather additional data. They iterate between fitting cost models and gathering additional observations. Sub-hypothesis H2.4 describes that the expected improvement criterion can be used to enhance the quality of regression models. This sub-hypothesis should be proven to prove hypothesis H2.

The evaluation of the CyberOpt-ICM subalgorithm is performed by the following experimental setup. The expected improvement criterion can be used to calculate new valid command signal configurations. An exhaustive evaluation of a production system is not suitable. Each evaluation of a production system takes time and is therefore expansive. Few evaluations shorten the time in which a production system does not operate optimally. The f_{ackley} reference cost function described in Definition 28, the $f_{rastrigin}$ reference cost function described in Definition 29, the f_{sphere} reference cost function described in Definition 30, and the $f_{griewangk}$ reference cost function described in Definition 31 are used to generate synthetic cost values C^*_{ij}. The experiments are described by the following list of steps:

Step 1 (Generating command signal configurations): In the first step, 10 command signal configurations are generated by a Latin hypercube design [San+].

Step 2 (Fitting and predicting a Gaussian process): In the second step, regression is used to fit a Gaussian process GP to the cost model training data set C^*_{ij}. The training data is generated from the reference cost functions.

Step 3 (Calculation of expected improvement criterion): In the third step, an expected improvement criterion value is calculated for each generated parameter configuration.

Step 4 (Selecting command signal configuration): In the last step, the command signal configuration with the highest expected improvement criterion value is used because this configuration should be evaluated to enhance the quality of the cost model c_{ij}.

Step 2 - 4 are repeated 200 times. Each experiment is repeated 20 times.

9.2.1. Evidence of Sub-Hypothesis H2.4

The following sections describe the results of expected improvement criterion and the reference cost functions f_{ackley}, $f_{rastrigin}$, f_{sphere}, and$f_{griewangk}$.

The best results are archived with 200 iterations. Table 40 summarizes the mean and stan-

dard deviation of optimal (minimum) values of the calculated cost models learned from data generated with the reference cost function f_{ackley} with 40, 80, 160, and 200 samples. The distances for each dimension between the optimal value of the reference cost function f_{ackley} and the learned cost model c_{ij} are 0.3654 (+/- 0.4386) for the dimension $d_f = 1$, 3.2367 (+/- 1.0766) for dimension $d_f = 2$, 5.0161 (+/- 1.1086) for dimension $d_f = 3$, 5.8550 (+/- 1.2835) for dimension $d_f = 4$, and 6.5893 (+/- 1.4243) for dimension $d_f = 5$.

Samples	$d_f = 1$	$d_f = 2$	$d_f = 3$	$d_f = 4$	$d_f = 5$
40	13.0059 (+/- 5.0763)	12.2761 (+/- 4.6312)	15.8849 (+/- 2.9742)	16.7408 (+/- 2.3318)	17.4374 (+/- 1.2039)
80	13.3478 (+/- 5.4959)	14.5044 (+/- 3.6069)	14.9196 (+/- 3.2187)	16.5143 (+/- 1.5653)	17.3778 (+/- 0.8343)
120	12.4026 (+/- 5.2946)	15.0863 (+/- 2.6976)	14.8109 (+/- 2.3751)	15.3758 (+/- 2.1839)	16.8097 (+/- 0.9060)
160	13.0219 (+/- 5.9075)	14.3049 (+/- 2.1603)	14.2563 (+/- 3.5148)	15.7681 (+/- 1.2912)	16.3185 (+/- 2.2794)
200	**0.3654 (+/- 0.4386)**	**3.2367 (+/- 1.0766)**	**5.0161 (+/- 1.1086)**	**5.8550 (+/- 1.2835)**	**6.5893 (+/- 1.4243)**

Table 40 Mean and standard deviation of the optimal (minimum) values of cost models learned from data generated by the f_{ackley} reference cost function with 40, 80, 160, and 200 samples (command signal configurations): $d_f \in \{1, 2, 3, 4, 5\}$ describes the dimensions of the f_{ackley} reference cost function.

The best results are archived with 200 iterations. Table 41 summarizes the mean and the standard deviation of optimal (minimum) values of calculated cost models learned from data generated with the reference cost function $f_{rastrigin}$ with 40, 80, 160, and 200 samples. The distances for each dimension between the optimal value of the reference cost function $f_{rastrigin}$ and the learned cost model c_{ij} are 0.5913 (+/- 0.5355) for the dimension $d_f = 1$, 6.7001 (+/- 4.2751) for dimension $d_f = 2$, 18.4839 (+/- 6.9347) for dimension $d_f = 3$, 41.5583 (+/- 10.1736) for dimension $d_f = 4$, and 64.5394 (+/- 9.5546) for dimension $d_f = 5$.

Samples	$d_f = 1$	$d_f = 2$	$d_f = 3$	$d_f = 4$	$d_f = 5$
40	35.7112 (+/- 22.9366)	91.9611 (+/- 50.9549)	143.0117 (+/- 38.3754)	180.8194 (+/- 47.4205)	203.1626 (+/- 67.9034)
80	48.2276 (+/- 28.4287)	99.9021 (+/- 37.6711)	139.7111 (+/- 43.0305)	154.9491 (+/- 40.7630)	209.2135 (+/- 54.3193)
120	61.2737 (+/- 33.5985)	87.1851 (+/- 40.2845)	127.5408 (+/- 49.0087)	175.8328 (+/- 67.6486)	224.3829 (+/- 51.0432)
160	47.0781 (+/- 29.7800)	76.4048 (+/- 41.2929)	136.4197 (+/- 55.9365)	165.7892 (+/- 69.3931)	221.2368 (+/- 75.0243)
200	**0.5913 (+/- 0.5355)**	**6.7001 (+/- 4.2751)**	**18.4839 (+/- 6.9347)**	**41.5583 (+/- 10.1736)**	**64.5394 (+/- 9.5546)**

Table 41 Mean and standard deviation of the optimal (minimum) values of cost models learned from data generated by the $f_{rastrigin}$ reference cost function with 40, 80, 160, and 200 samples (command signal configurations): $d_f \in \{1, 2, 3, 4, 5\}$ describes the dimensions of the $f_{rastrigin}$ reference cost function.

The best results are archived with 200 iterations. Table 42 summarizes the mean and the

standard deviation of optimal (minimum) values of the calculated cost models learned from data generated with the reference cost function f_{sphere} with 40, 80, 160, and 200 samples. The distances for each dimension between the optimal value of the f_{sphere} reference cost function and the learned cost model c_{ij} are 0.0054 (+/- 0.0088) for the dimension $d_f = 1$, 0.7926 (+/- 0.7430) for dimension $d_f = 2$, 4.5367 (+/- 3.5937) for dimension $d_f = 3$, 10.7288 (+/- 4.8807) for dimension $d_f = 4$, and 22.2899 (+/- 11.0030) for dimension $d_f = 5$.

Samples	$d_f = 1$	$d_f = 2$	$d_f = 3$	$d_f = 4$	$d_f = 5$
40	35.2736 (+/- 28.4845)	63.6836 (+/- 44.0134)	109.5623 (+/- 41.7751)	122.4817 (+/- 44.9136)	161.4139 (+/- 52.5357)
80	28.1721 (+/- 23.0162)	73.1547 (+/- 39.7185)	102.9314 (+/- 57.7892)	112.4325 (+/- 52.5923)	139.7518 (+/- 62.5227)
120	37.9670 (+/- 35.6007)	68.2701 (+/- 51.9483)	101.0767 (+/- 46.4419)	134.3414 (+/- 40.9701)	187.3160 (+/- 50.3305)
160	46.8398 (+/- 33.5332)	47.9186 (+/- 32.1122)	94.8047 (+/- 53.5147)	132.3483 (+/- 57.3794)	159.1670 (+/- 63.2687)
200	**0.0054 (+/- 0.0088)**	**0.7926 (+/- 0.7430)**	**4.5367 (+/- 3.5937)**	**10.7288 (+/- 4.8807)**	**22.2899 (+/- 11.0030)**

Table 42 Mean and standard deviation of the optimal (minimum) values of cost models learned from data generated by the f_{sphere} reference cost function with 40, 80, 160, and 200 samples (command signal configurations): $d_f \in \{1, 2, 3, 4, 5\}$ describes the dimensions of the f_{sphere} reference cost function.

The best results are archived with 200 iterations. Table 43 summarizes the mean and standard deviation of optimal (minimum) values of the calculated cost models learned from data generated with the reference cost function $f_{griewangk}$ with 40, 80, 160, and 200 samples.

Samples	$d_f = 1$	$d_f = 2$	$d_f = 3$	$d_f = 4$	$d_f = 5$
40	0.0049 (+/- 0.0049)	0.3404 (+/- 0.6164)	0.3555 (+/- 0.5273)	0.5848 (+/- 0.5504)	0.7895 (+/- 0.4936)
80	0.0044 (+/- 0.0049)	0.0169 (+/- 0.0106)	0.3072 (+/- 0.4826)	0.7143 (+/- 0.5325)	0.4916 (+/- 0.4725)
120	0.0030 (+/- 0.0045)	0.0169 (+/- 0.0106)	0.3110 (+/- 0.5406)	0.6913 (+/- 0.4868)	0.7649 (+/- 0.5943)
160	0.0030 (+/- 0.0045)	0.0168 (+/- 0.0106)	0.2683 (+/- 0.4502)	0.6441 (+/- 0.4219)	0.6019 (+/- 0.4744)
200	**0.0025 (+/- 0.0043)**	**0.0168 (+/- 0.0106)**	**0.0440 (+/- 0.0541)**	**0.0428 (+/- 0.0494)**	**0.0748 (+/- 0.1061)**

Table 43 Mean and standard deviation of the optimal (minimum) values of cost models that are learned from data generated by the $f_{griewangk}$ reference cost function with 40, 80, 160, and 200 samples (command signal configurations): $d_f \in \{1, 2, 3, 4, 5\}$ describes the dimensions of the $f_{griewangk}$ reference cost function.

The distances for each dimension between the optimal value of the $f_{griewangk}$ reference cost function and the learned cost model c_{ij} are 0.0025 (+/- 0.0043) for the dimension $d_f = 1$, 0.0168 (+/- 0.0106) for the dimension $d_f = 2$, 0.0440 (+/- 0.0541) for the dimension $d_f = 3$, 0.0428 (+/- 0.0494) for the dimension $d_f = 4$, and 0.0748 (+/- 0.1061) for the dimension $d_f = 5$.

Proof 4 examines sub-hypothesis H2.4. Sub-hypothesis H2.4 describes that the expected improvement criterion can be used to enhance the quality of regression models. Note that this proof is based on empirical results.

Proof 4 (Evidence of H2.4):

Proof. The expected improvement criterion can be used to enhance the quality of cost models.

1. The results in Table 20 show that the expected improvement criterion can be used to enhance the quality of the cost models of the storage scenario.

2. The results in Table 27 show that the expected improvement criterion can be used to enhance the quality of the cost models of the pick-and-place scenario.

3. The results in Table 40 show that the expected improvement criterion can be used to enhance the quality of cost models learned from data generated by the f_{ackley} reference cost function.

4. The results in Table 41 show that the expected improvement criterion can be used to enhance the quality of cost models learned from data generated by the $f_{rastrigin}$ reference cost function.

5. The results in Table 42 show that the expected improvement criterion can be used to enhance the quality of cost models learned from data generated by the f_{sphere} reference cost function.

6. The results in Table 43 show that the expected improvement criterion can be used to enhance the quality of cost models learned from data generated by the $f_{griewangk}$ reference cost function.

From Descriptions 1 - 6, it follows that the sub-hypothesis H2.4, which describes that the expected improvement criterion enhances the quality of cost models, is valid. □

9.3. Learning Sequences of Control Methods

This section describes the evaluation of the CyberOpt-LSC subalgorithm, see Section 6.3. The motivation of the CyberOpt-LSC algorithm is to learn sequences of control methods from an event log automatically. Sequences of control methods are required to solve the command signal configuration problem. Manual creation of a behavior model is not required, reducing manual engineering effort and synchronizing the learned behavior model with the current state of the production system. Sub-hypotheses H3.1 and H3.2 should be proven to prove hypothesis H3. Sub-hypothesis H3.1 describes that event logs of activities can be used to

learn a behavior model, and sub-hypothesis H3.2 describes that the fuzzy mining algorithm can be used to learn a behavior model from an event log.

9.3.1. Evidence of Sub-Hypothesis H3.1

Event logs of activities include information about the number of evaluations of the production system, activities, originator hardware components, and time values. The basic concept of process mining is to extract information about processes from event logs recorded by an information system. An event log contains events, and each event has a timestamp and refers to an activity and a case, e.g., a process instance. An activity defines a specific step in the process. Control-flow mining algorithms directly create a dependency graph (directed graph) from event logs. In order to prove sub-hypothesis H3.1, which describes that event logs of activities can be used to learn a behavior model, knowledge of event logs is analyzed, more precisely the ability to extract behavior and parallel behavior of an event log.

Table 44 describes an event log of the storage scenario, see Section 7.2.

Case	Activity	Start date	End date	Resource
1	A_1	2020 - 01 - 01 09:00:00	2020 - 01 - 01 09:02:00	M_1
1	A_2	2020 - 01 - 01 09:02:00	2020 - 01 - 01 09:05:00	M_2
1	A_3	2020 - 01 - 01 09:05:00	2020 - 01 - 01 09:07:00	M_3
2	A_1	2020 - 02 - 01 09:00:00	2020 - 02 - 01 09:02:00	M_1
2	A_2	2020 - 02 - 01 09:02:00	2020 - 02 - 01 09:05:00	M_2
...	...	...	...	...

Table 44 Event log of the storage scenario; hardware components: M_1, M_2 and M_3; activities: A_1 realizes $drive\text{-}F$, A_2 realizes $drive\text{-}C$, and A_3 realizes $drive\text{-}A$.

The behavior model is described by the following activities: $A_1 : drive\text{-}F$ (hardware component M_1), $A_2 : drive\text{-}C$ (hardware component M_2), and $A_3 : drive\text{-}A$ (hardware component M_3) are performed sequentially. Figure 41 illustrates the event log of Table 44. A data point (activity) is shown for each start and end date. Horizontal lines represent the start of a new activity. Knowledge about behavior can be extracted from the event log.

Table 45 describes an event log of the pick-and-place scenario, see Section 7.3.

Case	Activity	Start date	End date	Resource
1	A_1	2020 - 01 - 01 08:00:00	2020 - 01 - 01 08:05:00	M_1
1	A_2	2020 - 01 - 01 08:00:00	2020 - 01 - 01 08:04:00	M_2
1	A_3	2020 - 01 - 01 08:00:00	2020 - 01 - 01 08:03:00	M_3
1	A_4	2020 - 01 - 01 08:05:00	2020 - 01 - 01 08:10:00	M_2
1	A_5	2020 - 01 - 01 08:10:00	2020 - 01 - 01 08:12:00	M_1
1	A_6	2020 - 01 - 01 08:10:00	2020 - 01 - 01 08:11:00	M_2
1	A_7	2020 - 01 - 01 08:12:00	2020 - 01 - 01 08:15:00	M_2

Table 45 Event log of the pick-and-place scenario; hardware components: a conveyor system M_1, a robot M_2, and a compressor M_3; activities: A_1 realizes $drive\text{-}B$, A_2 realizes $drive\text{-}B$, A_3 realizes $produce\text{-}P$, A_4 realizes $take\text{-}B$, A_5 realizes $drive\text{-}A$, A_6 realizes $drive\text{-}C$, and A_7 realizes $drop\text{-}C$.

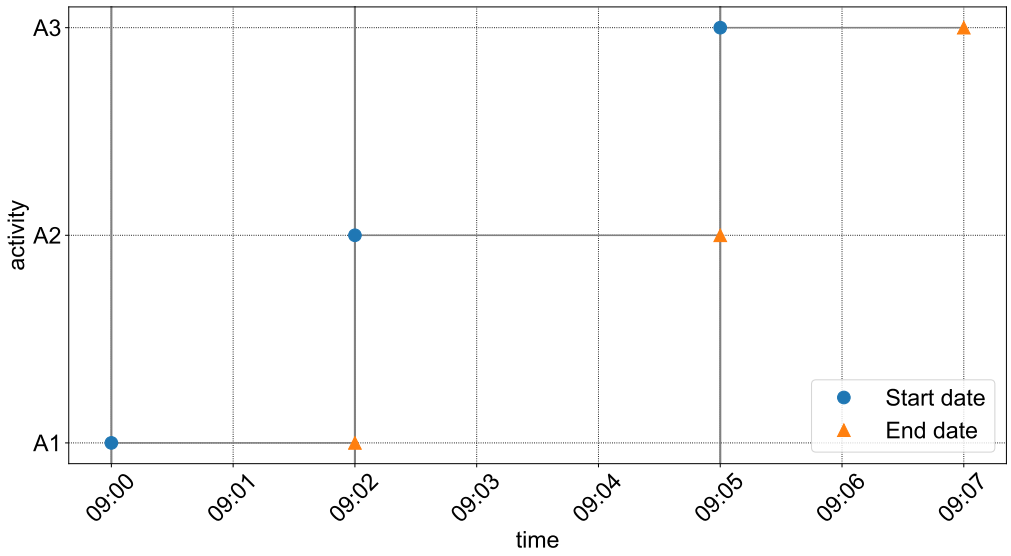


Figure 41 Illustration of the storage event log, see Table 44: Activities A1 - A3 are mapped to $A_1 - A_3$, A_1 realizes $drive\text{-}F$, A_2 realizes $drive\text{-}C$, and A_3 realizes $drive\text{-}A$.

The behavior model is described by the following activities: $A_1 : drive\text{-}B$ (conveyor system M_1), $A_2 : drive\text{-}B$ (robot M_2), and $A_3 : produce\text{-}P$ (compressor M_3) are performed at the same time, then $A_4 : take\text{-}B$ (robot M_2) and $A_5 : drive\text{-}A$ (conveyor system M_1) are performed, and finally $A_6 : drive\text{-}C$ and $A_7 : drop\text{-}C$ (robot M_2) are performed.

Figure 42 illustrates the event log of Table 45. A data point (activity) is shown for each start and end date. All activities between two horizontal lines are performed in parallel.

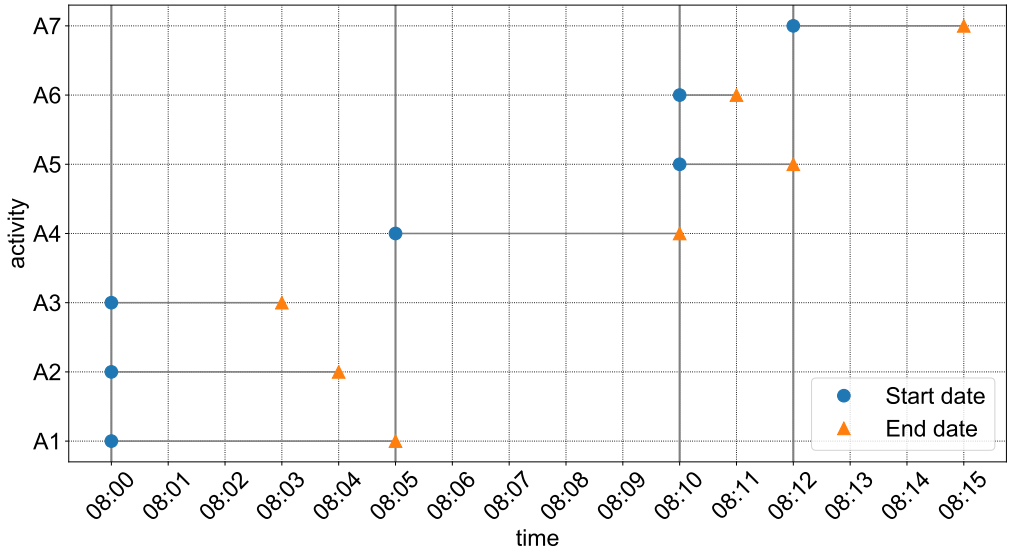


Figure 42 Illustration of the pick-and-place event log (see Table 45): A1 - A7 map to $A_1 - A_7$, A_1 realizes $drive\text{-}B$, A_2 realizes $drive\text{-}B$, A_3 realizes $produce\text{-}P$, A_4 realizes $take\text{-}B$, A_5 realizes $drive\text{-}A$, A_6 realizes $drive\text{-}C$, and A_7 realizes $drop\text{-}C$.

Proof 5 examines sub-hypothesis H3.1. Sub-hypothesis H3.1 describes that event logs of activities can be used to learn a behavior model. Note that this proof is based on empirical results.

Proof 5 (Evidence of H3.1):

Proof. Event logs of activities can be used to learn a behavior model.

1. Figure 41 shows that an event log of activities can be used to extract behavior.

2. Figure 42 shows that an event log of activities can be used to extract parallel behavior.

From Descriptions 1 and 2, it follows that the sub-hypothesis H3.1, which describes that event logs of activities can be used to learn a behavior model, is valid. □

9.3.2. Evidence of Sub-Hypothesis H3.2

The basic concept of process mining is to extract information about processes from transaction logs recorded by an information system [vdAal+03]. A transaction log contains events, and each event has a timestamp and refers to an activity and a case, e.g., a process instance. An activity defines a specific step in the process. The fuzzy mining algorithm described in [GV07] implements adaptive simplifications based on data clustering and graph clustering, e.g., to solve the concurrency problem. The problem occurs when event A and event B can be observed in any order, e.g., they are on two distinct parallel paths, then the transaction log contains both possible cases: event A is followed by event B and vice versa.

Figure 43 illustrates the fuzzy mining algorithm result for the storage scenario.

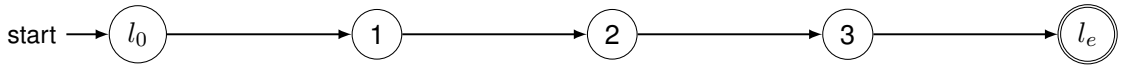


Figure 43 Fuzzy mining result of the storage scenario.

The directed graph of activities is defined as a quadruple (G, l_0, l_e, T). The finite set G describes the nodes of the directed graph of activities, where l_0 describes the start node and l_e describes the end node. $T \subseteq G \times G$ describes the transitions between nodes. Each node has an activity index i that refers to a software component A_i.

Figure 44 illustrates the fuzzy mining algorithm result for the pick-and-place scenario.

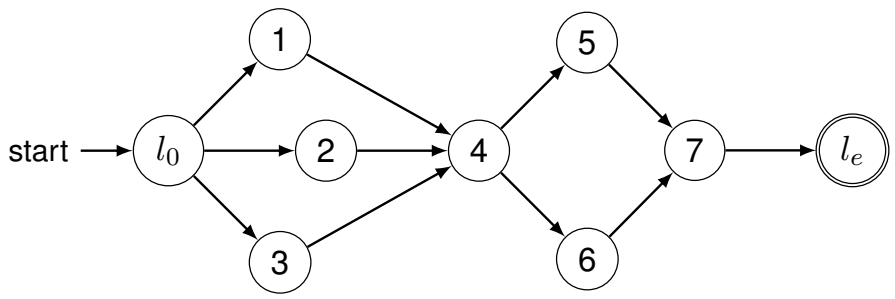


Figure 44 Fuzzy mining result of the pick-and-place scenario.

Proof 6 examines sub-hypothesis H3.2. Sub-hypothesis H3.2 describes that the fuzzy mining

algorithm can be used to learn a behavior model from an event log. Note that this proof is based on empirical results.

Proof 6 (Evidence of H3.2):

Proof. The fuzzy mining algorithm can be used to learn a behavior model.

1. In [GV07], it is shown that the fuzzy mining algorithm can be used to learn a behavior model from an event log.

2. Figure 43 shows that a behavior model for the storage scenario can be learned with the fuzzy mining algorithm.

3. Figure 44 shows that a behavior model for the pick-and-place scenario can be learned with the fuzzy mining algorithm.

From Descriptions 1 - 3, it follows that the sub-hypothesis H3.2, which describes that the fuzzy mining algorithm can be used to learn a behavior model from an event log, is valid. □

9.4. Solving the Command Signal Configuration Problem

This section describes the evaluation of the CyberOpt-SPC subalgorithm. The motivation of the CyberOpt-SPC algorithm is to automatically find optimal command signal configurations by solving the command signal configuration problem.

Sub-hypotheses H4.1, H4.2, and H4.3 should be proven to prove hypothesis H4. Sub-hypothesis H4.1 describes that human-in-the-loop and sampling are unsuitable for finding an optimal command signal configuration, sub-hypothesis H4.2 describes that a general mixed-integer nonlinear programming solver can be used to find an optimal command signal configuration, and sub-hypothesis H4.3 describes that the command signal configuration problem is in the complexity class NP-hard.

The evaluation of the CyberOpt-SIC subalgorithm is performed with the following experimental setup. The behavior model generator creates a scenario with n_p paths with n_a activities and n_c control methods, see Section 7.4.2.

9.4.1. Evidence of Sub-Hypothesis H4.1:

Human-in-the-loop describes a process in which a domain expert selects a plausible command signal configuration according to his domain knowledge. The human-in-the-loop is simulated using a heuristic that samples uniform random values. Sampling and validation describe a process in which a heuristic samples uniform random values to select a control

method j for each software component A_i and uniform random values for command signals for the selected control methods.

Table 46 summarizes the results from the behavior model generator. Each experiment is repeated 10000 times. The probability of finding an optimal command signal configuration with one path $n_p = 1$, one activity $n_a = 1$, and one control method $n_c = 1$ is 49.47%.

ID	Paths	Activities	Methods	Probability	Success
1	1	1	1	0.4947	1.0
2	2	1	1	0.3652	1.0
3	4	1	1	0.2605	1.0
4	8	1	1	0.1985	0.8
5	16	1	1	0.1707	0.6
6	1	2	1	0.4996	1.0
7	1	4	1	0.4993	0.95
8	1	8	1	0.5	0.85
9	1	16	1	0.5002	0.775
10	1	1	2	0.4973	1.0
11	1	1	4	0.4973	1.0
12	1	1	8	0.5016	1.0
13	1	1	16	0.4973	1.0
14	2	2	2	0.3924	1.0
15	4	4	4	0.3584	0.7
16	8	8	8	0.3584	0.55
17	16	16	16	0.3742	0.5

Table 46 Results from the behavior model generator.

Variation of paths: If the paths are varied ($n_p = 2$, $n_p = 4$, $n_p = 8$, and $n_p = 16$), then the probability of sampling an optimal command signal configuration is reduced to 36.52% for 2 paths, 26.05% for 4 paths, 19.85% for 8 paths, and 17.07% for 16 paths. The success rate that describes the percentage of optimal command signal configurations is reduced from 100% to 60%.

Variation of activities: If the activities are varied ($n_a = 2$, $n_a = 4$, $n_a = 8$, and $n_a = 16$), then the probability of sampling an optimal command signal configuration is 49.96% for 2 activities, 49.93% for 4 activities, 50% for 8 activities and 50.02% for 16 activities. The success rate that describes the percentage of optimal command signal configurations is reduced from 100% to 75.75%.

Variation of methods: If the control methods are varied ($n_c = 2$, $n_c = 4$, $n_c = 8$ and $n_c = 16$), then the probability of sampling an optimal command signal configuration is 49.73% for 2 control methods, 49.73% for 4 control methods, 50.16% for 8 control methods and 49.73% for 16 control methods. The success rate that describes the percentage of optimal command signal configurations is not reduced.

Variation of paths, activities and methods: If paths n_p, activities n_a and methods n_c are varied, then the probability is between 35.84% and 39.24%. The success rate is between 50% and 100%.

Figure 45 summarizes the different generator configurations and the corresponding success rates for each calculated time value.

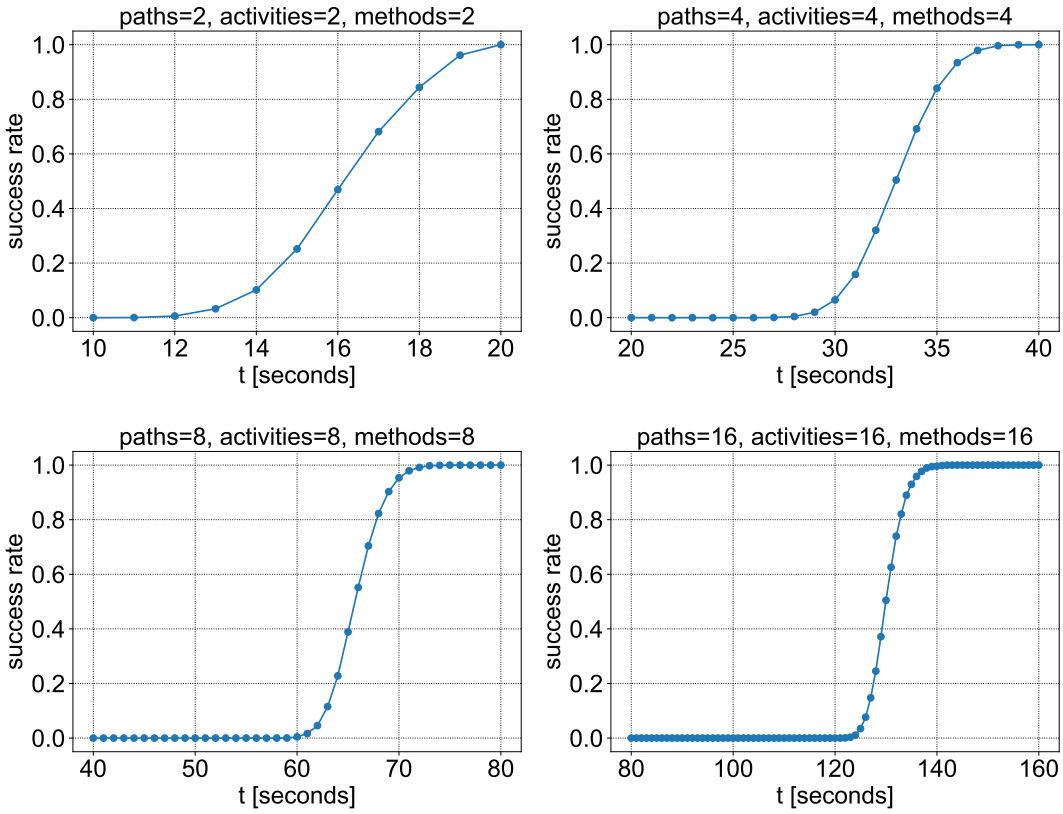


Figure 45 Success rates of sampling.

Proof 7 examines sub-hypothesis H4.1. Sub-hypothesis H4.1 describes that human-in-the-loop and sampling are not suitable for finding an optimal command signal configuration. Note that this proof is based on empirical results.

Proof 7 (Evidence of H4.1):

Proof. Sampling is not suitable for finding an optimal command signal configuration.

The results in Table 46 and Figure 45 show that sampling is not able to find an optimal command signal configurations. Sub-hypothesis H4.1, which describes that human-in-the-loop and sampling are not suitable for finding an optimal command signal configuration, is valid. □

9.4.2. Proof of Sub-Hypothesis H4.2

Proof 8 examines sub-hypothesis H4.2. Sub-hypothesis H4.2 describes that a general mixed-integer nonlinear programming solver can be used to find an optimal command signal config-

uration.

Proof 8 (Proof of H4.2):

Proof. A general mixed-integer nonlinear programming solver can be used to find an optimal command signal configuration.

The command signal configuration problem is defined as a mixed-integer nonlinear programming problem with a nonlinear objective function, linear constraints, and a mixture of continuous and binary variables. A general mixed-integer nonlinear programming solver can be used to solve the command signal configuration problem. Therefore, the sub-hypothesis H4.2 is valid. □

9.4.3. Proof of Sub-Hypothesis H4.3

In the following proof, the complexity class of the command signal configuration problem is analyzed; more precisely, it is proven that the command signal configuration problem is NP-hard. The proof is published in [OVN16]. Proof 9 examines sub-hypothesis H4.3. A similar problem is the multiple-choice knapsack problem defined in Definition 6, where the goal is to choose exactly one item j from each of k classes $N_i, i = 1, \ldots, k$, such that the profit sum is maximized. The multiple-choice knapsack problem is NP-hard [KPP04]. The idea is to show that one special case of a command signal configuration problem is a multiple-choice knapsack problem. If this is the case, the problem is NP-hard.

Proof 9 (Proof of H4.3):

Proof. The multiple-choice knapsack problem is NP-hard. The command signal configuration problem is NP-hard if the command signal configuration problem $\geq$ the multiple-choice knapsack problem. If a special case exists where the command signal configuration problem = multiple-choice knapsack problem, then the command signal configuration problem $\geq$ the multiple-choice knapsack problem.

1. We know that $\min f(.) = -\max -f(.)$ so that Equation (3.8) and Equation (4.35) are equal if $c_{ij}(.) = p_{ij}$ and all cost models $c_{ij}(.)$ are static.

2. Equation (3.9) and Equation (4.36) are equal if only one parameter for each control method $p_{ij} = w_{ij}$ exists and only one sequence $|\bar{S}| = 1$ exists.

From the above Descriptions 1 and 2, it follows that a special case exists where the command signal configuration problem = the multiple-choice knapsack problem, so that the command signal configuration problem $\geq$ the multiple-choice knapsack problem and the command signal configuration problem is NP-hard. □

10. Evaluation of the CyberOpt Algorithm

This chapter describes the evaluation of the CyberOpt algorithm. Section 1.2 describes the requirements R1 - R6 for an optimal command signal configuration, the requirements M1 - M6 for an optimization model, and the requirements B1 - B4 for a behavior model. The requirements for an optimal command signal configuration should be considered, as well as the requirements for an optimization model and the requirements for a behavior model. CyberOpt automates the following two tasks: (1) selection of optimal control methods from software components of an automation software structure that implement different control strategies and (2) calculation of optimal command signals for selected control strategies.

Chapter 4 describes the command signal configuration problem that must be solved by CyberOpt in order to find an optimal configuration. An optimal command signal configuration is one with a minimum cost value. Cost values in this work are energy consumption values. Chapter 6 describes the CyberOpt algorithm that uses the following machine learning techniques to reduce manual engineering effort: a machine learning technique of regression and polynomial expansion to learn cost models from production system observations (T1), the expected improvement criterion to calculate new valid command signal configurations that should be evaluated in order to obtain more observations of the production system (T2), a machine learning technique called process mining to learn sequences of control methods from production system observations (T3), and mixed-integer nonlinear programming to solve the command signal configuration problem (T4). The CyberOpt algorithm consists of five subalgorithms: (1) CyberOpt-SIC subalgorithm described in Section 6.1, (2) CyberOpt-LCM subalgorithm described in Section 6.2, (3) CyberOpt-LSC subalgorithm described in Section 6.3, (5) CyberOpt-SPC subalgorithm described in Section 6.4, and (4) CyberOpt-ICM subalgorithm described in Section 6.5. Chapter 7 describes application scenarios from discrete manufacturing. For each scenario, the automation software structure, command signals, command signal constraints, and sequences of control methods are described. Chapter 8 evaluates the CyberOpt algorithm. Optimal command signal configurations are calculated for the application scenarios. For this purpose, the following five experiments are defined: (1) ground truth, (2) random walk, (3) black-box optimization, (4) CyberOpt approach, and (5) CyberOpt MEM. The results from the ground truth experiment are used as reference results. The random walk experiment tries to find an optimal command signal configuration by random sampling. The black-box optimization attempts to find an optimal command signal configuration using a black-box optimization approach. The CyberOpt and CyberOpt MEM experiments attempt to find the optimal command signal configuration using the CyberOpt algorithm. The memory operation mode MEM uses all energy consumption values from previous calculations. Chapter 9 evaluates the CyberOpt subalgorithms. The evaluation aims to prove that the four machine learning techniques reduce the manual engineering effort. For this purpose, four hypotheses are defined. Hypothesis H1 describes that an optimal command signal configuration solution does not require manual engineering steps. Hypothesis

H2 describes that a machine learning algorithm exists that learns cost models that can then be used by a command signal configuration algorithm to find optimal command signal configurations. Hypothesis H3 describes that a machine learning algorithm exists that learns sequences of control methods that can then be used by a command signal configuration algorithm to find optimal command signal configurations. Hypothesis H4 describes that an algorithm exists that finds optimal command signal configurations. Sub-hypotheses are defined to prove the four hypotheses of this work. Table 47 assesses the requirements R1 - R6, M1 - M6, B1 - B4, and the sub-hypotheses H2.1 - H4.3 for each CyberOpt subalgorithm.

ID	CyberOpt-SIC	CyberOpt-LCM	CyberOpt-ICM	CyberOpt-LSC	CyberOpt-SPC	CyberOpt
R1	+	+	+	+	−	+
R2	−	−	+	−	+	+
R3	+	+	−	−	+	+
R4	+	−	−	−	+	+
R5	+	−	−	−	+	+
R6	+	−	−	−	+	+
M1	+	−	−	−	+	+
M2	−	−	−	−	+	+
M3	+	−	−	−	+	+
M4	+	−	−	−	+	+
M5	+	−	−	−	+	+
M6	+	−	−	−	+	+
B1	−	+	−	−	−	+
B2	−	+	−	−	−	+
B3	−	+	−	−	−	+
B4	−	−	−	+	−	+
H2.1	−	+	−	−	−	+
H2.2	−	+	−	−	−	+
H2.3	−	+	−	−	−	+
H2.4	−	−	+	−	−	+
H3.1	−	−	−	+	−	+
H3.2	−	−	−	+	−	+
H4.1	−	−	−	−	+	+
H4.2	−	−	−	−	+	+
H4.3	−	−	−	−	+	+

Table 47 Overview of requirements R1 - R6 for an optimal command signal configuration, requirements M1 - M6 for an optimization model, requirements B1 - B4 for a behavior model, sub-hypotheses H2.1 - H4.3, and their corresponding subalgorithm.

The CyberOpt-SIC subalgorithm automatically calculates valid command signal configurations. It reduces manual engineering steps (R1). Implicit timing parameters are used (R3). Sequences (R4), parallel sequences (R5), and the selection of different control methods (R6) are supported. Valid command signal configurations are required to set up automation software. When an automation software is set up with a valid command signal configuration, energy consumption values can be measured, and an event log can be recorded during the runtime of a production system.

The CyberOpt-LCM subalgorithm automatically learns a cost model for each control method from a software component. Cost models are required to compare different command signal configurations in order to find an optimal command signal configuration. Manual creation of cost models is not required, reducing the manual engineering effort (R1). In Section 9.1, the CyberOpt-LCM subalgorithm is used to learn cost models from data generated by the reference cost functions f_{ackley}, $f_{rastrigin}$, f_{sphere}, and $f_{griewangk}$, see Section 7.4.1. Proof 1 examines sub-hypothesis H2.1, which describes that regression analysis can be used to learn cost models. A cost model can be learned for each reference cost function. The sub-hypothesis H2.1 is valid. Proof 2 examines sub-hypothesis H2.2. For the reference cost functions f_{ackley}, $f_{rastrigin}$, f_{sphere}, and $f_{griewangk}$, a polynomial of degree $d_p = 1$ is not sufficient to predict cost values. This statement is valid for the function dimensions $d_f \in \{1, 2, 3, 4, 5\}$. Sub-hypothesis H2.2, which describes that linear cost models are not sufficient to predict energy consumption values, is valid. Proof 3 examines sub-hypothesis H2.3. For each reference cost function, a nonlinear cost model can be learned. The sub-hypothesis H2.3, which describes that polynomial expansion can be used to model nonlinear cost models, is valid. Note that Proof 1, Proof 2, and Proof 3 are based on empirical results. Each cost model has implicit timing parameters and is used to represent the timing (B1), continuous behavior (B2) and cost (B3) of the behavior model.

The CyberOpt-LSC subalgorithm automatically learns sequences of control methods from an event log. Sequences of control methods are required to solve the command signal configuration problem. Manual creation of a behavior model is not required, reducing manual engineering effort (R1) and synchronizing the learned behavior model with the current state of the production system. Proof 5 described in Section 9.3 examines sub-hypothesis H3.1. Sub-hypothesis H3.1 describes that event logs of activities can be used to learn a behavior model. Using the event log of the storage scenario, it is shown that an event log of activities can be used to extract behavior, and using an event log of the pick-and-place scenario, it is shown that an event log of activities can be used to extract parallel behavior. Proof 6 examines sub-hypothesis H3.2, which describes that the fuzzy mining algorithm can be used to learn a behavior model from an event log. In [GV07], it is shown that this is possible. It is shown that fuzzy mining can also be used to learn sequences of control methods from events logs of the storage scenario and the pick-and-place scenario. The behavior model of the CyberOpt-LSC subalgorithm is defined as a directed graph of activities (B4). The design concept separation of concerns is used. Cost models (B3) are learned from the timing behavior (B1) and continuous behavior (B2). The activities refer to control methods, and each control method has a cost model. Therefore, the behavior model of CyberOpt satisfies all the requirements B1 - B4. Note that the directed graph is only required for internal use of the algorithm to calculate sequences of control methods.

The CyberOpt-SPC subalgorithm solves the command signal configuration problem. A global optimal command signal configuration is found (R2), and implicit timing parameters are used (R3). Sequences (R4), parallel sequences (R5), and the selection of different control meth-

ods (R6) are supported. The command signal configuration problem can have a linear or nonlinear objective function (M1) and (M2), linear or nonlinear constraints (M3) and (M4), continuous (M5), and discrete (M6) variables. An optimal command signal configuration can be calculated by human-in-the-loop, a sampling and validation approach, or by using mixed-integer nonlinear programming solving techniques. Proof 7 described in Section 9.4 examines sub-hypothesis H4.1, which describes that human-in-the-loop and sampling are not suitable for finding an optimal command signal configuration. Experimental results have shown that the probability of finding an optimal command signal configuration is 39.24% for a scenario with $n_p = 2$ paths, $n_a = 2$ activities, and $n_c = 2$ control methods with a success rate of 100%. For a scenario with $n_p = 16$ paths, $n_a = 16$ activities, and $n_c = 16$ control methods, the probability of finding an optimal command signal configuration is 37.42% with a success rate of 50%. By solving the command signal configuration problem, an optimal command signal configuration is found. A nonlinear objective function (M2) and a global optimum (R2) are not required. Proof 8 examines sub-hypothesis H4.2, which describes that a general mixed-integer nonlinear programming solver can be used to find an optimal command signal configuration. Sub-hypothesis H4.1 is valid if the command signal configuration problem is described by at least linear cost models. Note that two cost values are required for a linear cost model. The command signal configuration problem can have a linear or nonlinear objective function (M1) and (M2), linear or nonlinear constraints (M3) and (M4), continuous (M5), and discrete (M6) variables. Proof 9 examines sub-hypothesis H4.3, proving that the command signal configuration problem is NP-hard. The set of NP-hard problems describes problems that are difficult to solve because no polynomial-time algorithm is known, and it is not possible to verify if a solution is a feasible solution in polynomial time. Despite the theoretical fact that the command signal configuration problem falls into the NP-hard complexity class, research was published to implement efficient algorithms to solve these optimization problems. Section 3.2.2 describes several techniques for solving mixed-integer nonlinear programming problems.

The CyberOpt-ICM subalgorithm calculates new valid command signal configurations that should be evaluated in order to obtain more observations of the production system. With more observations, cost models become more accurate, improving a global optimum (R2). In Section 9.2, the CyberOpt-ICM subalgorithm is used to learn cost models from data generated by the reference cost functions f_{ackley}, $f_{rastrigin}$, f_{sphere}, and $f_{griewangk}$. More precisely, the CyberOpt-ICM subalgorithm calculates a new valid command signal configuration that should be evaluated to obtain more observations of the production system. The observations of the production system are simulated by the reference cost functions. Proof 4 examines sub-hypothesis H2.4, which describes that the expected improvement criterion can be used to enhance the quality of regression models. It is shown that the expected improvement criterion can be used to enhance the quality of cost models learned from data generated by the reference functions f_{ackley}, $f_{rastrigin}$, f_{sphere} and $f_{griewangk}$. Note that Proof 4 is based on empirical results.

The CyberOpt algorithm satisfies the requirements R1 - R6 for an optimal command signal configuration, the requirements M1 - M6 for an optimization model, and the requirements B1 - B4 for a behavior model. The main hypothesis H1 of this work describes that an optimal command signal configuration solution does not require manual engineering steps because the vision of cyber-physical production systems is that they adapt to new production goals without extensive manual engineering effort. It could be proven that this hypothesis is valid. Therefore, the following hypotheses and sub-hypotheses are proven:

H2: Given valid command signal configurations and observations from the runtime of a production system, a machine learning algorithm exists that learns cost models that can then be used by a command signal configuration algorithm to find optimal command signal configurations. Hypothesis H2 is valid because sub-hypotheses H2.1, H2.2, H2.3, and H2.4 are valid:

- **H2.1:** Regression analysis can be used to learn cost models (see Proof 1).
- **H2.2:** Linear cost models are not sufficient to predict energy consumption values. Non-linear cost models should be used (see Proof 2).
- **H2.3:** Polynomial expansion can be used to model nonlinear cost models (see Proof 3).
- **H2.4:** The expected improvement criterion can be used to enhance the quality of cost models (see Proof 4).

H3: Given valid command signal configurations and observations from the runtime of the production system, a machine learning algorithm exists that learns sequences of control methods that can then be used by a command signal configuration algorithm to find optimal command signal configurations. Hypothesis H3 is valid because sub-hypotheses H3.1 and H3.2 are valid:

- **H3.1:** Event logs of activities can be used to learn a behavior model (see Proof 5).
- **H3.2:** The fuzzy mining algorithm can be used to learn a behavior model (see Proof 6).

H4: Given a command signal configuration problem, an algorithm exists that finds optimal command signal configurations. Hypothesis H4 is valid, because sub-hypotheses H4.1, H4.2, and H4.3 are valid:

- **H4.1:** Human-in-the-loop and sampling are not suitable for finding an optimal command signal configuration (see Proof 7).
- **H4.2:** A general mixed-integer linear programming solver can be used to find an optimal command signal configuration (see Proof 8).
- **H4.3:** The command signal configuration problem falls into the NP-hard complexity class (see Proof 9).

11. Summary and Outlook

This work introduces a novel approach to machine learning and constrained optimization of command signal configuration named CyberOpt. The approach helps to adapt cyber-physical production systems to new requirements and is not a general-purpose solution for all cyber-physical production systems. CyberOpt can be used for the following three specific production scenarios from the discrete manufacturing domain: (1) transport scenarios, (2) storage scenarios, and (3) pick-and-place scenarios.

The production goal is described by a behavior model that encodes sequences of activities. A software component contains one or more control methods with command signals that implement the behavior of an activity with different control strategies. Command signals increase the reusability of automation software components. Command signals add degrees of freedom to automation software components so that automation software components can be configured for a specific production scenario. A decision parameter describes which control strategy of an automation software component should be used to perform an activity. Only one control strategy can be selected from an automation software component. A timing parameter of a control strategy describes the time period within which the control strategy should perform the activity. Command signals specify the degrees of freedom of control strategy implementations.

The CyberOpt approach automates the following two tasks:

(i) Selection of optimal control methods from software components of an automation software structure that implement different control strategies.

(ii) Calculation of optimal command signals for the selected control strategies.

Without the possibility of feedback from a production system, e.g., in the form of energy consumption observations or time observations, the selection of optimal control strategies from software components of an automation software structure and the calculation of optimal command signals for selected control strategies is not possible. Production system observations are used to calculate cost models, which can then be used to compare different control strategies and different command signal configurations. Since the amount of time and energy consumed are universally observable quantities, no distinctions are made between open-loop control strategies and closed-loop control strategies.

The general idea of adapting automation software components to different control strategies and command signals to specific production scenarios is described by a three-step optimization process. The three-step optimization process consists of three decisions: (1) selecting the optimal sequence of control methods, (2) calculating the optimal timing parameters, and

(3) calculating optimal command signals. Decision 1 describes that an optimal control strategy must be selected for each software component. Cost models from production system observations are used for this purpose. Decision 2 describes that optimal timing parameters must be calculated for each selected control strategy. Decision 3 describes that optimal command signals must be calculated for each selected control strategy.

CyberOpt implements this three-step optimization process. Therefore, a formal framework for command signal configuration is introduced. The framework consists of a formal problem description, input and output concepts, and six tasks: Task 1: find valid command signal configuration, Task 2: record new data from the production system, Task 3: learn cost models, Task 4: learn sequences of control methods, Task 5: find an optimal command signal configuration, and Task 6: improve cost models. Each task is described by input and output concepts. The formal framework is implemented by the CyberOpt algorithm. The CyberOpt algorithm consists of five subalgorithms: (1) CyberOpt-SIC subalgorithm, (2) CyberOpt-LSC subalgorithm, (3) CyberOpt-LCM subalgorithm, (4) CyberOpt-SPC subalgorithm, and (5) CyberOpt-ICM subalgorithm. The CyberOpt-SIC subalgorithm realizes Task 1. The subalgorithm samples initial and valid command signal configurations. The initial and valid command signal configuration is required to set up the automation software of a production system. Once the automation software is set up, an evaluation of the production system can be performed to obtain an event log and cost model training data. The evaluation of the production system is described in Task 2. The CyberOpt-LSC subalgorithm realizes Task 4. The subalgorithm learns sequences of control methods from an event log. The event log is created by an evaluation of the production system described in Task 2. The CyberOpt-LCM subalgorithm realizes Task 3. The subalgorithm learns cost models from cost model training data. Cost model training data is created by an evaluation of the production system described in Task 2. The CyberOpt-SPC subalgorithm realizes Task 5. The subalgorithm solves the command signal configuration problem. The CyberOpt-ICM subalgorithm realizes Task 6. The subalgorithm improves the quality of the cost models. A new valid command signal configuration is calculated, which should be evaluated to obtain more observations of the production system.

Detailed and systematic analysis of state of the art has shown that the following machine learning techniques have not yet been used in this combination to solve the command signal configuration problem:

(i) In contrast to using manually predefined optimization models or simulation models, CyberOpt uses a machine learning technique called regression and polynomial expansion to learn cost models from observations of the production system, reducing manual engineering effort.

(ii) A machine learning technique named process mining is used to learn sequences of control methods from observations of the production system to reduce manual engineering effort. Manual creation of a behavior model is unnecessary, so manual en-

gineering effort is reduced, and the learned behavior model is synchronized with the current state of the production system.

(iii) CyberOpt uses the expected improvement criterion to calculate new valid command signal configurations that should be evaluated to obtain more observations of a production system. With more observations, cost models are more accurate. Black-box optimization approaches use this technique to gather additional data. They iterate between fitting cost models and gathering additional observations.

(iv) CyberOpt uses a modified mixed-integer nonlinear programming problem, standard solving techniques, and a standard solver to find optimal command signal configurations. The mixed-integer nonlinear programming problem is modified to use learned cost models instead of predefined mathematical models.

Three general applied scenarios are defined: a transport scenario, a storage scenario, and a pick-and-place scenario. Based on the application scenarios, it is shown that CyberOpt can be used for these specific production scenarios. Furthermore, synthetic extensions are introduced: four reference cost functions and a behavior model generator. The idea is to extend common real-world applications with synthetic data to eliminate the “toy model” problem and the problem of assuming that synthetic data does not correspond to real-world applications. The main hypothesis of this work describes that an optimal command signal configuration solution does not require manual engineering steps because the concept of cyber-physical production systems is that they adapt to new production goals without much manual engineering effort. It could be proven that this hypothesis is valid. Therefore, the following hypotheses and sub-hypotheses are proven:

(i) Given valid command signal configurations and observations from the runtime of a production system, a machine learning algorithm exists that learns cost models that can then be used by a command signal configuration algorithm to find optimal command signal configurations.

  (i) Regression analysis can be used to learn cost models.

  (ii) Linear cost models are not sufficient to predict energy consumption values. Nonlinear cost models should be used.

  (iii) Polynomial expansion can be used to model nonlinear cost models.

  (iv) The expected improvement criterion can be used to enhance the quality of cost models.

(ii) Given valid command signal configurations and observations from the runtime of the production system, a machine learning algorithm exists that learns sequences of control methods that can then be used by a command signal configuration algorithm to find optimal command signal configurations.

(i) Event logs of activities can be used to learn a behavior model.

(ii) The fuzzy mining algorithm can be used to learn a behavior model.

(iii) Given a command signal configuration problem, an algorithm exists that finds optimal command signal configurations.

(i) Human-in-the-loop and sampling are not suitable for finding an optimal command signal configuration.

(ii) A general mixed-integer nonlinear programming solver can be used to find optimal command signal configurations.

(iii) The command signal configuration problem falls into the NP-hard complexity class.

A systematic analysis has shown that the CyberOpt approach finds optimal command signals with fewer costly evaluations of the production systems compared to black-box optimization.

In future work, the CyberOpt framework can be used to develop new algorithms. The formal framework defines the six tasks that are independent of any specific machine learning technique. Task 6 that describes the improvement of cost models to reduce the number of evaluations of command signal configurations is realized by the CyberOpt-ICM subalgorithm. The CyberOpt-ICM subalgorithm uses Gaussian processes and the expected improvement (EI) criterion. Gaussian processes can be replaced by decision trees, or gradient boosted trees. The expected improvement criterion can be replaced by a lower confidence limit or a probability of improvement criterion. Task 5, which solves the command signal configuration problem, is realized by the CyberOpt-SPC subalgorithm. The problem is defined as a mixed-integer nonlinear programming problem with a nonlinear objective function, linear constraints, and a mixture of continuous and binary variables. The signal configuration problem is NP-hard. This work uses general mixed-integer nonlinear programming solvers to solve a signal configuration problem. Research was published to implement efficient algorithms to solve these optimization problems. A possible research question could be whether quantum computers can solve this problem better. The hyperparameter optimization of neural networks can also benefit from CyberOpt. CyberOpt implements black-box optimization with constraints. Usually, hyperparameter optimization of neural networks uses only black-box optimization without constraints. An open question is whether constraints can optimize the process of hyperparameter optimization and how they can be formulated. The CyberOpt approach can also be used for behavior optimization in other domains, e.g., real-time localization, mobile robot scenarios, process automation, website optimization, and transportation planning. In general, there are no limitations to using CyberOpt for other problems. CyberOpt can be used for any problem with sequences of activities and observable feedback.

Bibliography

[Ach09] T. Achterberg, “Scip: Solving constraint integer programs,” *Mathematical Programming Computation*, vol. 1, pp. 1–41, Jan. 2009.

[AD94] R. Alur and D. L. Dill, “A theory of timed automata,” *Theoretical computer science*, vol. 126, pp. 183–235, Apr. 1994.

[AG06] F. Antonio and C. Gentile, “Perspective cuts for a class of convex 0-1 mixed integer programs,” *Mathematical Programming*, vol. 106, pp. 225–236, Jun. 2006.

[Alu+93] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*. Berlin, Germany: Springer, 1993.

[AMR01] I. Akrotirianakis, I. Maros, and B. Rustem, “An outer approximation based branch and cut algorithm for convex 0-1 minlp problems,” *Optimization Methods and Software*, vol. 16, pp. 21–47, Jan. 2001.

[Ana+14] H. Anacker, R. Dumitrescu, J. Gausemeier, P. Iwanek, and T. Schierbaum, “Methodology for the identification of potentials for the integration of self-optimization in mechatronic systems,” *Procedia Technology*, vol. 15, pp. 17–26, Jan. 2014.

[Beh+01] G. Behrmann *et al.*, *Minimum-cost reachability for priced time automata*. Cham, Switzerland: Springer, 2001.

[Bel+09] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter, “Branching and bounds tightening techniques for non-convex minlp,” *Optimization Methods and Software*, vol. 24, pp. 597–634, Oct. 2009.

[Bel+10] P. Belotti, S. Cafieri, J. Lee, and L. Liberti, “Feasibility-based bounds tightening via fixed points,” in *Proc. 4th International Conference on Combinatorial Optimization and Applications (COCOA)*, Hawaii, USA, Dec. 2010, pp. 65–76.

[Bel+13] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan, “Mixed-integer nonlinear optimization,” *Acta Numerica*, vol. 22, pp. 1–131, May 2013.

[Ber+20] D. E. Bernal, S. Vigerske, F. Trespalacios, and I. E. Grossmann, “Improving the performance of dicopt in convex minlp problems using a feasibility pump,” *Optimization Methods and Software*, vol. 35, pp. 171–190, Feb. 2020.

[BG14] T. Berthold and A. Gleixner, “Undercover: A primal minlp heuristic exploring a largest sub-mip,” *Mathematical Programming*, vol. 144, pp. 315–346, Apr. 2014.

[BL12a] S. Burer and A. Letchford, “Non-convex mixed-integer nonlinear programming: A survey,” *Surveys in Operations Research and Management Science*, vol. 17, pp. 97–106, Jul. 2012.

[BL12b] S. Burer and A. N. Letchford, “Non-convex mixed-integer nonlinear programming: A survey,” *Surveys in Operations Research and Management Science*, vol. 17, pp. 97–106, Jul. 2012.

[Bon+08] P. Bonami *et al.*, “An algorithmic framework for convex mixed integer nonlinear programs,” *Discrete Optimization*, vol. 5, pp. 186–204, May 2008.

[Bos+17] F. Boschi *et al.*, “From key business factors to kpis within a reconfigurable and flexible cyber-physical system,” in *Proc. International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, New Jersey, USA, Jun. 2017, pp. 732–740.

[BR13] M. Blum and M. A. Riedmiller, “Optimization of gaussian process hyperparameters using rprop,” in *Proc. European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium, Apr. 2013, pp. 339–344.

[BV04] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, USA: Cambridge University Press, 2004.

[Cal+17] A. Calà *et al.*, “Migration from traditional towards cyber-physical production systems,” in *Proc. 15th International Conference on Industrial Informatics (INDIN)*, Emden, Germany, Jul. 2017, pp. 1147–152.

[Che+06] V. C. Chen, K.-L. Tsui, R. R. Barton, and M. Meckesheimer, “A review on design, modeling and applications of computer experiments,” *IIE Transactions*, vol. 38, no. 4, pp. 273–291, Feb. 2006.

[CN14] A. Costa and G. Nannicini, “Rbfopt: An open-source library for black-box optimization with costly function evaluations,” *Optimization Online*, vol. 4538, Sep. 2014.

[CS99] S. Ceria and J. Soares, “Convex programming for disjunctive convex optimization,” *Mathematical Programming*, vol. 86, pp. 595–614, Dec. 1999.

[Dak65] R. J. Dakin, “A tree-search algorithm for mixed integer programming problems,” *The Computer Journal*, vol. 8, pp. 250–255, Jan. 1965.

[Dan98] G. B. Dantzig, *Linear Programming and Extensions*. New Jersey, USA: Princeton University Press, 1998.

[DeT+13] G. DeTommasi, R. Vitelli, L. Boncagni, and A. C. Neto, “Modeling of marte-based real-time applications with sysml,” *IEEE Transactions on Industrial Informatics*, vol. 9, pp. 2407–2415, 2013.

[DG86] M. A. Duran and I. E. Grossmann, “An outer-approximation algorithm for a class of mixed-integer nonlinear programs,” *Mathematical Programming*, vol. 36, pp. 307–339, Oct. 1986.

[FL94] R. Fletcher and S. Leyffer, “Solving mixed integer nonlinear programs by outer approximation,” *Mathematical Programming*, vol. 66, pp. 327–349, Aug. 1994.

[Ger74] J. Gergonne, “The application of the method of least squares to the interpolation of sequences,” *Historia Mathematica*, vol. 1, no. 4, pp. 439–447, 1974.

[GR85] O. K. Gupta and A. Ravindran, “Branch and bound experiments in convex nonlinear integer programming,” *Management Science*, vol. 31, pp. 1533–1546, Dec. 1985.

[Gün+14] J. Günther, P. M. Pilarski, G. Helfrich, H. Shen, and K. Diepold, “First steps towards an intelligent laser welding architecture using deep neural networks and reinforcement learning,” *Procedia Technology*, vol. 15, pp. 474–483, 2014.

[GV07] C. W. Günther and W. M. P. Van Der Aalst, “Fuzzy mining: Adaptive process simplification based on multi-perspective metrics,” in *Proc. 5th International Conference on Business Process Management*, Brisbane, Australia, Sep. 2007, pp. 328–343.

[HB13] C. Heinzemann and S. Becker, “Executing reconfigurations in hierarchical component architectures,” in *Proc. 16th International ACM Sigsoft symposium on Component-based software engineering (CBSE)*, New York, USA, Jun. 2013, pp. 3–12.

[HBO14] H. Hijazi, P. Bonami, and A. Ouorou, “An outer-inner approximation for separable mixed-integer nonlinear programs,” *INFORMS Journal on Computing*, vol. 26, pp. 31–44, Feb. 2014.

[Hei+13] C. Heinzemann, O. Sudmann, W. Schäfer, and M. Tichy, “A discipline-spanning development process for self-adaptive mechatronic systems,” in *Proc. International Conference on Software and System Process (ICSSP)*, New York, USA, May 2013, pp. 36–45.

[HHL11] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Sequential model-based optimization for general algorithm configuration,” in *Proc. 5th International Conference on Learning and Intelligent Optimization*, Rome, Italy, Sep. 2011, pp. 507–523.

[HHL13] F. Hutter, H. Hoos, and K. Leyton-Brown, “An evaluation of sequential model-based optimization for expensive blackbox functions,” in *Proc. 15th Annual Conference Companion on Genetic and Evolutionary Computation*, New York, USA, Jul. 2013, pp. 1209–1216.

[Höl+12] C. Hölscher, D. Zimmer, J. H. Kessler, M. Krüger, and A. Trächtler, “Hierarchical optimization of coupled self-optimizing systems,” in *Proc. 10th IEEE International Conference on Industrial Informatics (INDIN)*, Beijing, China, Jul. 2012, pp. 1080–1085.

[JSW98] D. R. Jones, M. Schonlau, and W. J. Welch, “Efficient global optimization of expensive black-box functions,” *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[Jün+09] M. Jünger *et al.*, *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Heidelberg, Germany: Springer, 2009.

[Kan+15] F. Kang, S. Han, R. Salgado, and J. Li, “System probabilistic stability analysis of soil slopes using gaussian process regression with latin hypercube sampling,” *Computers and Geotechnics*, vol. 63, pp. 13–25, Oct. 2015.

[Kar+18] S. Karnouskos, R. Sinha, P. Leitão, L. Ribeiro, and T. I. Strasser, “Assessing the integration of software agents and industrial automation systems with iso/iec 25010,” in *Proc. 16th International Conference on Industrial Informatics (INDIN)*, Porto, Portugal, Jul. 2018, pp. 61–66.

[Kar+19] S. Karnouskos, L. Ribeiro, P. Leitão, A. Lüder, and B. Vogel-Heuser, “Key directions for industrial agent based cyber-physical production systems,” in *Proc. International Conference on Industrial Cyber Physical Systems (ICPS)*, Taiwan, China, May 2019, pp. 17–22.

[Ked+15] N. Keddis, B. Javed, G. Igna, and A. Zoitl, “Optimizing schedules for adaptable manufacturing systems,” in *Proc. 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, Luxembourg City, Luxembourg, Sep. 2015, pp. 1–8.

[Kha80] L. G. Khachiyan, “Polynomial algorithms in linear programming,” *USSR Computational Mathematics and Mathematical Physics*, vol. 20, pp. 53–72, 1980.

[KPP04] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Heidelberg, Germany: Springer, 2004.

[Kro+18] J. Kronqvist, D. Bernal, A. Lundell, and I. Grossmann, “A review and comparison of solvers for convex minlp,” *Optimization and Engineering*, vol. 20, pp. 397–455, Dec. 2018.

[KST14] M. Kowal, I. Schaefer, and M. Tribastone, “Family-based performance analysis of variant-rich software systems,” in *Proc. 17th International Conference on Fundamental Approaches to Software Engineering (FASE)*, New York, USA, Apr. 2014, pp. 94–108.

[LD60] A. H. Land and A. G. Doig, “An automatic method of solving discrete programming problems,” *Econometrica*, vol. 28, pp. 497–520, Jul. 1960.

[Lee08] E. A. Lee, “Cyber physical systems: Design challenges,” in *Proc. 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, Washington, USA, May 2008, pp. 363–369.

[Lei+18] P. Leitão, S. Karnouskos, L. Ribeiro, P. Moutis, J. Barbosa, and T. I. Strasser, “Integration patterns for interfacing software agents with industrial automation systems,” in *Proc. 44th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, Washington D.C., USA, Oct. 2018, pp. 2908–2913.

[Lie+14] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, “Assessing the state-of-practice of model-based engineering in the embedded systems domain,” in *Model-Driven Engineering Languages and Systems*, Cham, Switzerland: Springer, 2014, pp. 166–182.

[LL12] J. Lee and S. Leyffer, *Mixed Integer Nonlinear Programming*. New York, USA: Springer, 2012.

[LM06] L. Liberti and N. Maculan, *Global Optimization: From Theory to Implementation*. Heidelberg, Germany: Springer Science & Business Media, 2006.

[Lub+16] M. Lubin, E. Yamangil, R. Bent, and J. Vielma, "Extended formulations in mixed-integer convex programming," in *Proc. International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, Liège, Belgium, Jun. 2016.

[LY15] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*. Cham, Switzerland: Springer, 2015.

[Mai+11] A. Maier, A. Vodencarevic, O. Niggemann, R. Just, and M. Jaeger, "Anomaly detection in production plants using timed automata," in *Proc. 8th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, Noordwijkerhout, Netherlands, Jul. 2011, pp. 363–369.

[Mai14] A. Maier, "Online passive learning of timed automata for cyber-physical production systems," in *Proc. 2014 12th International Conference on Industrial Informatics (INDIN)*, Porta Alegre, Brazil, Jul. 2014, pp. 60–66.

[Mar99] R. K. Martin, *Large Scale Linear and Integer Optimization: A Unified Approach*. New York, USA: Springer, 1999.

[Men+03] H. Mendelbaum, R. Yehezkael, T. Hirst, A. Teitelbaum, and S. Bloch, "Defining parallel automata and their conflicts," in *Proc. 7th Circuits, Systems, Communications & Computers (CSCC)*, Corfu, Greece, Jul. 2003, pp. 126–133.

[MF13] R. Misener and C. Floudas, "Glomiqo: Global mixed-integer quadratic optimizer," *Journal of Global Optimization*, vol. 57, pp. 3–50, Sep. 2013.

[MF14] R. Misener and C. A. Floudas, "Antigone: Algorithms for continuous integer global optimization of nonlinear equations," *Journal of Global Optimization*, vol. 59, pp. 503–526, Mar. 2014.

[MK87] K. G. Murty and S. N. Kabadi, "Some np-complete problems in quadratic and nonlinear programming," *Mathematical programming*, vol. 39, pp. 117–129, Jun. 1987.

[MPT00] S. Martello, D. Pisinger, and P. Toth, "New trends in exact algorithms for the 0-1 knapsack problem," *European Journal of Operational Research*, vol. 123, pp. 325–332, Jun. 2000.

[MSF15] R. Misener, J. Smadbeck, and C. Floudas, "Dynamically generated cutting planes for mixed-integer quadratically constrained quadratic programs and their incorporation into glomiqo," *Optimization Methods and Software*, vol. 30, pp. 215–249, Jan. 2015.

[Mur89] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, pp. 541–580, Apr. 1989.

[NH18] A. Nystrom and J. Hughes, “Leveraging sparsity to speed up polynomial feature expansions of csr matrices using K-simplex numbers,” *arXiv preprint:1803*, 2018.

[Nig+12] O. Niggemann, B. Stein, A. Vodencarevic, A. Maier, and H. K. Büning, “Learning behavior models for hybrid timed systems,” in *Proc. Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, Toronto, Canada, Jul. 2012, pp. 1083–1090.

[OBV15] M. Obermeier, S. Braun, and B. Vogel-Heuser, “A model-driven approach on object-oriented plc programming for manufacturing systems with regard to usability,” *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 790–800, 2015.

[Oga01] K. Ogata, *Modern Control Engineering*. Upper Saddle River, USA: Prentice Hall PTR, 2001.

[ON15] J. Otto and O. Niggemann, “Automatic parameterization of automation software for plug-and-produce,” in *AAAI-15 Workshop on Algorithm Configuration (AlgoConf)*, Austin, USA, Jan. 2015.

[OVN16] J. Otto, B. Vogel-Heuser, and O. Niggemann, “Optimizing modular and reconfigurable cyber-physical production systems by determining parameters automatically,” in *Proc. IEEE 14th International Conference on Industrial Informatics (INDIN)*, Futuroscope-Poitiers, France, Jul. 2016, pp. 1100–1105.

[OVN18a] J. Otto, B. Vogel-Heuser, and O. Niggemann, “Online parameter estimation for cyber-physical production systems based on mixed integer nonlinear programming, process mining and black-box optimization techniques,” *at-Automatisierungstechnik*, vol. 66, no. 4, pp. 331–343, 2018.

[OVN18b] J. Otto, B. Vogel-Heuser, and O. Niggemann, “Automatic parameter estimation for reusable software components of modular and reconfigurable cyber-physical production systems in the domain of discrete manufacturing,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 99, pp. 275–282, 2018.

[PW00] R. Pörn and T. Westerlund, “A cutting plane method for minimizing pseudo-convex functions in the mixed integer case,” *Computers & Chemical Engineering*, vol. 24, pp. 2655–2665, Dec. 2000.

[Ras06] C. E. Rasmussen, *Gaussian processes for machine learning*. London, UK: MIT Press, 2006.

[Ric+10] F. Ricca, M. D. Penta, M. Torchiano, P. Tonella, and M. Ceccato, “How developers’ experience and ability influence web application comprehension tasks supported by uml stereotypes: A series of four experiments,” *IEEE Transactions on Software Engineering*, vol. 36, pp. 96–118, 2010.

[RLL10] M. Roth, J.-J. Lesage, and L. Litz, “Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems,” in *Proc. American Control Conference (ACC)*, Baltimore, USA, Jun. 2010, pp. 2601–2606.

[Rod+13] J. Rodriguez *et al.*, “State of the art of finite control set model predictive control in power electronics,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 2, pp. 1003–1016, 2013.

[Rot11] F. Rothlauf, *Design of Modern Heuristics: Principles and Application*. Heidelberg, Germany: Springer, 2011.

[RS95] H. Ryoo and N. Sahinidis, “Global optimization of nonconvex nlps and minlps with applications in process design,” *Computers & Chemical Engineering*, vol. 19, pp. 551–566, May 1995.

[RS96] H.-S. Ryoo and N. V. Sahinidis, “A branch-and-reduce approach to global optimization,” *Journal of Global Optimization*, vol. 8, pp. 107–138, Sep. 1996.

[RSV16] S. Rehberger, L. Spreiter, and B. Vogel-Heuser, “An agent approach to flexible automated production systems based on discrete and continuous reasoning,” in *Proc. 12th IEEE International Conference on Automation Science and Engineering (CASE)*, Fort Worth, USA, Aug. 2016, pp. 1249–1256.

[RW06] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. London, UK: MIT Press, 2006.

[Sah96] N. V. Sahinidis, “Baron: A general purpose global optimization software package,” *Journal of Global Optimization*, vol. 8, pp. 201–205, Mar. 1996.

[San+] T. J. Santner, B. J. Williams, W. Notz, and B. J. Williams, *The design and analysis of computer experiments*. New York, USA: Springer, vol. 1.

[Sch04] H. Schichl, “Models and the history of modeling,” in *Modeling Languages in Mathematical Optimization*, New York, USA: Springer, 2004, pp. 25–36.

[SM99] R. A. Stubbs and S. Mehrotra, “A branch-and-cut method for 0-1 mixed convex programming,” *Mathematical Programming*, vol. 86, pp. 515–532, Dec. 1999.

[SMG13] M. A. Stephens, C. Manzie, and M. C. Good, “Model predictive control for reference tracking on an industrial machine tool servo drive,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 2, pp. 808–816, 2013.

[SW01] C. Still and T. Westerlund, “Extended cutting plane algorithm,” in *Encyclopedia of Optimization*. Boston, MA: Springer, 2001, pp. 593–601.

[Tik+14] D. Tikhonov, D. Schütz, S. Ulewicz, and B. Vogel-Heuser, “Towards industrial application of model-driven platform-independent plc programming using uml,” in *Proc. 40th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, Dallas, USA, Oct. 2014, pp. 2638–2644.

[TS05] M. Tawarmalani and N. Sahinidis, “A polyhedral branch-and-cut approach to global optimizatio,” *Mathematical Programming*, vol. 103, pp. 225–249, May 2005.

[vdAal+03] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters, “Workflow mining: A survey of issues and approaches,” *Data and Knowledge Engineering*, vol. 47, no. 2, pp. 237–267, 2003.

[vdAWM04] 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, pp. 1128–1142, 2004.

[VG17] S. Vigerske and A. Gleixner, “Scip: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework,” *Optimization Methods and Software*, vol. 33, pp. 1–31, Jul. 2017.

[VH16] B. Vogel-Heuser and D. Hess, “Guest editorial industry 4.0–prerequisites and visions,” *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 411–413, 2016.

[Vog+14] B. Vogel-Heuser, B. Weisenberger, H.-G. Meyer, and J. Plossnig, “Modeling of power consumption in manufacturing: Gross and detailed planning in consideration of all forms of energy as planning resources including load management during runtime,” in *Proc. International Conference on Industrial Technology (ICIT)*, Busan, Korea, Feb. 2014, pp. 659–663.

[Vog+15] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, “Evolution of software in automated production systems: Challenges and research directions,” *Journal of Systems and Software*, vol. 110, pp. 54–84, 2015.

[WB10] Y. Wang and S. Boyd, “Fast model predictive control using online optimization,” *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010.

[Wes+98] T. Westerlund, H. Skrifvars, I. Harjunkoski, and R. Pörn, “An extended cutting plane method for a class of non-convex minlp problems,” *Computers & Chemical Engineering*, vol. 22, pp. 357–365, Jun. 1998.

[WNS15] S. Windmann, O. Niggemann, and H. Stichweh, “Energy efficiency optimization by automatic coordination of motor speeds in conveying systems,” in *Proc. IEEE International Conference on Industrial Technology (ICIT)*, Seville, Spain, Mar. 2015, pp. 731–737.

[WP02] T. Westerlund and R. Pörn, “Solving pseudo-convex mixed integer optimization problems by cutting plane techniques,” *Optimization and Engineering*, vol. 3, pp. 253–280, Sep. 2002.

[WP95] T. Westerlund and F. Pettersson, “An extended cutting plane method for solving convex minlp problems,” *Computers & Chemical Engineering*, vol. 19, pp. 131–136, Jun. 1995.

[WvDD06] A. Weijters, W. M. van Der Aalst, and A. A. De Medeiros, “Process mining with the heuristics miner-algorithm,” *Working Paper Series*, vol. 166, pp. 1–34, 2006.

[YHL12] G. Yuan, C. Ho, and C. Lin, “Recent advances of large-scale linear classification,” *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2584–2603, 2012.

[Zho+17] K. Zhou, M. Kılınç, X. Chen, and N. Sahinidis, “An efficient strategy for the activation of mip relaxations in a multicore global minlp solver,” *Journal of Global Optimization*, vol. 70, pp. 497–516, Aug. 2017.

[Zou+18] M. Zou, F. Ocker, E. Huang, B. Vogel-Heuser, and C.-H. Chen, “Design parameter optimization of automated production systems,” in *Proc. 14th International Conference on Automation Science and Engineering (CASE)*, Munich, Germany, Aug. 2018, pp. 359–364.

[ZXG14] R. Zhang, A. Xue, and F. Gao, “Temperature control of industrial coke furnace using novel state space model predictive control,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2084–2092, 2014.

List of Tables

List of Figures