Applying Machine Learning in a Robot/Operator Collaborative Assembly Setup

Master Thesis

June 2019 - February 2020

Master:	Manufacturing Systems Engineering
Department:	Mechanical Engineering
Research Group:	Dynamics and Control
Student:	Ronald Cockx
Identity Number:	0971042
Thesis Supervisors:	Professor Dr. Ir. I.J.B.F. Adan
	Professor Dr. Ir. A.A.J. Lefeber
Date:	February 6, 2020
Report Number:	DC 2020.019

Acknowledgment

This thesis provides the methodology and results of the research on the application of machine learning on TNO's FieldLab Flexible Manufacturing collaborative assembly setup. This research has been my final project in the master Manufacturing Systems Engineering at the Eindhoven University of Technology and has been a big learning experience and of significant value for my master. First and foremost, I would like to thank Assistant Professor Dr. Ir. A.A.J. Lefeber for his guidance and valuable feedback during the research project. Moreover, I would like to thank Professor Dr. Ir. I.J.B.F. Adan for his counsel and guidance during the research and the valuable information and discussions during the manufacturing related seminars.

In addition, I would like to thank Bram Kale for his guidance during the start of this research together with Ir. J. van Oosterhout and Dr. T. Bosch who provided valuable information with regards to the assembly setup and the vision of the Fieldlab Flexible Manufacturing.

Finally, I would like to thank Maarten Monhemius, Dirk van de Sande and Jeroen Didden for their insights, opinions and tips that helped me obtain insights and results described in this thesis.

Abstract

The companies Bosch Rexroth, Tegema and Omron have collaborated with TNO, Eindhoven University of Technology and Avans University of Applied Science in order to design and build a state of the art, flexible assembly line in which an operator and a collaborative robot (cobot) together assemble low volume high variable products. This research aims to minimize the cycle time of each individual assembly by implementing flexible and adaptive behavior dependent on the human operator on the assembly work cell by optimizing the order in which the assembly tasks are performed, using operator specific process time data.

The cobot is idle for a large portion of the assembly of the current example product assembled on the demonstrator setup. A product assembly for which the utilization of the operator and the cobot are close to equal is desired to gather information of the attainable performance gains by alternating the assembly order dependent on the operator's process times. Therefore, a theoretical example is developed together with two operator characteristics, for which the optimized assembly order for Operator 1 results in a cycle time of 280 seconds per assembly and the optimized assembly order for Operator 2 results in a cycle time equal to 260 seconds. Applying the wrong assembly order for Operator 1, results in a cycle time of 305 seconds per assembly, which is equal to an increase of almost 9%. Applying the wrong assembly order for Operator 2 results in a cycle time of 270 seconds per assembly, which is equal to an increase of almost 4%.

The goal of this research is to formulate a suitable optimization technique for continuously minimizing the cycle time of each product assembly in order to minimize the total makespan of the assembly procedure. Minimization of the makespan is achieved by optimizing the assembly order of the individual assemblies of products depending on the operator's process times per assembly task, subjected to the product's assembly precedence constraints and resource assignment constraints. Mixed integer linear programming (MILP) is applied to find the optimal assembly order corresponding to process times.

A procedure and its limitations to optimize the assembly order on the expected process times by solving the MILP online is provided. The expected process times are computed on a weighted moving average, based on a first order Kalman filter. For small products, the makespan of the assembly procedure can be minimized in real time by applying the online optimization procedure, however, when complexity increases, computation times limit the applicability. Therefore, three computationally cheap supervised classification algorithms are posed to recognize the currently working operator and apply the preferred assembly order (Euclidean distance, modal closest, and K-nearest neighbors). Furthermore, K-means clustering is posed as extension to the supervised classification algorithms to increase the flexibility of the supervised classification algorithms by adding the ability to recognize a new operator and learn the new operator's characteristics.

Discrete event simulations are applied to gain insight into the possible performance gains by applying the algorithms on the original setup. The results of the simulations show that the makespan of the assembly can be minimized by applying a suitable assembly order. The K-nearest neighbors algorithm is the most suitable algorithm, outperforming the other procedures slightly. The K-means clustering algorithm can be applied to recognize new operators and learn their characteristics. All of the proposed algorithms are relatively simple to implement, allowing the makespan minimization procedure to be implemented with low effort.

Contents

List of Figures

Chapter 1

Introduction

A new 200 hectares high tech campus has recently been built in the center of one of the largest city parks in Eindhoven, near Eindhoven Airport. The campus is known as the Brainport Industries Campus (BIC) and it is a state-of-the-art working and learning environment for the next generation in high-tech manufacturing. It is home to broad alliances between suppliers, specialized companies and innovative educational and knowledge institutions. The aim of the campus is to provide the first ever location where high-tech suppliers innovate and produce collaboratively and share resources, such as cleanrooms, flexible manufacturing areas, warehouses and advanced facilities [1].

The companies Bosch Rexroth, Tegema and Omron have collaborated with TNO, Eindhoven University of Technology and Avans University of Applied Science in order to design and build a state of the art, flexible assembly line. A demonstrator of this assembly line can be found at the BIC, located at the FieldLab Flexible Manufacturing area. The demonstrator setup consists of a collaborative robot (cobot) and a human operator who collaboratively assemble products. Some tasks are processed by the human operator, others are done by the cobot. A longer term objective of the FieldLab is to have tasks processed by either the operator or the cobot, or both simultaneously. The cobot is designed to collaborate with a human operator, hence, for safety reasons, the cobot uses lower speeds and smaller forces than a regular industrial robot.

The BIC assembly demonstrator shown in Figure 1.1 is designed for flexible manufacturing. It is used to test and study a variety of flexibility improvements. Flexibility can for example be found in exchangeable part bins, causing setup time for a new product to be low. The demonstrator uses light projection which shows instructions of the assembly steps on the table, resulting in high flexibility because one operator can assemble a large variety of products without training and new inexperienced operators can start assembling products without much training as all required steps are instructed to the operator via the projection. Additionally, the demonstrator setup is used to improve collaboration between the human operator and the cobot.

The companies in the FieldLab have opted for the assembly setup to be using smart software to optimize the assembly efficiency and increase the flexibility. Part of the desired smart software is the use of self learning abilities in the assembly setup. The term self learning immediately raises the idea or artificial intelligence (AI), which is defined as the study of systems that perceive their environment and take actions that maximize their chance of successfully achieving their goals [2]. For a system to be intelligent, it requires the ability to *learn*, where learning is defined as the process to gather new or to modify existing knowledge [2]. Research on AI has been divided into subfields, such as machine learning (ML) and artificial neural networks (ANN) [3]. The latter is used for systems to learn to perform tasks by considering examples, generally without being programmed with task-specific rules [4]. This is arguably very desirable in future upgrades of the setup, however, for now it's considered to be too complex to study in this research. ML is defined as the study of algorithms and statistical models used to perform tasks without using explicit instructions [5]. It can be split into subfields: supervised, semi-supervised, reinforced, and unsupervised learning [2]. An algorithm is labeled as supervised if a set of training data of which the input and the output is known, is used to give feedback

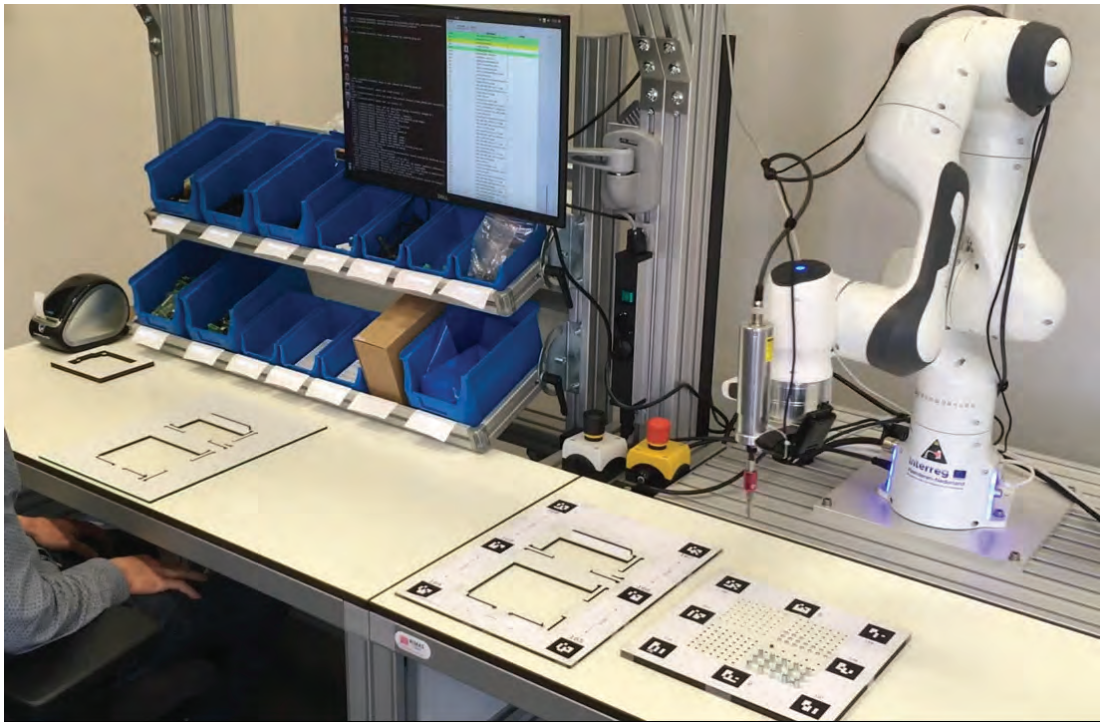


Figure 1.1: TNO demonstrator setup located at the BIC in Eindhoven.

to the system when encountering new data [2]. If the outputs of the training set are not known and the algorithm searches for structures in the data, the algorithm is known as unsupervised [2]. ML is considered to be a suitable approach for the application of0- "self learning abilities" to the assembly setup. Possible usage of ML in the assembly setup can be found in:

- Bin configuration: measure the most frequent parts to pick and adapt the location of the bins to optimize picking tasks.
- Order sequence: measure the orders of the day and sequence them as such that the setup time is reduced as much as possible.
- Use of a smart screwdriver: measure the applied torque for each screw and adapt the torque and speed for each screw position on the previous measured data.
- Assembly task order: measure the time needed for and between assembly actions and optimize the assembly task order to reduce the average cycle time.
- Process time variation: measure time variations in arrival and departure times and optimize the scheduling accordingly.
- Test sequence: measure test results and fails and adapt the test sequence to discover the most frequent errors as soon as possible.
- Human awareness: measure response time and working speed and change the lightning level to activate the human brain to help the operator focus.
- Operator confidence: detect odd behavior of the operator and instruct the operator how to return to the normal situation with the goal to reduce errors.

Unfortunately, not all of the options are attainable. For example, a test sequence is not yet applied to the assembly setup. Moreover, operator confidence and human awareness are considered too complex

for the current research because they require a high amount of input data. Order sequence and process time variation are expected to result in low performance gains, as the current demonstrator produces a low volume of products per day, with a high variety of utilization. This leaves three possible usages of ML: bin configuration, use of a smart screwdriver, and alternating the assembly order. From these three possible usages, the assembly order option is studied for this research because it requires only the measured process times and the waiting times for assembly tasks to be optimized and is expected to obtain more severe performance gains than the other two usages.

Opposed to a robot/cobot, humans tend to be unpredictable, inconsistent and highly variable, thus one of the key characteristics of an operator in an assembly procedure are the operator's process times. Moreover, these process times are easily measurable. It is expected that applying the same assembly instructions for all operators causes the risk of creating undesirable waiting times for the robot or the operator because of differences in the individual process times of the operators. These waiting times increase the cycle time of an assembly. The cycle time is defined as the time from when a job is released into a station to when it exits [6]. In order to obtain high efficiency and minimize production costs, it is desirable to minimize the sum of the cycle times of all assembled products for a certain set of assemblies. We define the sum of the cycle times of a set of assemblies as the *makespan* of the set. In this research, the makespan is minimized by continuously minimizing the cycle time of each assembly step by rearranging the order of the assembly steps for different operators. A vast amount of algorithms and procedures is applicable for such task. Commonly, schedule optimization using Mixed Integer Linear Programming (MILP) is applied to achieve the best outcome of an objective function subjected to predetermined constraints. Solving the optimization problem gives exact results, however, such optimization problems are computationally expensive. Commonly in scheduling problems, computation times are reduced by applying approximations, genetic algorithms or heuristics and research on these heuristics has been done for more than 20 years [1].

Process times of the assembly tasks in the assembly setup are assumed to be operator dependent. Essentially, the cobot is expected to process tasks with consistent process times, i.e., the process time variances of the cobot are assumed to be negligible. This operator dependency of the process times of the assembly procedure opens the door to supervised learning. Supervised learning is based on input-output samples which are applied as training samples to classify new data [7]. Since process times are expected to be operator dependent, the operators and their corresponding process times can be applied as classified training set. New measured process times can be compared with the training sets and the currently working operator can be recognized. The optimized assembly order corresponding to the recognized operator's mean process times can then be communicated to the system to apply a (near) optimal assembly order for the currently working operator. One of the advantages of applying supervised learning opposed to unsupervised learning is the ease with which it can be implemented. In general, supervised learning algorithms tend to be relatively easy to understand and to implement [2].

All measures of supervised learning generate a function that maps inputs to desired outputs [8]. The applied function defines the effectiveness of the algorithm. It is therefore important to find a function which is suited to map the input to the desired output. Popular supervised learning algorithms are linear classifiers, support vector machines, decision trees, K-nearest neighbors, and random forests [2]. K-nearest neighbors is an applicable algorithm in the case of recognizing the working operator on the assembly system as it uses a test set with labeled data and is therefore applied in this research. When a new, unlabeled data point is put into the algorithm, the new data is compared to its K nearest located neighbors and their corresponding label in order to label the new data point [2]. Moreover, two functions are proposed which are less complex than the K nearest neighbors algorithm. This is done because if the performance gain for a simple algorithm is equal to the performance gain of a complex algorithm, the simple algorithm is preferred [2]. These simpler functions require the mean process time per operator as test set. The simplest function computes the Euclidean distance between the input process times and the mean process times of the operators. The input data is labeled to the operator to which the input process times are the closest. Additionally, a function is defined which computes the differences between the input process time and the mean process times per assembly task. It then labels each assembly task to the closest operator and

computes the modal value of the labels. These three functions (Euclidean distance, modal closest, and K-nearest neighbors) are chosen because they are all easy to compute and implement. Moreover, they are all computationally cheap. More supervised learning algorithms can be applied, however, they are considered to be out of the scope of this research.

A disadvantage of supervised learning is the requirement of a test set. Hence, supervised learning is not applicable to recognize a new operator for which no test set is implemented. It is desirable to obtain the ability to recognize new operators with which the system has no experience yet. Commonly, clustering techniques are applied when data instances are to be divided into groups [2], such as hierarchical clustering, partitional clustering, density-based clustering, and grid-based clustering [9]. For this research, a start has been made of modeling new operators by applying K-means, which is an algorithm corresponding to the partitional clustering technique. K-means clustering is a commonly applied method as it iteratively reallocates data point labels causing the algorithm to improve over time opposed to hierarchical clustering models [9]. Density-based clustering models are also expected to be applicable. When a threshold value of data points within a region is reached, the region is labeled as new cluster. This can be combined with K-means clustering to reallocate data instances close to the newly labeled region and could therefore improve over time. The same holds for Grid-Based models. Unfortunately, implementing the last two clustering techniques remains open for further research.

The research topic and scope have been introduced in this chapter, next the problem definition is explained in more detail in Chapter 2. A formulation for computing the optimal assembly order using Mixed Integer Linear Programming (MILP) is given in Chapter 3. In Chapter 4 a procedure to optimize the assembly order online is proposed together with the set of supervised classification algorithms with the capability to minimize the makespan of the assembly. Additionally, the K-means algorithm to recognize a new, unknown operator while applying supervised learning is explained. These algorithms are proposed in order of increasing complexity. Chapter 5 provides a discrete event simulation model used to simulate the effects of the different makespan minimization techniques on the cobot/operator collaborative assembly setup. The results of the applied techniques on the discrete event simulation are discussed in Chapter 6, and the conclusions, limitations, and recommendations based on these results are provided in Chapter 7.

Chapter 2

Problem Definition

The introduction provided a brief description of the demonstrator setup and the research goal, together with a brief explanation of the approach of this research. In this chapter, a detailed description of the setup is given together with an overview of the possible gains and the limitations of one of the real world products assembled on the demonstrator setup. Furthermore, a theoretical example is described, which is applied to provide insight into the possible gains of applying operator dependent assembly order optimization. Finally, the research question of this graduation project is posed.

2.1 TNO BIC Demonstrator Setup

The demonstrator setup located at the BIC FieldLab Flexible Manufacturing area, provided by TNO, is shown in Figure 1.1. It is a highly flexible assembly line, in which different products are assembled with minimal set-up time. Currently, the demonstrator contains workspace for an operator, workspace for a cobot, a cobot, bins with the to-be-assembled parts, light projection with the assembly instructions shown within the operator's workspace, and a computer containing information about the assembly tasks. A tester is present in which the current flow in the printed circuit board (PCB) is tested. Moreover, a label-writer is located near the operator to print the product label. A vision system is applied, which notices when the operator has started a task, how many parts are picked from the bins, and when the operator has finished the task. The start and finish times of the tasks performed by the cobot are also measured.

The following steps are required for assembling the Omron product:

1. the PCB is prepared:
   (a) the PCB is picked from the bin and placed on the table;
   (b) prepare the PCB;
2. the PCB is moved to the tester and the current flow in the PCB is tested;
3. the heat sink is mounted:
   (a) the rear casing is picked from the bin and placed on the table;
   (b) the heat sink is placed in the rear casing;
4. the heat sink is secured:
   (a) the rear casing is moved to the cobot;
   (b) the heat sink is secured with one screw by the cobot;
5. the front casing is picked and placed on the table and the PCB is placed in the front casing;
6. the front casing with PCB is moved to the cobot and the PCB is secured tightly in the front casing by the cobot, using four screws;

7. placing the product label:
   (a) the rear casing is moved back to the operator's workspace;
   (b) the product label is placed on the rear casing;
8. the rear casing is placed on the front casing at cobot's workspace;
9. the cobot assembles the rear casing to the front casing by securing four screws in the casings;
10. a box is picked from the bin and prepared for packing;
11. the assembled product is picked from the cobot and packed in the box;

An overview of this assembly procedure is shown in Figure 2.1.

The applied assembly order as described above and shown in Figure 2.1 can be alternated as much as the precedence constraints allow. For example, the PCB can not be tested before it is prepared and the product cannot be packed before it is assembled. An overview of all precedence constraints is shown in Figure 2.2. In addition, this overview contains the information of the resource that is able to perform a task and the expected process time for the task. As can be seen from the overview, the assembly process can start with tasks 1, 3, 7 and 10. A string of tasks has to be fulfilled on the front casing of the product, starting with task 1. Three tasks are performed on the rear casing (task 3, 4 and 7). Both casings are assembled not before the tasks on the individual cases are finished. Preparing the box (task 10) can be done at any moment, as long as it is finished before the product is packed. Packing of the product has to be the final task.

Figure 2.3 shows the Gantt chart as result of applying the process times given in Figure 2.2 and the assembly schedule as described above and indicated in Figure 2.1. Moreover, the same assembly scheme with corresponding Gantt chart is found when optimizing the assembly order with the process times given in Figure 2.1, by solving the optimization problem as formulated in Chapter 3. It is notable that the cobot is idle for a large portion of the assembly. Adjusting the assembly order for varying operator dependent process times could decrease the idle time, nevertheless, the idle time of the cobot causes the attainable gains on this particular assembly procedure to be small. A product assembly for which the utilization of the operator and the utilization of the cobot are close to equal is desirable to gain insight into attainable performance gains by alternating the assembly order dependent on the operator's process times. The next section describes a theoretical example of a product that can be assembled on the cobot/operator collaborative assembly setup for which the utilization of the operator and the cobot are close to equal.

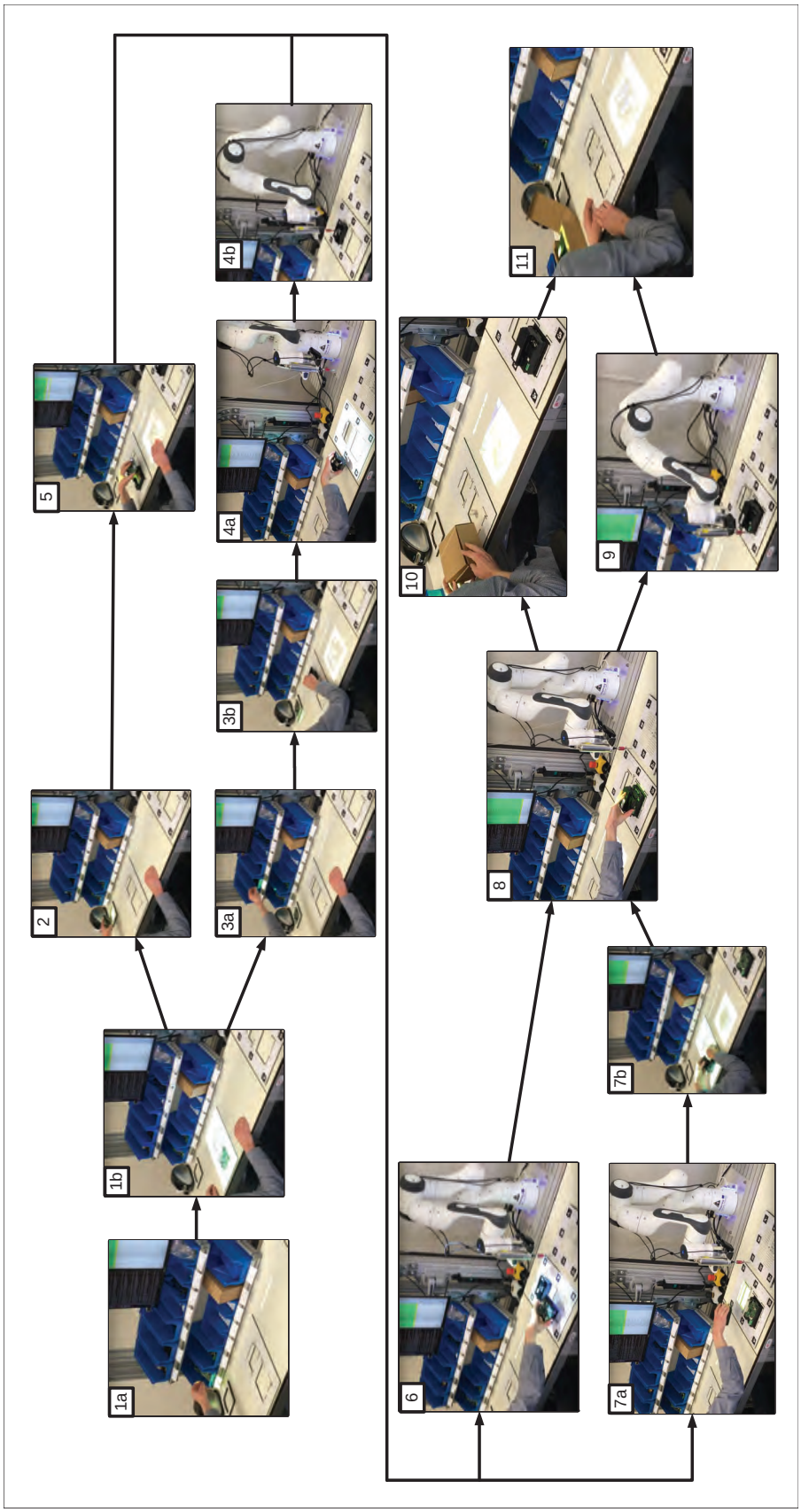


Figure 2.1: Overview of the assembly steps to assemble the Omron product.

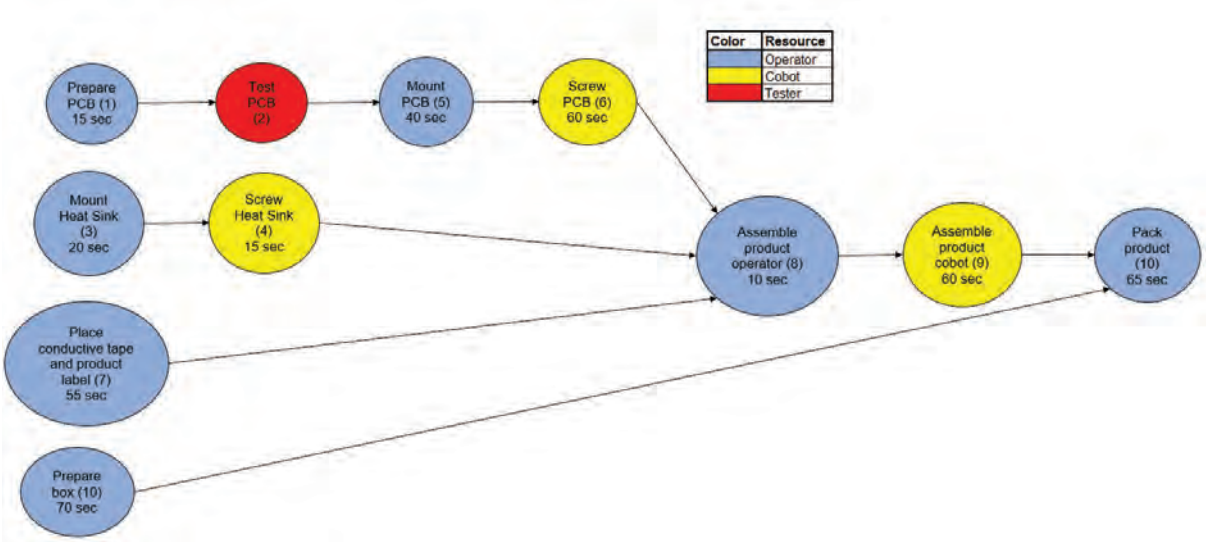


Figure 2.2: Overview of the precedence constraints, the resource assignment and the process times of the assembly procedure of the Omron product.

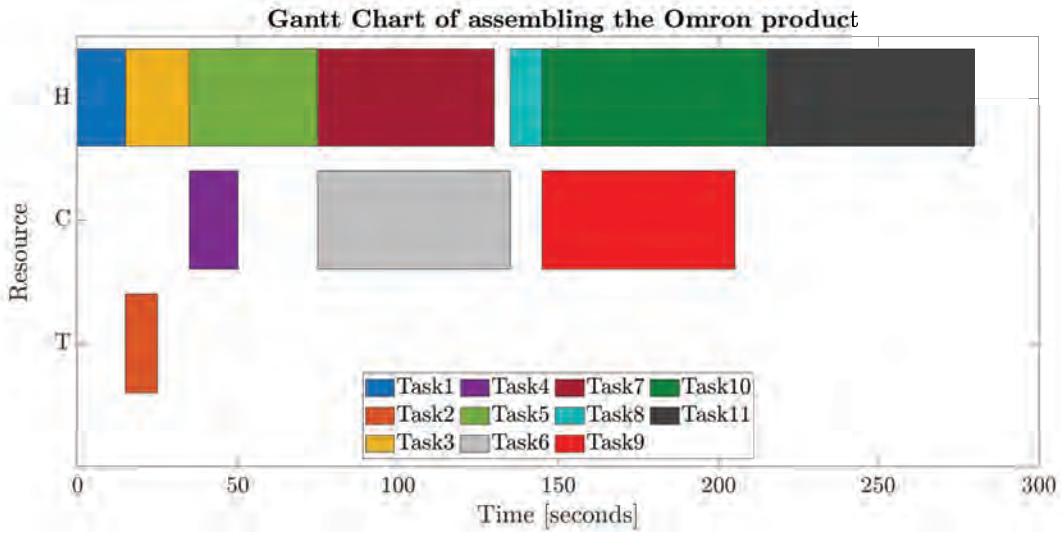
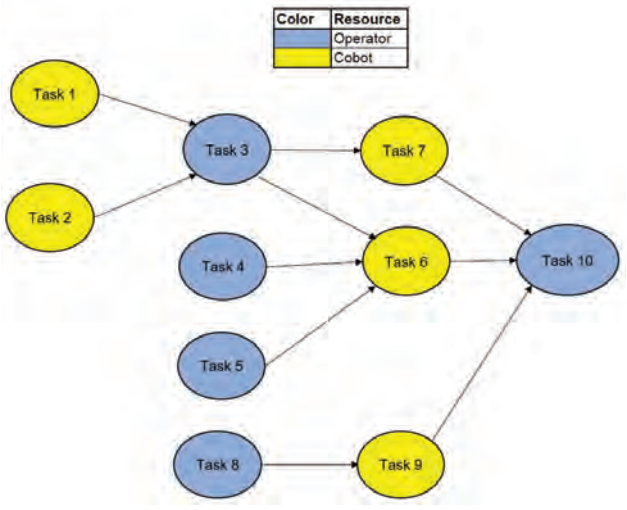


Figure 2.3: Gantt chart of the assembly procedure for the Omron product.

2.2 Theoretical Example

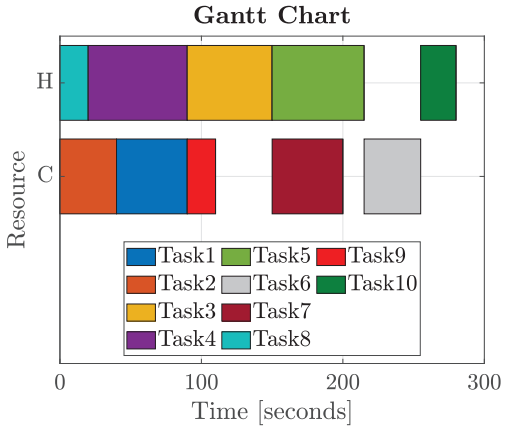
An overview of the demonstrator setup and the assembly of the Omron product has been provided in Section 2.1. Additionally, the limitations of the performance gains have been explained. A theoretical assembly example applicable to gain insight into the attainable performance gains of optimizing the assembly sequence depending on operator's process times is provided in this section. The precedence overview and resource assignment of the example assembly product is shown in Figure 2.4. For the example, five tasks are assigned to the operator and five tasks are assigned to the cobot. Five of the ten tasks can be used as starting task (task 1, 2, 4, 5, and 8) as no tasks have to be finished prior to starting one of these tasks. Each operator task is followed by a cobot task and vice versa. A task can consist of multiple actions (e.g., tightening multiple screws or picking a part, placing it on the table and mounting another part on it). Alternating the order of the individual actions performed by the same resource in series is not considered to be beneficial as it is assumed that no waiting time can be removed by alternating individual actions rather than tasks.

Two operators are defined, having different process times for the performed tasks. The process times of the two different operators are given in the table next to the precedence overview of Figure 2.4. It is assumed that the cobot's process times are consistent. The potential performance gains of alternating the assembly order depending on the operator's process times are shown by the four Gantt charts depicted in Figure 2.5. First of all, Figure 2.5a shows the optimized assembly order for Operator 1, using the process times as shown in the table of Figure 2.4. Applying this assembly order on the process times corresponding to Operator 1, results in a cycle time of 280 seconds per assembly. The same is done for the process times corresponding to Operator 2. This results in a different assembly order, as shown in Figure 2.5b. The cycle time for Operator 2 is equal to 260 seconds if the optimized assembly order for this operator is applied. Applying the assembly order of Operator 2 on the process times corresponding to Operator 1 and vice versa results in the Gantt charts shown in Figure 2.5c and Figure 2.5d respectively. Applying the wrong assembly order for Operator 1, results in a cycle time of 305 seconds per assembly as can be seen in Figure 2.5c, which is equal to an increase of almost 9%. Applying the wrong assembly order for Operator 2 results in a cycle time of 270 seconds per assembly, which is equal to an increase of almost 4%.

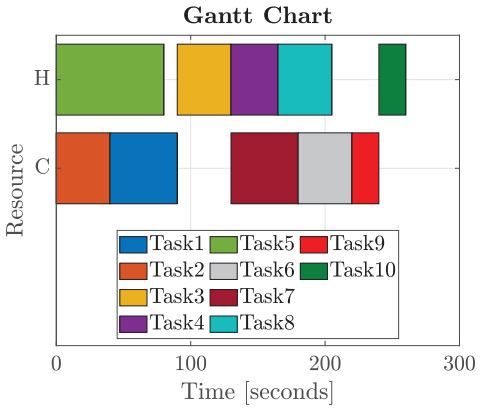


Task #	Process Times [seconds]	
	Operator 1	Operator 2
1	50	50
2	40	40
3	60	40
4	70	35
5	65	80
6	40	40
7	50	50
8	20	40
9	20	20
10	25	20

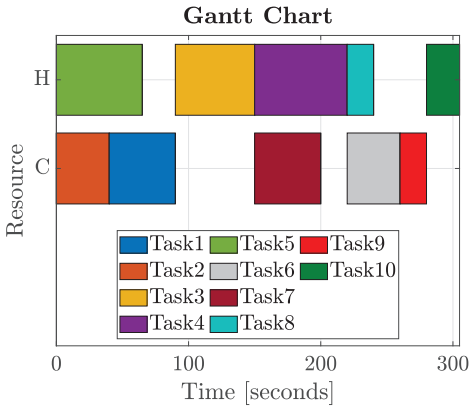
Figure 2.4: Overview of the precedence constraints and the resource assignment of the assembly procedure of the example product, with the process times of the two operators on the right of the overview.



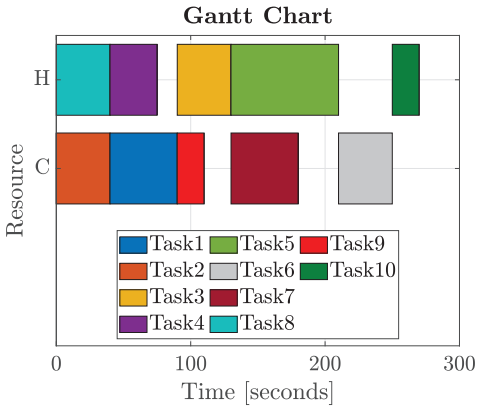
(a) Gantt chart of applying the optimized assembly order for operator 1. The assembly is finished after 280 seconds.



(b) Gantt chart of applying the optimized assembly order for operator 2. The assembly is finished after 260 seconds.



(c) Gantt chart of applying the optimized assembly order for operator 2, with the process times of operator 1. The assembly is finished after 305 seconds.



(d) Gantt chart of applying the optimized assembly order for operator 1, with the process times of operator 2. The assembly is finished after 270 seconds.

Figure 2.5: Gantt charts for two operators, using either their own optimized assembly order (2.5a and 2.5b) or each others optimized assembly order (2.5c and 2.5d).

2.3 Research Question and Approach

The goal of this research is to formulate a suitable optimization procedure to continuously minimize the cycle time of each product assembly by optimizing the assembly order of the individual assemblies of products depending on the operator's process times per assembly task, subjected to the product's assembly precedence constraints and resource assignment constraints.

The operator is recognized based on expected process times, computed by applying an exponentially weighted moving average (EWMA). The measure of computation of this exponentially weighted moving average is explained in Section 4.1. These expected process times are used by different makespan minimization procedures to compute the desired assembly order. First, an online optimization procedure using mixed integer linear programming is discussed in Section 4.2. This procedure is expected to result in the best possible solutions as it provides the exact optimal assembly order for the expected process times subjected to the precedence constraints. Mixed integer linear programming applied on the assembly setup is discussed in detail in Chapter 3. The practical application of online optimization is limited by the required computation time. A set of supervised classification algorithms is

introduced in Section 4.3. All measures of supervised learning generate a function that maps inputs to desired outputs [8]. These algorithms aim to apply a suitable assembly order based on the expected process times, to overcome the computation time limitations of online optimization. For this research, three different functions are applied:

- Euclidean distance: recognizing the operator by computing the Euclidean distance between the expected process times and the predefined process times corresponding to an operator. This algorithm is explained in Subsec 4.3.1.;
- Modal closest: recognizing the operator by computing the distance of the expected process times of individual assembly tasks and the predefined process times corresponding to an operator, label each tasks to the closest located operator and apply the modal value of the labels as recognized operator. This algorithm is explained in Subsection 4.3.2.
- K-nearest neighbors: recognizing the operator by computing the distance between the expected process times and the process times measured during a set of assemblies used to generate a test set. The computed expected process times are examined to the process times in the test set and a set of the closest located process times within the test set is selected. The size of this set is equal to predefined parameter K. The K selected process times from the test set are then subjected to a distance-weighted rule in order to compute the recognized operator. This procedure, including the corresponding distance-weighted rule, is explained in Subsection 4.3.3.

The supervised classification algorithms are explained in the order as depicted above because they are increasingly complex to implement. Evaluating the results of applying those algorithms shows the possible performance gains for applying algorithms of different complexity. If the performance gain for a simple algorithm is equal to the performance gain of a complex algorithm, the simple algorithm is preferred [2].

The online optimization is not bounded by a pool of known operators and their corresponding process times, opposed to the supervised classification algorithms. The online optimization will suggest the optimal assembly order based on the expected process times, only bounded by the precedence constraints and resource assignment. A vast amount of possible assembly orders can be found by online optimization, hence, this procedure is expected to outperform the supervised classification algorithms in minimizing the cycle times of assemblies. Moreover, this flexibility allows a new, unknown operator to start assembling, causing the system to adjust the assembly order to the process times corresponding to the new, unknown operator. Supervised classification algorithms are not able to recognize new operators as they do not receive any feedback corresponding to the unknown operator. The supervised classification algorithms are extended with a clustering algorithm known as *K-means* clustering in order to combine the flexibility of online optimization together with the computational benefits of the supervised classification algorithms. This procedure applies the measured process times to define a set of cluster means, with the size of the set equal to parameter K. Each cluster mean can be defined as an operator. This procedure is explained in detail in Section 4.4.

A discrete event simulation, as described in Chapter 5, is used to examine the effects of the operator recognition procedures depicted above. Stochastic process times with a variety of process time variances are applied for two different operators to study the effectiveness of the procedures dependent on the operators' process time variances. The results of the different techniques are compared and discussed in Chapter 6 to propose the most suitable procedure to apply on the TNO demonstrator setup.

In this chapter, the assembly setup has been explained, together with the performance gain limitations, a theoretical example with the expectation to provide more insight into the possible performance gains, the research question, and the research approach have been discussed. In the next chapter the mixed integer linear programming optimization is discussed.

Chapter 3

Mixed Integer Linear Programming

In Chapter 2 the problem description and research question have been posed. The aim is to minimize the makespan of the total assembly time by optimizing the assembly order of each individual assembly, depending on the operator's process times per assembly task, subjected to the product's assembly precedence constraints and resource assignment constraints. Minimizing the makespan is done by minimizing each cycle time, hence, minimizing the time needed for each individual assembly. In order to minimize the cycle time, the problem can be defined as a mathematical optimization in which some variables are integers, some variables are binary, and some variables are non-integers. Hence, the optimization problem can be formulated as a mixed integer linear programming (MILP) problem. This chapter explains the required input data from the assembly procedure to solve the MILP in Section 3.1, continued by the design variables for the minimization, which are introduced in the Section 3.2. Section 3.3 gives an explanation of the constraints, followed by a formal overview of the optimization in Section 3.4.

3.1 Input Data

In this section, the input data is defined. First of all, each assembly takes T amount of tasks to complete, hence, there exists a set of integer task numbers $t \in \{1, 2, ..., T\}$. The assembly is done by R resources resulting in a set of resources $r \in \{1, 2, ..., R\}$. In the example of an assembly line with one operator and one cobot, some tasks can be performed by the operator, others can be performed the cobot, and the aim of the setup is that in the future tasks can executed by both. The task feasibility on a resource is denoted in the binary variable f, where:

$$f_{rt} = \begin{cases} 1, & \text{if task } t \in \{1, 2, ..., T\} \text{ can be performed on resource } r \in \{1, 2, ..., R\}, \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

Performing a task on a resource requires a certain amount of time d, where the continuous variable d_{rt} expresses the duration of takes t on resource r in seconds. Moreover, some tasks cannot start before other tasks are finished. Hence, an assembly procedure has precedence constraints denoted by the variable P_t representing the set of direct predecessors of task t, e.g., if $P_3 = \{1, 2\}$ then tasks 1 and 2 have to be finished before task 3 can start.
An overview of the input data can be found in Table 3.1. This input data is known and fixed for each assembly procedure and is used together with the design variables explained in Section 3.2 to perform the optimization explained in Section 3.3.

3.2 Design Variables

In Section 3.1, the input data from the assembly system required to minimize the cycle time has been described. Design variables are needed to compute the cycle time and constrain the optimization of the cycle time. Those design variables are introduced and explained in this section.

Table 3.1: Overview of the input data.

Notation	Definition
T	The number of tasks to perform to complete the assembly.
R	The number of resources to which tasks can be assigned.
t, u	Indexes for the tasks $(t, u) \in \{1, 2, ..., T\}$.
r	Index for the resources $r \in \{1, 2, ..., R\}$.
f	Binary variable denoting the feasibility of resource r to perform task t, i.e., $f_{rt} = 1$ iff resource $r \in \{1, 2, ..., R\}$ can perform task $t \in \{1, 2, ..., T\}$.
d	The continuous variable containing the duration of performing a task t on a resource r, i.e., d_{rt} expresses the duration of task $t \in \{1, 2, ..., T\}$ to be performed on resource $r \in \{1, 2, 2..., R\}$.
P	The variable expressing direct predecessors of tasks, i.e., P_t is the set of tasks that have to be finished before task $t \in \{1, 2, ...T\}$ can start.

Each task $t \in \{1, 2, ..., T\}$ has to be assigned to one resource $r \in \{1, 2, ..., R\}$. The binary variable $\mathbf{x^{RT}}$ is used to denote the task assignment to the resources:

$$x_{rt}^{RT} = \begin{cases} 1, & \text{if task } t \in \{1, 2, ..., T\} \text{ is assigned to resource } r \in \{1, 2, ..., R\}, \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

A task is finished at a certain completion time, for which the continuous variable $\mathbf{x^C}$ is used. Hence, x_t^C denotes how long after the start of the first task, task $t \in \{1, 2, ..., T\}$ is completed. The completion time of the task that is completed last equals the maximum completion time of the assembly, denoted by $x^{C\max}$. Hence $x^{C\max}$ is equal to the moment in time at which the assembly is finished.
Tasks that are performed consecutively on the same resource are denoted in the binary variable $\mathbf{x^{TT}}$, denoting that a task $t \in \{1, 2, ..., T\}$ is executed before a task $u \in \{1, 2, ..., T\}$:

$$x_{tu}^{TT} = \begin{cases} 1, & \text{if task } t \in T \text{ is performed before task } u \in \{1, 2, ..., T\} \text{ on the same resource.} \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

Tasks that are performed consecutively either on the same resource or on different resources are captured by the binary variable $\mathbf{x}^{\text{CONS}}$, where:

$$x_{tu}^{\text{CONS}} = \begin{cases} 1, & \text{if task } t \in \{1, 2, ..., T\} \text{ is completed before task } u \neq t \text{ is started,} \\ 0, & \text{otherwise.} \end{cases} \tag{3.4}$$

Moreover, tasks that are performed simultaneously are captured by the binary variable $\mathbf{x}^{\text{SIM}}$, in which:

$$x_{tu}^{\text{SIM}} = \begin{cases} 1, & \text{if task } t \in \{1, 2, ..., T\} \text{ is performed simultaneous with task } u \in \{1, 2, ..., T\}, \\ 0, & \text{otherwise.} \end{cases} \tag{3.5}$$

Finally, conditional constraints can be defined with the Big M-notation [10], which penalizes a variable with a sufficiently large positive value M if desired. For example, if we would like A to be smaller than B if C is equal to 0 then:

$$A < B + M \cdot C,$$

i.e., if $C = 0$ this results in $A < B + 0$ or $A < B$. If $C > 0$ this results in $A < B + M \cdot C$, where M should be sufficiently large to ensure that the constraint is always met.
The value of M and how and when it is applied is explained in Section 3.3 together with how the above described variables are used to constrain the minimization of the cycle time. An overview of the design variables can be found in Table 3.2.

Table 3.2: Description of the used variables.

Notation	Definition
x^{RT}	Binary variable to assign a task to a resource, where $x_{rt}^{RT} = 1$ iff task $t \in \{1, 2, ..., t\}$ is performed on resource $r \in R$.
x^{C}	Continuous variable denoting the completion time of a task, i.e., x_t^C expresses the point in time when task $t \in \{1, 2, ..., T\}$ is completed.
$x^{C\max}$	Value denoting the completion time of the final task. I.e., the assembly is finished at $x^{C\max}$.
x^{TT}	Binary variable denoting tasks performed consecutively on the same resource, i.e., x_{tu}^{TT} denotes task $t \in \{1, 2, ..., T\}$ is performed before task $u \in \{1, 2, ..., T\}$ on the same resource.
x_{tu}^{CONS}	Binary variable indicating task $u \in \{1, 2, ..., T\}$ is performed after task $t \neq u$ is completed.
x_{tu}^{SIM}	Binary variable indicating task $t \in t\{1, 2, ..., T\}$ is performed (partially) simultaneous with task $u \in \{1, 2, ..., T\}$.
M	Large constant used to have a constraint applied only when needed.

3.3 Constraints and Objective Function

The input data and design variables have been described in Section 3.1 and Section 3.2 respectively. Furthermore, an overview of those variables is provided by Table 3.1 and Table 3.2. These variables are used to constrain the optimization objective.
First of all, each task can only be performed on exactly one resource::

$$\sum_{r=1}^{R} x_{rt}^{RT} = 1 \quad \forall t \in \{1, 2, ..., T\}. \tag{3.6}$$

A task $t \in \{1, 2, ..., T\}$ can only be assigned to a resource $r \in \{1, 2, ..., R\}$ if that resource is feasible to perform the task (3.7):

$$\begin{aligned} x_{rt}^{RT} \leq f_{rt} \quad & \forall r \in \{1, 2, .., R\}, \\ & \forall t \in \{1, 2, ..., T\}. \end{aligned} \tag{3.7}$$

A constraint is required which ensures the precedence constraints of the tasks which are subjected to those precedence constraints. Therefore, if task $t \in \{1, 2, ..., T\}$ has to be completed before task $u \in \{1, 2, ..., T\}$ where $t \neq u$ can start, task t is in the set P_u. In essence, if task t is a task that has to be completed before task u can start, the start time of task u is ought to be greater or equal to the completion time of task t (3.8):

$$\begin{aligned} x_t^C \leq x_u^C - \sum_{r=1}^{R} d_{ru} \cdot x_{ru}^{RT} \quad & \forall t \in P_u, \\ & \forall u \in \{1, 2, ..., T\}. \end{aligned} \tag{3.8}$$

In general, the start time of the assembly procedure should be greater than or equal to zero, hence, the completion time of all tasks $t \in \{1, 2, ..., T\}$ minus their duration d_{rt} should be greater than or equal to zero (3.9):

$$0 \leq x_t^C - \sum_{r=1}^{R} d_{rt} \cdot x_{rt}^{RT} \quad \forall t \in \{1, 2, ..., T\}. \tag{3.9}$$

Additionally, each task $t \in \{1, 2, ..., T\}$ has to be finished before or on the maximum completion time $x^{C\max}$ (3.10):

$$x_t^C \leq x^{C\max} \quad \forall t \in \{1, 2, ..., T\}. \tag{3.10}$$

Tasks cannot be performed simultaneously on the same resource, hence, if a task $t \in \{1, 2, ..., T\}$ is performed before a task $u \in \{1, 2, ..., T\}$ on the same resource $r \in \{1, 2, ..., R\}$, the start time of task

u has to be greater than or equal to the completion time of task t (3.11):

$$x_t^C \leq x_u^C - d_{ru} + M(3 - x_{rt}^{RT} - x_{ru}^{RT} - x_{tu}^{TT}) \quad \begin{array}{l} \forall t \in \{1,2,...,T\}, \\ \forall u \in \{1,2,...,T\}, \\ t \neq u \\ \forall r \in \{1,2,...,R\}, \end{array} \tag{3.11}$$

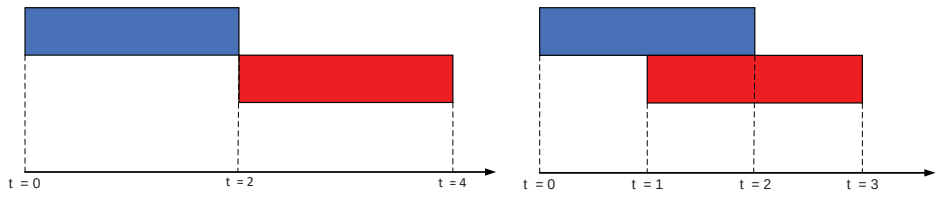
where $x_u^C - d_{ru}$ denotes the start time of task u and $M(3 - x_{rt}^{RT} - x_{ru}^{RT} - x_{tu}^{TT})$ ensures that the constraint is only applied if task $t \in \{1,2,...,T\}$ and task $u \in \{1,2,...,T\}$ are on the same resource $r \in \{1,2,...,R\}$ and task t is performed before task u. If not, $3 - x_{rt}^{RT} - x_{ru}^{RT} - x^{TT}$ is larger than 0 and multiplied with M, where:

$$M = \sum_{t=1}^{T} \max_{1 \leq r \leq R} d_{rt} + \max_{\substack{1 \leq r \leq R \\ 1 \leq t \leq T}} d_{rt}. \tag{3.12}$$

Hence, M is equal to the maximum possible completion time (all tasks being performed consecutively and on the resource that requires the longest completion time for the task) plus the longest possible duration for one task. Using this value for M on constraint (3.11) ensures that if a task t is not performed on the same resource r as task u, constraint (3.11) is fulfilled by M, instead of the start time of task u ($x_u^C - d_{ru}$), allowing parallel assembly between the set of resources $r \in \{1,2,..,R\}$. If tasks are processed on the same resource then they are either performed consecutively in the order $t \rightarrow u$ or the order $u \rightarrow t$. Therefore, an additional constraint is required which ensures that either x_{tu}^{TT} or x_{ut}^{TT} if task t and u are processed on the same resource:

$$-1 + x_{rt}^{RT} + x_{ru}^{RT} \leq x_{tu}^{TT} + x_{ut}^{TT} \quad \begin{array}{l} \forall t \in \{1,2,...,T\} \\ \forall u \in \{1,2,...,T\}, \\ t \neq u, \\ \forall r \in \{1,2,...,R\}. \end{array} \tag{3.13}$$

As can be seen in Figure 1.1, the workspace does not offer specific buffer space, thus, it is not possible to store a part anywhere else than in the workspace of the operator or the cobot. The buffer space is dependent on the size of the product and the specific workspace setup. If the allowed buffer space and the resources' workspaces are filled, blocking occurs. To prevent blocking, a predetermined number of tasks $u \in \{1,2,...,T\}$ is allowed to be processed simultaneously with task $t \in \{1,2,...,T\}$, depending on the amount of buffer space. Two tasks are processed simultaneously if the finish time of the first task exceeds the start time of the second task. The difference between consecutively processed tasks and simultaneously processed tasks is indicated in Figure 3.1, in which the blue tasks is processed on a different resource than the red task. Two tasks are processed consecutively if the finish time of the first task (blue in Figure 3.1 does not exceed the start time of the second task (red in Figure 3.1. This is depicted in Figure 3.1a. If the finish time of the first task exceeds the start time of the second task, the tasks are performed simultaneously, as shown in Figure 3.1b.



(a) Consecutive. (b) Simultaneous.

Figure 3.1: Two tasks with their start and completion times when produced consecutively or simultaneous on two different resources.

A blocking constraint requires a set of simultaneously processed tasks. Tasks are processed simultaneously if they are not processed consecutively, as can be seen in Figure 3.1, therefore, a constraint is added to ensure a set of consecutive tasks is computed:

$$x_u^C - \sum_{r=1}^{R} x_{ru}^{RT} \cdot d_{ru} - x_t^C \leq M x_{tu}^{\text{CONS}} \quad \forall t \in \{1,2,...,T\}, \quad \forall u \in \{1,2,...,T\}. \tag{3.14}$$

However, if the completion time of task u exceeds the start time of u, (3.14) does not force $x_{tu}^{\text{CONS}} = 0$. A constraint is added to ensure $x_{tu}^{\text{CONS}} = 0$ if t and u are not performed consecutively:

$$M(x_{tu}^{\text{CONS}} - 1) \leq x_u^C - \sum_{r=1}^{R} x_{ru}^{RT} \cdot d_{ru} - x_t^C \quad \forall t \in \{1,2,...,T\}, \quad \forall u \in \{1,2,...,T\}. \tag{3.15}$$

If task $t \in \{1,2,...,T\}$ and $u \neq t$ are not performed consecutively, the tasks are performed simultaneously. Hence:

$$x_{tu}^{\text{CONS}} + x_{ut}^{\text{CONS}} = 1 - x_{tu}^{\text{SIM}} \quad \forall t \in \{1,2,...,T\} \quad \forall u \in \{1,2,...,T\} \quad \forall t \neq u. \tag{3.16}$$

For the assembly of the analyzed example as described in Section 2.2, the maximum allowed number of tasks that can be performed simultaneously is assumed to be equal to the number of resources. When applying this optimization to a practical situation, this limit can be adjusted to the real amount of possible parts on the workspace before blocking occurs, by adjusting the value on the right side of (3.17).

$$\sum_{u \in \{1,2,...,T\},\, u \neq t} x_{tu}^{\text{SIM}} \leq R \quad \forall t \in \{1,2,...,T\}. \tag{3.17}$$

If a task $u \in \{1,2,...,T\}$ succeeds task $t \in \{1,2,...,T\}$ on the same resource as task t, then $x_{tu}^{TT} = 1$ (3.13). Hence, if $x_{tu}^{TT} = 1$ then $x_{tu}^{\text{CONS}} = 1$ and $x_{tu}^{\text{SIM}} = 0$ as tasks cannot be performed simultaneously on the same resource. Applying this knowledge, the following constraints can be defined:

$$x_{tu}^{TT} \leq x_{tu}^{\text{CONS}} \quad \forall t \in \{1,2,...,T\}, \quad \forall u \in \{1,2,...,T\}. \tag{3.18}$$

$$x_{tu}^{TT} + x_{tu}^{\text{SIM}} \leq 1 \quad \forall t \in \{1,2,...,T\}, \quad \forall u \in \{1,2,...,T\}. \tag{3.19}$$

The objective of the optimization is to minimize the cycle time of an assembly procedure, thus, the goal is to minimize the maximum completion time $x^{C\max}$. The above described constraints, have to be fulfilled when minimizing the cycle time of the assembly. Minimizing the cycle time is done by finding the minimal possible value for the maximum completion time $x^{C\max}$. Hence, the objective function is defined as:

$$\min_{\mathbf{x^{RT}}, \mathbf{x^{C}}, x^{C\max}, \mathbf{x^{TT}}, \mathbf{x^{\text{CONS}}}, \mathbf{x^{\text{SIM}}}} x^{C\max} + \epsilon \cdot \sum_{t=1}^{T} x_t^C, \tag{3.20}$$

where $x^{C\max}$ is used to minimize the overall assembly time and $\epsilon \cdot \sum_{t=1}^{T} x_t^C$ is used to ensure individual tasks are performed as early as possible and unnecessary waiting times are avoided. Moreover, $\epsilon \cdot \sum_{t=1}^{T} x_t^C$ lowers the computation time as it decreases the amount of optimal solutions. In (3.20), $\epsilon = 1 \cdot 10^{-5}$ so the removal of unnecessary waiting times and increased efficiency are applied without influencing the output value of the objective function significantly.

3.4 MILP overview

An overview of the optimization consisting of the objective function (3.20) and the constraints (3.6) up to (3.19) is given below.

$$\min_{\mathbf{x^{RT}}, \mathbf{x^{C}}, x^{C\max}, \mathbf{x^{TT}}, \mathbf{x}^{\mathrm{CONS}}, \mathbf{x}^{\mathrm{SIM}}} \quad x^{C\max} + \epsilon \cdot \sum_{t=1}^{T} x_t^C$$

subject to

$$\sum_{r=1}^{R} x_{rt}^{RT} = 1 \qquad \forall t \in \{1,2,...,T\}$$

$$x_{rt}^{RT} \leq f_{rt} \qquad \forall r \in \{1,2,..,R\}$$
$$\forall t \in \{1,2,...,T\}$$

$$x_t^C \leq x_u^C - \sum_{r=1}^{R} d_{ru} \cdot x_{ru}^{RT} \qquad \forall t \in P_u$$
$$\forall u \in \{1,2,...,T\}$$

$$0 \leq x_t^C - \sum_{r=1}^{R} d_{rt} \cdot x_{rt}^{RT} \qquad \forall t \in \{1,2,...,T\}$$

$$x_t^C \leq x^{C\max} \qquad \forall t \in \{1,2,...,T\}$$

$$x_t^C \leq x_u^C - d_{ru} + M(3 - x_{rt}^{RT} - x_{ru}^{RT} - x_{tu}^{TT}) \qquad \forall t \in \{1,2,...,T\}$$
$$\forall u \in \{1,2,...,T\}$$
$$t \neq u$$
$$\forall r \in \{1,2,...,R\}$$

$$M = \sum_{t=1}^{T} \max_{1 \leq r \leq R} d_{rt} + \max_{\substack{1 \leq r \leq R \\ 1 \leq t \leq T}} d_{rt}$$

$$-1 + x_{rt}^{RT} + x_{ru}^{RT} \leq x_{tu}^{TT} + x_{ut}^{TT} \qquad \forall t \in \{1,2,...,T\}$$
$$\forall u \in \{1,2,...,T\}$$
$$t \neq u$$
$$\forall r \in \{1,2,...,R\}$$

$$x_u^C - \sum_{r=1}^{R} x_{ru}^{RT} \cdot d_{ru} - x_t^C \leq M x_{tu}^{\mathrm{CONS}} \qquad \forall t \in \{1,2,...,T\}$$
$$\forall u \in \{1,2,...,T\}$$

$$M(x_{tu}^{\mathrm{CONS}} - 1) \leq x_u^C - \sum_{r=1}^{R} x_{ru}^{RT} \cdot d_{ru} - x_t^C \qquad \forall t \in \{1,2,...,T\}$$
$$\forall u \in \{1,2,...,T\}$$

$$x_{tu}^{\mathrm{CONS}} + x_{ut}^{\mathrm{CONS}} = 1 - x_{tu}^{\mathrm{SIM}} \qquad \forall t \in \{1,2,...,T\}$$
$$\forall u \in \{1,2,...,T\}$$
$$\forall u \neq t$$

$$\sum_{u \in \{1,2,...,T\} \wedge u \neq t} x_{tu}^{\mathrm{SIM}} \leq R \qquad \forall t \in \{1,2,...,T\}$$

$$x_{tu}^{TT} \leq x_{tu}^{\mathrm{CONS}} \qquad \forall t \in \{1,2,...,T\}$$
$$\forall u \in \{1,2,...,T\}$$

$$x_{tu}^{TT} + x_{tu}^{\mathrm{SIM}} \leq 1 \qquad \forall t \in \{1,2,...,T\}$$
$$\forall u \in \{1,2,...,T\}$$

The MILP formulation stated above is applied to compute the optimal assembly orders such as the assembly orders for the operators of Figure 2.5. The assembly order, together with the process times, resource assignments and precedence constraints can then be implemented in a discrete event simulation, to show how differences in stochastic process times can alternate the efficiency of a certain assembly order. This optimization is applied to find the optimal assembly order corresponding to process times, as can be seen in the example of Figure 2.4 and Figure 2.5. These assembly orders are needed for the supervised classification algorithms explained in Section 4.3. Moreover, the formulated MILP can be continuously used to optimize the assembly order online on the expected process times, as discussed in Subsection 4.2.

Chapter 4

Makespan Minimization Procedures

In the previous chapter, the MILP has been formulated. By solving this MILP for the process times of the operators, the optimal assembly orders of the operators can be determined. The goal of this research is to minimize the makespan of the assembly by alternating the order in which the assembly assembly tasks are performed to a (near) optimal assembly order. A vast amount of techniques can be applied to optimize the assembly order, but this research is limited to four different techniques. Three of the techniques are supervised classification algorithms and one is an unsupervised algorithm. In this chapter, the four different algorithms are described. The chapter starts with explaining how the expected upcoming process times are predicted. The predictions are based on an exponential weighted moving average (EWMA), based on a first order Kalman filter. The process of predicting the expected process times is equal for all makespan minimization algorithms and is explained in Section 4.1. The procedure and limitations of applying the expected process time to continuously optimize the assembly order by solving the optimization problem discussed in Chapter 3 is discussed in Section 4.2 online. Section 4.3 describes three supervised classification algorithms, which recognize operators known by the assembly system. Finally, Section 4.4 describes how a new, unknown operator can be recognized by the system when applying the supervised classification algorithms.

4.1 Expected Process Times

The operator's process times are assumed to be stochastic process times. Hence, the process time of each assembly step is unknown beforehand. However, a set of expected process times is required to compute a preferred assembly order. This set of expected process times contains the expected process time per assembly task. During each product assembly, the process times of all assembly steps are measured. If one operator is assembling multiple products in series, some variations can occur in the process times. If the next assemblies are performed by a different operator, the system measures different process times. The assembly setup is expected to react to the different working operators by applying the optimal assembly order for the currently recognized operator. Hence, the system should not react to small process time variations, but should react to large process times differences corresponding to operator changes. Filtering out the process time variation can be done by taking the average of the measured process times. However, as the data set of the measured process times increases, the newly measured process times have less impact on the average. Hence, the swiftness with which operator switches are observed decreases, until eventually no operator switches are observed anymore. Often in such a scenario, an exponentially weighted moving average (EWMA) is applied. This EWMA computes a weighted average of the sequence by applying weights that decrease geometrically with the age of the observations [11]. It is commonly applied for smoothing random fluctuations because it is applying a declining weight on older data, it is easy to compute, and low quantity of data is required. A new value of the average is obtained by using the value of the average from the last period and the current value of the variable:

$$\overline{x}(n) = \alpha(n)x(n) + (1 - \alpha(n))\,\overline{x}(n-1), \tag{4.1}$$

where $0 \leq \alpha(n) \leq 1$, $x(n)$ is the current value of the variable, and $\overline{x}(n)$ is the expected value of the variable at period n [12]. When applying this to the assembly procedure to predict the expected process times, the variables x and $\overline{x}(n)$ in (4.1) change to the process times of the assembly steps. This causes (4.1) to change to:

$$\overline{\mathbf{PT}}(n+1) = \boldsymbol{\alpha}(n)\mathbf{PT}(n) + (\mathbf{I} - \boldsymbol{\alpha}(n))\overline{\mathbf{PT}}(n), \tag{4.2}$$

where n is the current assembly with n in $\{1, 2, ..., N\}$ and N is the final assembly. $\overline{\mathbf{PT}}(n+1)$ is the vector containing the expected process times for all assembly steps in the upcoming assembly procedure, $\mathbf{PT}(n)$ is a vector containing the measured process times of the current assembly procedure, and $\boldsymbol{\alpha}(n)$ is a diagonal matrix containing the weight parameters for each assembly step during assembly n.

The value for the parameter $\boldsymbol{\alpha}$ is key for the prediction performance. When a small value for $\boldsymbol{\alpha}$ is applied (i.e., $\boldsymbol{\alpha} \approx \mathbf{0}$), almost all process time variations are filtered out, but the system reacts slowly or not at all to operator switches. When applying a large value for $\boldsymbol{\alpha}$ (i.e., $\boldsymbol{\alpha} \approx \mathbf{I}$), the system reacts immediately to operator switches, but is highly sensitive to process time variations, with a high risk of misinterpreting a process time outlier as an operator switch. Hence, the accuracy of the expected process time is dependent on the value for $\boldsymbol{\alpha}(n)$. Figure 4.1 shows the average difference between the expected process times computed according to (4.2) and the measured process times for the values of $\boldsymbol{\alpha}(n)$ as indicated on the x-axis and different process time variances as indicated in the legend on the right of the figure. The blue dots and dashed, black line indicate the optimal value of $\boldsymbol{\alpha}(n)$ for the different process time variances. Essentially, a high process time variance shows a low optimal value for $\boldsymbol{\alpha}(n)$, while low process time variances show high optimal values for $\boldsymbol{\alpha}(n)$. Producing estimates of unknown variables and observed measurements dependent on statistical noise and other inaccuracies is commonly done using a first order Kalman filter [13]. Such a Kalman filter applies a *Kalman gain* to compute the estimate based on previous estimates and previous measurements. Therefore, the Kalman filter is applicable to compute the desired value for $\boldsymbol{\alpha}$ as is shown next.

The Kalman filter model assumes the true state at time n is evolved from the state at $n-1$ according to:

$$\mathbf{x}(n) = \mathbf{F}(n)\mathbf{x}(n-1) + \mathbf{w}(n) \tag{4.3}$$

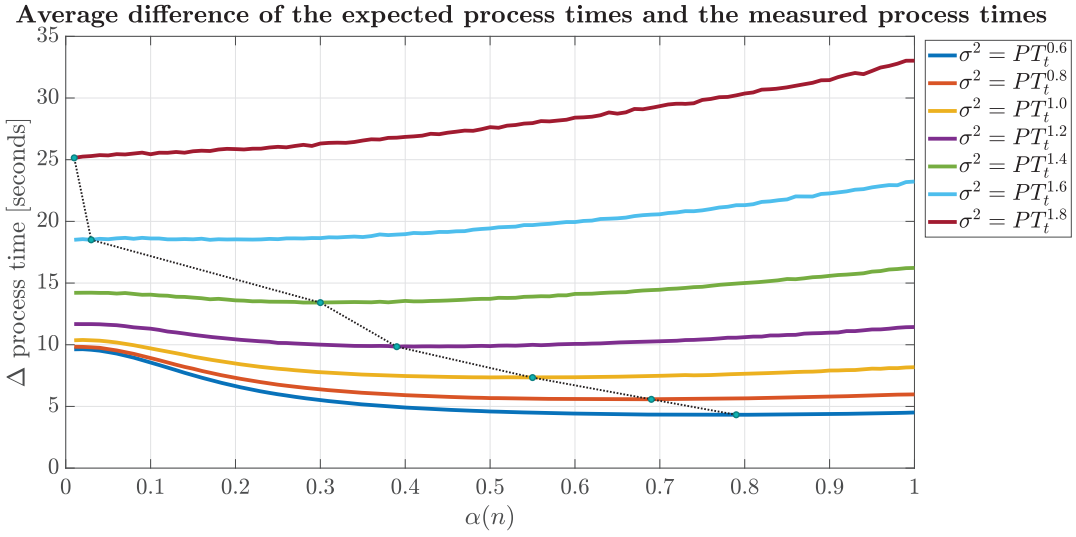


Figure 4.1: Average difference between the expected process computed using (4.2) and the measured process time for the value for $\boldsymbol{\alpha}(n)$ indicated on the x-axis and different process time variances, as indicated in the legend on the right of the figure.

where $\mathbf{x}(n)$ is the output at time period n, $\mathbf{F}(n)$ is the state transition model which is applied to the previous state $\mathbf{x}(n-1)$, and $\mathbf{w}(n)$ is the process noise at time period n [14], [15]. Moreover, a measurement of the true state $x(n)$ at time n is made according to:

$$\mathbf{y}(n) = \mathbf{H}(n)\mathbf{x}(n) + \mathbf{v}(n), \tag{4.4}$$

where $\mathbf{y}(n)$ is the measured output at time period n, $\mathbf{H}(n)$ is the observation model which maps the true state space into the observed space, and $\mathbf{v}(n)$ is the measurement noise at time period n.

In general, the Kalman filter is applied to estimate the current state using the estimate from the previous time step $n-1$ and the current measurement. The state estimate is denoted as $\hat{x}(n|m)$ where n is equal to the current time period and $m \leq n$.

The Kalman filter is applied in two phases [16]. First, a state estimate is predicted based on the previous estimation and the corresponding estimate covariance is predicted. These are known as the *a priori* state estimate:

$$\hat{\mathbf{x}}(n|n-1) = \mathbf{F}(n)\hat{\mathbf{x}}(n-1|n-1), \tag{4.5}$$

and the *a priori* covariance:

$$\mathbf{P}(n|n-1) = \mathbf{F}(n)\mathbf{P}(n-1|n-1)\mathbf{F}^{\mathbf{T}}(n) + \mathbf{Q}(n), \tag{4.6}$$

where $\mathbf{Q}(n)$ is known as the covariance of the process noise $\mathbf{w}(n)$. The second phase is to refine the predicted state estimate by combining the prediction with the current measurement. First, the predicted measurement error $\tilde{\mathbf{y}}(n)$ is computed:

$$\tilde{\mathbf{y}}(n) = \mathbf{y}(n) - \mathbf{H}(n)\hat{\mathbf{x}}(n|n-1), \tag{4.7}$$

and the corresponding error covariance $\mathbf{S}(n)$ is computed:

$$\mathbf{S}(n) = \mathbf{H}(n)\mathbf{P}(n|n-1)\mathbf{H}^{\mathbf{T}}(n) + \mathbf{R}(n), \tag{4.8}$$

where $\mathbf{R}(n)$ is equal to the covariance of the measurement noise $\mathbf{v}(n)$. This pre-fit error covariance is used to compute the optimal Kalman gain $\mathbf{L}(n)$:

$$\mathbf{L}(n) = \mathbf{P}(n|n-1)\mathbf{H}^{\mathbf{T}}(n)\mathbf{S}^{-1}(n). \tag{4.9}$$

This Kalman gain is then applied to compute the updated state estimate, known as the *a posteriori* state estimate:

$$\hat{\mathbf{x}}(n|n) = \hat{\mathbf{x}}(n|n-1) + \mathbf{L}(n)\tilde{\mathbf{y}}(n), \tag{4.10}$$

and the *a posteriori* estimate covariance:

$$\mathbf{P}(n|n) = (\mathbf{I} - \mathbf{L}(n)\mathbf{H}(n))\mathbf{P}(n|n-1). \tag{4.11}$$

Finally, the estimated measurement error $\tilde{\mathbf{y}}(n|n)$ is computed:

$$\tilde{\mathbf{y}}(n|n) = \mathbf{y}(n) - \mathbf{H}(n)\hat{\mathbf{x}}(n|n), \tag{4.12}$$

If a measurement is missed, the system continues with prediction only. On the other hand, if multiple measurements are available at once, the system is updated with all updated measurements.

The Kalman filter is applied on the assembly to predict the process times of the currently working operator. Firstly, it is assumed that a task is performed with consistent process times (i.e., $PT_t(n+1) = PT_t(n)$ for all tasks $t \in \{1, 2, ..., T\}$ where T is the total number of tasks to complete an assembly. However, the measured process times of the tasks assigned to the operator are not deterministic but stochastic, i.e., the process times are not exactly equal for each assembly, but a certain variance of the process times is noticed. This variance is defined as the measurement noise $\mathbf{v}(n)$, thus $\mathbf{v}(n) = \mathbf{PT}(n) - \mathbf{PT}^{\mathrm{op(o)}}$, where o is the currently working operator. This measurement

noise is assumed to be normally distributed with $\mu = 0$ and $\sigma^2(n) = \mathbf{R}(n)$. Applying this knowledge enables the ability to compute $R(n)$:

$$R_t(n) = \frac{1}{n}\sum_{m=1}^{n}\left(PT_t(m) - PT_t^{\mathrm{op}(o)}\right)^2, \quad \forall t \in \{1, 2, ..., T\}.$$

This measurement error can be measured when one operator is assembling the same product long enough for the measurement noise to converge to steady state. Figure 4.2 provides an example to show that the measured variance converges to steady state for all analyzed variances for Task 3 processed by Operator 1. Since the measurement noise converges to a steady state, it is assumed that $\mathbf{R}(n+1) = \mathbf{R}(n) = \mathbf{R}$ for all $n \in \{1, 2, ..., \infty\}$.

The desired assembly setup has more than one operator. The system is not aware of planned operator switches, hence, these operator switches have to be taken into account during the computation of the expected process times. Essentially $PT_t(n+1) = PT_t(n)$ is not only disturbed by the measurement noise $\mathbf{v}(n)$ but can also be disturbed by an operator switch. The noise caused by the operator switches is defined as the process noise $\mathbf{w}(n)$, which is assumed to be normally distributed with $\mu = 0$ and $\sigma^2 = \mathbf{Q}(n)$. When assuming the process times to be deterministic, the variance of the process noise can be computed using the process times of the operators. First, the mean of the process times of all operators is computed:

$$\overline{PT}_t = \frac{\sum_{o=1}^{O} PT_t^{\mathrm{op}(o)}}{O} \quad \forall t \in \{1, 2, ..., T\},$$

where O is the number of known operators. This mean process time $\overline{\mathbf{PT}}$ is used to compute the process noise variance:

$$Q_t(n) = \frac{1}{n}\sum_{m=1}^{n}\left(PT_t(m) - \overline{PT}_t\right)^2, \quad \forall t \in \{1, 2, ..., T\}.$$

Figure 4.3 provides an example to show that the measured variance converges to steady state for all analyzed operator switch frequencies for Task 3 when the process times are deterministic and equal to the process times given in Figure 2.4. The number of operators O in the example is equal to 2. Since the measurement noise converges to a steady state, it is assumed that $\mathbf{Q}(n+1) = \mathbf{Q}(n) = \mathbf{Q}$ for all $n \in \{1, 2, ..., \infty\}$.

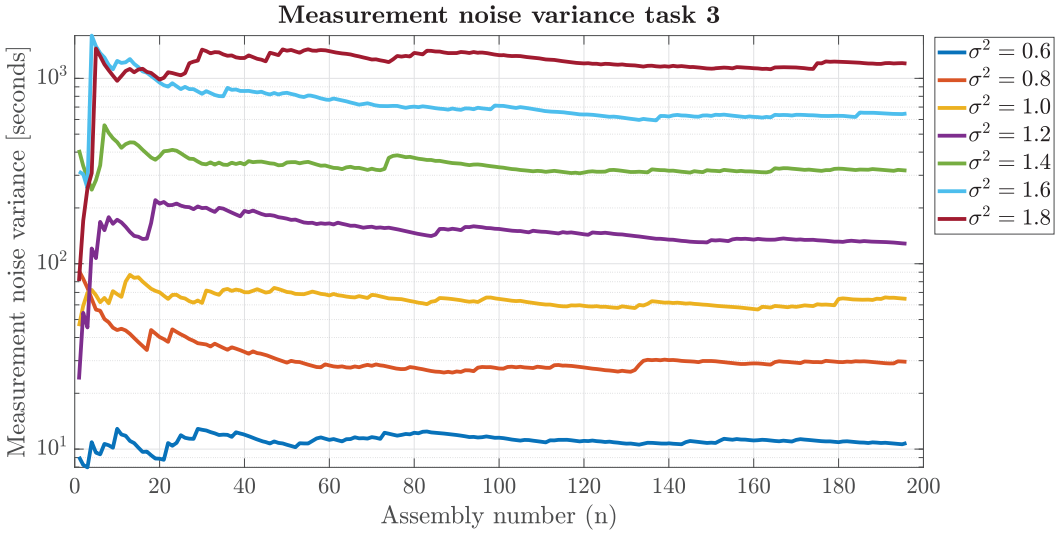


Figure 4.2: Measured process time variance of Operator 1 on Task 3.

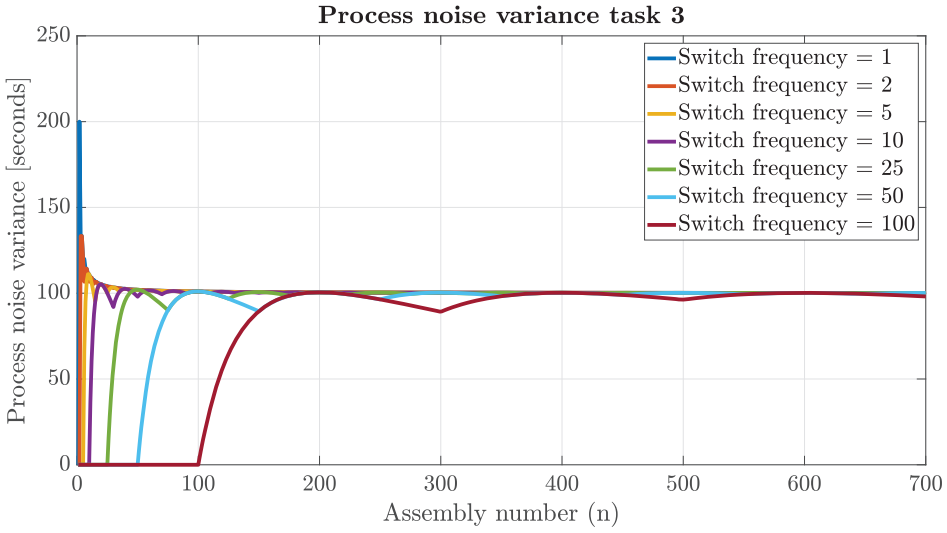


Figure 4.3: Process noise variance of Operator 1 on Task 3.

Furthermore, it is assumed that no state transition model and no observation model need to be applied, hence, $\mathbf{F}(n) = \mathbf{H}(n) = \mathbf{I} \quad \forall n \in \{1, 2, ..., \infty\}$. Substituting $\mathbf{F}(n) = \mathbf{H}(n) = \mathbf{I}$ in (4.3) and (4.4) gives:

$$\begin{aligned}\mathbf{x}(n) &= \mathbf{x}(n-1) + \mathbf{w}(n),\\ \mathbf{y}(n) &= \mathbf{x}(n) + \mathbf{v}(n).\end{aligned}$$

Moreover, substituting $\mathbf{F}(n) = \mathbf{H}(n) = \mathbf{I}$ in (4.5) and (4.6) results in:

$$\begin{aligned}\hat{\mathbf{x}}(n|n-1) &= \hat{\mathbf{x}}(n-1|n-1),\\ \mathbf{P}(n|n-1) &= \mathbf{P}(n-1|n-1) + \mathbf{Q}(n).\end{aligned}$$

Now, the predicted measurement error and corresponding covariance as given by (4.7) becomes:

$$\begin{aligned}\tilde{\mathbf{y}}(n) &= \mathbf{y}(n) - \hat{\mathbf{x}}(n|n-1),\\ &= \mathbf{x}(n) + \mathbf{v}(n) - \hat{\mathbf{x}}(n|n-1),\end{aligned}$$

The variance R convergence to steady state over time. The same holds for $\mathbf{v}$, which converges to a mean equal to $\mathbf{0}$. Thus, $\mathbf{v} = \mathbf{0}$ is applied:

$$\begin{aligned}\tilde{\mathbf{y}}(n) &= \mathbf{x}(n) + \hat{\mathbf{x}}(n|n-1),\\ \mathbf{S}(n) &= \mathbf{P}(n|n-1) + \mathbf{R}(n),\end{aligned}$$

with the Kalman gain as given by (4.9) equal to:

$$\mathbf{L}(n) = \mathbf{P}(n|n-1)\mathbf{S}^{-1}(n).$$

The Kalman gain can be computed by solving the Discrete Algebraic Riccati Equation (DARE)[17]:

$$\mathbf{A}^T\mathbf{X}\mathbf{A} - \mathbf{X} - \left(\mathbf{A}^T\mathbf{X}\mathbf{B} + \mathbf{P}(n|n-1)\right)\left(\mathbf{B}^T\mathbf{X}\mathbf{B} + \mathbf{S}(n)\right)^{-1}\left(\mathbf{A}^T\mathbf{X}\mathbf{B} + \mathbf{P}(n|n-1)\right)^T + \mathbf{Y} = 0, \quad (4.13)$$

where $\mathbf{A} = \mathbf{B} = \mathbf{O}$, $\mathbf{Y} = \mathbf{I}$ and the Kalman gain equal to:

$$\begin{aligned}\mathbf{L}(n) &= \left(\mathbf{B}^T\mathbf{X}\mathbf{B} + \mathbf{S}(n)\right)^{-1}\left(\mathbf{B}^T\mathbf{X}\mathbf{A} + \mathbf{P}^T(n|n-1)\right), &(4.14)\\ &= \mathbf{S}^{-1}(n)\mathbf{P}^T(n|n-1). &(4.15)\end{aligned}$$

The *a posteriori* state estimate of (4.10) becomes:

$$\begin{aligned}\hat{\mathbf{x}}(n|n) &= \hat{\mathbf{x}}(n|n-1) + \mathbf{L}(n)\tilde{\mathbf{y}}(n),\\ &= \hat{\mathbf{x}}(n|n-1) + \mathbf{L}(n)\left(\mathbf{x}(n) - \hat{\mathbf{x}}(n|n-1)\right),\\ &= \mathbf{L}(n)\mathbf{x}(n) + \left(\mathbf{I} - \mathbf{L}(n)\right)\hat{\mathbf{x}}(n|n-1).\end{aligned}$$

Since $\mathbf{F}(n) = \mathbf{I}$ and (4.5):

$$\hat{\mathbf{x}}(n+1|n) = \mathbf{L}(n)\mathbf{x}(n) + \left(\mathbf{I} - \mathbf{L}(n)\right)\hat{\mathbf{x}}(n|n-1).$$

Hence, the *a priori* state estimate of the upcoming time step can be computed by using the *a priori* state of the current time step and the measured output of the current time step. Substituting $\overline{\mathbf{PT}}(n+1) = \hat{\mathbf{x}}(n+1|n)$, $\overline{\mathbf{PT}}(n) = \hat{\mathbf{x}}(n|n-1)$, $\mathbf{x}(n) = \mathbf{PT}(n)$, $\boldsymbol{\alpha}(n) = \mathbf{L}(n)$ in (4.2) indicates that predicting the upcoming process times can be computed using the Kalman filter, which is dependent on the process noise variances and the measurement noise variance. I.e., if the measurement noise variance $\mathbf{R}$ increases, $\mathbf{S}(n)$ increases, causing the Kalman gain to decrease. Moreover, if the process noise variance $\mathbf{Q}$ increases, $\mathbf{P}(n|n-1)$ increases, the Kalman gain increases and the effect of the measurement noise on the Kalman gain decreases. This is intuitive, because if the difference in process times between two operators is low, the distinction between the operators is easily affected by measurement noise, while for large difference between the operators, this measurement noise is much less of an influence.

4.2 Online Optimization

Section 4.1 describes how the data of measured process times is used to compute expected process times for the next time period. The aim of this research is to minimize the makespan of the assemblies by minimizing the cycle time of each assembly by applying a suitable assembly order. An optimal assembly order can be found by solving (3.4), as explained in Chapter 3. This optimization function can be solved by mixed integer linear programming and requires only a limited amount of input:

- the task feasibility on a resource (as defined by (3.1)),
- the precedence constraints,
- the process times.

The task feasibility and precedence constraints are constant for identical products, hence, the process times are the only variable inputs. Hence, the expected process times computed using (4.2) can be fed to the MILP. The output of the MILP is the resource assignment, the expected completion times of all tasks and the expected maximum completion time. These completion times can then be used to compute the assembly order. Thus, optimizing the assembly order online by solving objective function (3.20) using a MILP solver, results in optimal assembly orders for the expected process times $\overline{\mathbf{PT}}(n+1)$.

Solving the MILP is computationally expensive. The more feasible solutions available, the more time a solver needs to find the optimal solution. Thus, products with a relatively high amount of assembly tasks and a low amount of precedence constraints require long computation times. The practical applicability of optimizing the assembly order online solving the MILP is dependent on the required computation time. Therefore, the required computation time dependent on the assembly procedure's complexity is examined. In order to do so, random assembly schemes have been produced for assemblies of 4 tasks up to 14 tasks. All randomly generated assembly schemes have a set of precedence constraints with a size equal to the amount of tasks T. Moreover, each assembly scheme has at least one task that has no precedence constraints, hence, at least one starting task. Moreover, all schemes have one task that is processed last. All other tasks are either directly linked to the final task by precedence constraints (e.g., task 6 in Figure 2.4), or indirectly linke to the final task by precedence constraints (e.g., task 5 in Figure 2.4). Figure 4.4 shows the required computation

times for the MILP solver to solve the objective function when increasing the product's complexity.

On average, the required computation time increases to multiple seconds when assemblies exceed 9 assembly tasks. Therefore, it is not beneficial to wait until an assembly is finished in order to use the measured process times $\mathbf{PT}(n)$ to predict the upcoming process times $\overline{\mathbf{PT}}(n+1)$, but to use the process times of the previous assembly $\mathbf{PT}(n-1)$ to predict the process times for the upcoming assembly $\overline{\mathbf{PT}}(n+1)$. Therefore, the computation time can be equal to the cycle time of assembly n to optimize the assembly order for assembly $n+1$ without delaying the assemblies.

It is denoted in Section 4.1 that if measurements are missed for computing the Kalman a posteriori estimate, the system continues by applying the a priori estimate. Hence $\hat{\mathbf{x}}(n-1|n-1) = \hat{\mathbf{x}}(n-1|n-2)$ in (4.5), therefore, $\hat{\mathbf{x}}(n|n-1) = \hat{\mathbf{x}}(n-1|n-2)$. Thus, when computing the expected process times for assembly procedures with tasks equal to or larger than 9 tasks and a comparable amount of precedence constraints:

$$\overline{\mathbf{PT}}(n+1) = \boldsymbol{\alpha}(n-1)\mathbf{PT}(n-1) + (\mathbf{I} - \boldsymbol{\alpha}(n-1))\,\overline{\mathbf{PT}}(n).$$

When the number of tasks grows to 13, the average computation time grows to a little over three minutes. When the number of tasks is equal to 14, this computation time has grown to more than six minutes on average. The cycle times of the evaluated assemblies of that size is on average comparable to six minutes, hence, the upper bound of the complexity of the assemblies for which the optimization can be applied is equal to 13, if the number of precedence constraints is also equal to 13 or close to 13. When the assembly has more precedence constraints, the computation time becomes shorter, hence, the number of assembly tasks can be increased when applying this procedure. If a task can be performed by either the operator or the cobot, the number of possible solutions increase largely, hence, the computation time grows largely, thus the complexity upper bound decreases.

The computation time of the online optimization procedure limits its practical applicability. A set of computationally cheap supervised classification algorithms is proposed in the next section, with the aim to provide a suitable assembly order based on the expected process times, computed according to Section 4.1 with computation times of negligible length. The supervised classification algorithms all map the expected process times to the optimal assembly order by examining the distance between the expected process times and the process times corresponding to the different operators. A set of three supervised classification algorithms is proposed, all using different functions to map the expected process times to the known process times labeled to the operators. This is discussed in detail in Section 4.3.

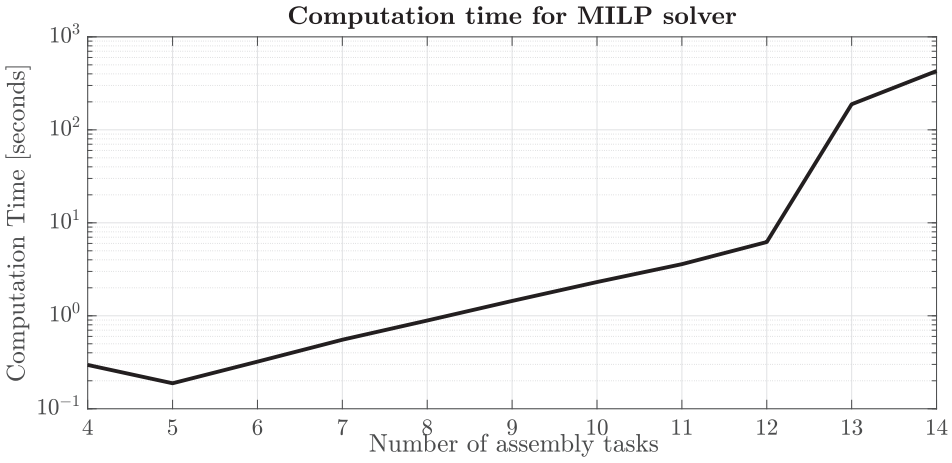


Figure 4.4: Computation time of solving the MILP optimization of Chapter 3 with the predicted process times computed according to Section 4.1 against the number of assembly tasks.

4.3 Supervised Operator Classification

Section 4.1 described how the data of measured process times is used to compute the expected process times for the next time period. Moreover, Section 4.2 described how the expected process times can be used to compute the optimal assembly order corresponding to those expected process times. However, the computation time of the optimization limits the applicability of optimizing the assembly order online. Therefore, a computationally cheaper procedure is desirable to continuously minimize the cycle time of the assemblies. Supervised classification is seen as a suitable solution to compute the assembly orders using computationally cheap algorithms. Supervised classification is performed based on input-output samples [7] by applying a function that maps inputs to desired outputs [8]. The supervised classification algorithms compute the recognized operator by examining the distances between the expected process times and the process times corresponding to a pool of operators known by the system.

This section describes multiple algorithms that can compute a suitable assembly order by applying supervised classification. For this research, three supervised classification approaches are defined and studied. The three approaches have three different levels of complexity, however, they are all easily implemented in the existing system. More complex extensions of these algorithms are available in literature and in practice but are left out of the scope of this research.

The three algorithms defined in this section have in common that they are relatively easy to understand and simple to implement. Opposed to the online optimization procedure, the supervised classification algorithms require more input information. A predefined assembly order for each known operator is required, which can be computed using the MILP formulated in Chapter 3. In order to compute this assembly order, the average process time for each assembly step for each operator has to be known, such as given for the example in Figure 2.4. This information is necessary to provide feedback to the algorithms and can be obtained during a learning period in which an operator assembles a product in a predetermined assembly order and the corresponding process times are measured.

4.3.1 Euclidean Distance Algorithm

Euclidean distance is defined as the straight-line distance between two points in the Euclidean space [2]. If $\mathbf{a}$ is a vector $(a_1, a_2, ..., a_n)$ and $\mathbf{b}$ is a vector $(b_1, b_2, ..., b_n)$, the distance D form $\mathbf{a}$ to $\mathbf{b}$ or vice versa is given by:

$$\begin{aligned} D(\mathbf{a}, \mathbf{b}) = D(\mathbf{b}, \mathbf{a}) &= \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + ... + (a_n - b_n)^2} \\ &= \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}. \end{aligned} \tag{4.16}$$

Equation (4.16) is applied to recognize the currently working operator by computing the Euclidean distance between the predicted process times and the average process times corresponding to the possible working operators $\mathbf{PT}^{\mathbf{op(o)}}$ with $o \in \{1, 2, ..., O\}$ and O equal to the number of known operators. The operator closest to the expected process time according to the Euclidean distance, is defined as the recognized operator.

This methodology can be applied to the example of Figure 2.4. To recap:

- The assembly tasks performed by the operator are tasks [3, 4, 5, 8, 10].
- The corresponding process times for Operator 1 are $\mathbf{PT}^{\mathbf{op1}}$ = [60, 70, 65, 20, 25] seconds respectively.
- The corresponding process times for Operator 2 are $\mathbf{PT}^{\mathbf{op2}}$ = [40, 35, 80, 40, 20] seconds respectively.

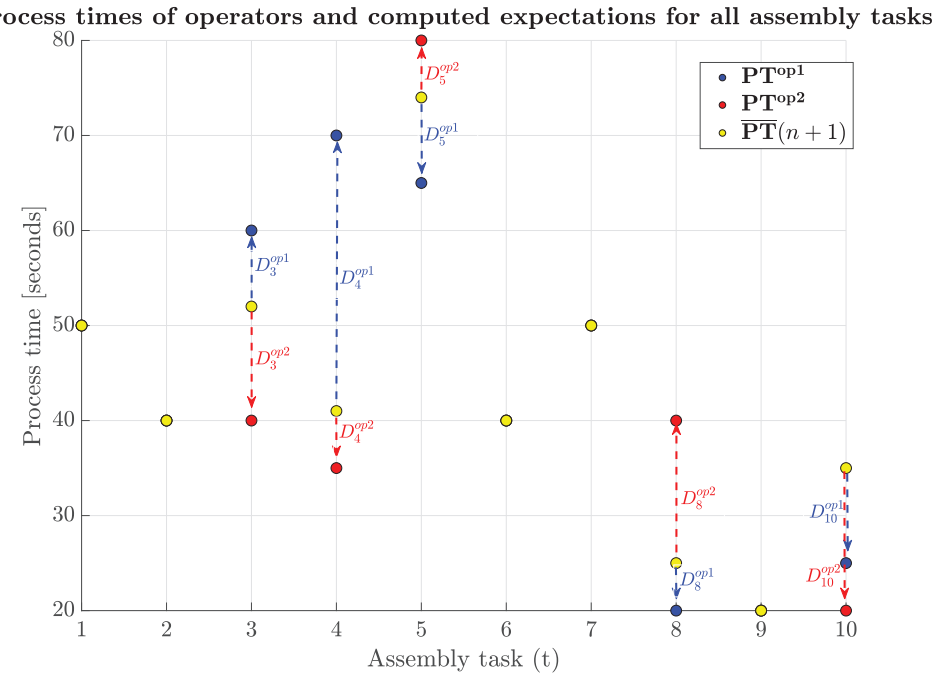


Figure 4.5: Process times of the operators and the computed process times using the Kalman filter based EWMA for assembly $(n+1)$ indicating the distances between the expected process times and the process times corresponding to the operators.

Now, assume the predicted process times for tasks [3, 4, 5, 8, 10] for an assembly to be equal to $\overline{\mathbf{PT}}(n+1) = [52, 41, 74, 25, 35]$ seconds respectively. The process times of the operators and expected process times are indicated in Figure 4.5, together with an indication of the distances between the expected process times and the process times corresponding to the operators. The Euclidean distance algorithm recognizes Operator 2 as currently working operator as is shown in Table 4.1 and applies the assembly order corresponding to operator 2 as computed by solving the MILP of Chapter 3 with the process times denoted in Figure 2.4. In the special case when $D^{\text{op1}}(n+1) = D^{\text{op2}}(n+1)$, the algorithm applies Operator$(n+1)$ = Operator(n). A generic description of this procedure is given in Algorithm 1.

The advantage of the Euclidean distance algorithm is that it is computationally cheap and easy to implement. However, a downside of this algorithm is that one large outlier can be the sole reason to recognize a certain operator. The modal closest algorithm described in the next section aims to filter out such outliers.

Table 4.1: Example of recognizing the working operator using the Euclidean distance algorithm.

Step 1.a:	**Step 1.b**:
$D^{\text{op1}}(n+1) = \sqrt{\sum_{t=1}^{T}(\overline{PT}_t(n+1) - PT_t^{\text{op1}})^2}$	$D^{\text{op2}}(n+1) = \sqrt{\sum_{t=1}^{T}(\overline{PT}_t(n+1) - PT_t^{\text{op2}})^2}$
$D^{\text{op1}}(n+1) = 61$	$D^{\text{op2}}(n+1) = 54$
Step 2:	
Operator$(n+1)$ = Operator $\left(\min[D^{\text{op1}}(n+1),\, D^{\text{op2}}(n+1)]\right) = 2$	

Algorithm 1 Pseudo-code for computing the Euclidean distance algorithm

Require: Assembly number n, expected process time $\overline{\mathbf{PT}}(n+1)$, number of possible operators O, average process times per operator $\mathbf{PT}^{\mathrm{op}(\mathbf{o})}\ \forall\, o \in (1,2,...,O)$, and the number of tasks T.
Ensure: Recognized expected Operator$(n+1)$.
1: **procedure** EUCLIDEAN DISTANCE CLASSIFICATION$(n, \overline{\mathbf{PT}}(n+1), \mathbf{PT}, O, T)$
2: **for all** $o \in (1,2,...,O)$ **do**
3: $D^{\mathrm{op}(o)}(n+1) \leftarrow \sqrt{\sum_{t=1}^{T}(\overline{PT}_t(n+1) - PT_t^{op}(o))^2}$ ▷ (4.16)
4: **end for**
5: Operator$(n+1) \leftarrow$ operator$(\min \mathbf{D}(n+1))$
6: **if** size (Operator$(n+1)) > 1$ **then**
7: Operator$(n+1) \leftarrow$Operator(n)
8: **end if**
9: **end procedure**

4.3.2 Modal Closest Algorithm

The modal closest algorithm compares the predicted process time per assembly step, obtained as explained in Section 4.1, and evaluates them to the process times corresponding to the operators. For each assembly step, the distance between the predicted process time and the process times corresponding to the operators are computed. The predicted process time is linked to the closest operator. This is done for all assembly steps performed by the operators. The most occurring operator to which the process times are linked (i.e., the modal value), is seen as the recognized operator.

This procedure is examined to the same example as analyzed in Subsection 4.3.1. The distance between the predicted process times and the process times corresponding to the operators is evaluated for each assembly task and depicted in Table 4.2, where $\mathbf{D}^{\mathbf{op1}}$ is the vector containing the distances between the predicted process times and the process times corresponding to Operator 1 ($\mathbf{PT}^{\mathbf{op1}}$), $\mathbf{D}^{\mathbf{op2}}$ is the vector containing the distances between the predicted process times and the process times corresponding to Operator 2 ($\mathbf{PT}^{\mathbf{op2}}$). The results of this procedure on that example are shown in Table 4.2.

If an even number of operator tasks result in both operators as modal value, the recognized operator corresponding to the expected process times $\overline{\mathbf{PT}}(n+1)$ is equal to the recognized operator in the period n. Moreover, if an assembly task has process times equally close to the predicted process times for two operators, that assembly task is left out of the modal value computation. A generic description of the procedure is given by Algorithm 2.

The advantage of this algorithm is that outliers in process times do not affect the operator recognizing process drastically. A distance of 1 second is of equal importance as a distance of 50 seconds. Hence,

Table 4.2: Example of recognizing the working operator using the Modal closest algorithm.

Step 1.a:	**Step 1.b**:
$\mathbf{D}^{\mathrm{op1}}(n+1) = \sqrt{(\overline{\mathbf{PT}}(n+1) - \mathbf{PT}^{\mathrm{op1}}) \cdot (\overline{\mathbf{PT}}(n+1) - \mathbf{PT}^{\mathrm{op1}})}$	$\mathbf{D}^{\mathrm{op2}}(n+1) = \sqrt{(\overline{\mathbf{PT}}(n+1) - \mathbf{PT}^{\mathrm{op2}}) \cdot (\overline{\mathbf{PT}}(n+1) - \mathbf{PT}^{\mathrm{op2}})}$
$\mathbf{D}^{\mathrm{op1}}(n+1) = \sqrt{\left(\begin{bmatrix}52\\41\\74\\25\\35\end{bmatrix} - \begin{bmatrix}60\\70\\65\\20\\25\end{bmatrix}\right) \cdot \left(\begin{bmatrix}52\\41\\74\\25\\35\end{bmatrix} - \begin{bmatrix}60\\70\\65\\20\\25\end{bmatrix}\right)}$	$\mathbf{D}^{\mathrm{op2}}(n+1) = \sqrt{\left(\begin{bmatrix}52\\41\\74\\25\\35\end{bmatrix} - \begin{bmatrix}40\\35\\80\\40\\20\end{bmatrix}\right) \cdot \left(\begin{bmatrix}52\\41\\74\\25\\35\end{bmatrix} - \begin{bmatrix}40\\35\\80\\40\\20\end{bmatrix}\right)}$
$\mathbf{D}^{\mathrm{op1}}(n+1) = \begin{bmatrix}8\\29\\9\\5\\10\end{bmatrix}$	$\mathbf{D}^{\mathrm{op2}}(n+1) = \begin{bmatrix}12\\6\\6\\15\\15\end{bmatrix}$
Step 2:	
$\mathrm{Operator}(n+1) = Mo \begin{bmatrix}1\\2\\2\\1\\1\end{bmatrix} = 1$	

Algorithm 2 Pseudo-code for computing the Modal closest algorithm

Require: Assembly number n, expected process time $\overline{\mathbf{PT}}(n+1)$, number of possible operators O, average process times per operator $\mathbf{PT}^{\mathrm{op(o)}}\ \forall\, o \in (1,2,...,O)$, number of tasks T, and resource assignment $\mathbf{x}^{\mathbf{RT}}$.
Ensure: Recognized expected Operator$(n+1)$.

procedure MODAL CLOSEST CLASSIFICATION$(n, \overline{\mathbf{PT}}(n+1), \mathbf{PT}, O, T, \mathbf{x}^{\mathbf{RT}})$
  for all $t \in x_t^{RT} =$ Operator $\quad \forall t \in (1,2,...,T)$ **do**
    for all $o \in (1,2,...,O)$ **do**
      $D_t^{\mathrm{op}(o)}(n+1) \leftarrow \sqrt{(\overline{PT}_t(n+1) - PT_t^{\mathrm{op}(o)})^2}$
    end for
    Task-Operator$_t(n+1) \leftarrow$ operator $(\min \mathbf{D_t}(n+1))$
    if size(Task-Operator)$_t(n+1) > 1$ **then**
      Ignore Task-Operator$_t$
    end if
  end for
  Operator$(n+1) \leftarrow Mo$ (Task-Operator$(n+1)$)
  if size(Operator$(n+1)) > 1$ **then**
    Operator$(n+1) \leftarrow$Operator(n)
  end if
end procedure

if an operator makes a mistake, causing one assembly task to take much more time than expected and therefore become an outlier, the mistake only influences the distance for one assembly task and not for the whole assembly procedure.

4.3.3 K-Nearest Neighbors Algorithm

The third applied classification algorithm is known as *K-nearest neighbors*. It's an instance based learning algorithm which evaluates a data point to the K nearest located data points in a certain test set [2], such as indicated in Figure 4.6. It is notable in Figure 4.6 that for Task 10, both Operator 1 and Operator 2 have measured process times of 23 seconds and 20 seconds. In the figure, only the markers corresponding to Operator 2 are visible, the markers corresponding to Operator 1 are hidden beneath the markers of Operator 2. As discussed in the introduction of this section, the

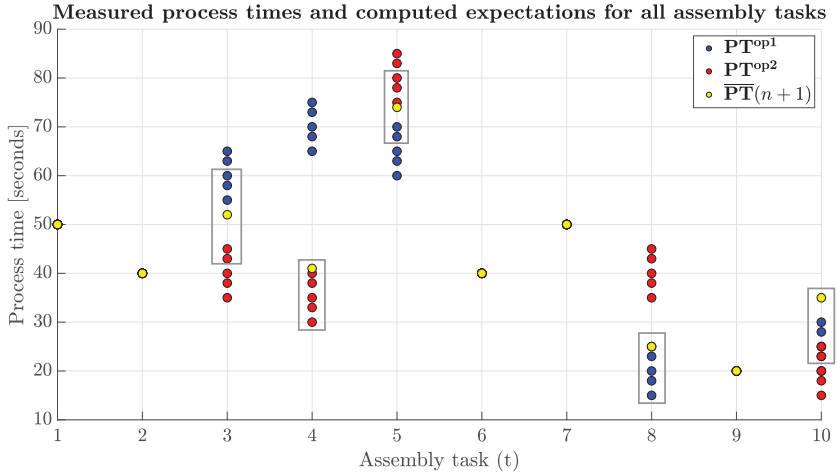


Figure 4.6: Example measurements of process times of the operators and the computed process times using the Kalman filter based EWMA for assembly $(n+1)$ indicating the $K = 5$ nearest neighbors of the expected process times. Two markers corresponding to Operator 1 at Task 10 are hidden below the markers of Operator 2: at 20 seconds and 25 seconds. Therefore, only four markers are visibly selected as neighbors, however, a blue marker is hidden below the bottom neighbor.

process times corresponding to the operators can be obtained during a learning period in which an operator assembles a product in a predetermined assembly order and measuring the corresponding process times. The previously discussed algorithms (Euclidean distance in Subsection 4.3.1 and modal closest in Subsection 4.3.2) require the mean value of the measured process times during that learning period labeled per operator. K-nearest neighbors does not require the mean value of the process times measured during the learning period, but requires the whole set of measured process times and the corresponding operators to which those process times have been labeled during the learning period. This set of data is defined as the test set. The distance between the expected process times of each assembly task and the K process times for the same assembly task in the test set are computed. Hence, for each assembly task, the algorithm searches K nearest neighbors and examines their labels. Next, a weight w_t^k is attributed to the neighbor $k \in (1, 2, ..., K)$ of task t performed by the operator. This weight is defined as:

$$w_t^k = \begin{cases} \frac{D_t^K - D_t^k}{D_t^k - D_t^1}, & D_t^K \neq D_t^1 \\ 1, & D_t^K = D_t^1 \end{cases} \tag{4.17}$$

where w_t^k is the weight corresponding to task $t \in (1, 2, ..., T)$ and neighbor $k \in (1, 2, ..., K)$. Moreover, D_t^k is the distance between the predicted process time for task $t \in (1, 2, ..., T)$ and the process time for that same task by neighbor $k \in (1, 2, ..., K)$ [18].

Applying (4.17) for all tasks results in the number of tasks multiplied with the number of neighbors K weights. For each operator corresponding to selected neighbors in the test set, the sum of the weights is computed. The operator with the highest sum of the weights, is the recognized operator.

A computationally cheaper possible recognition strategy is to compute the K nearest neighbors for all assembly tasks and use the modal value of their corresponding operator to select the recognized operator. The modal value is then computed and used as discussed in Subsection 4.3.2. This process is defined as the *majority rule* [18]. However, this strategy requires an odd value for K and the performance is limited compared to using the distance-weighted rule of (4.17) as can be seen in Figure 4.7. The figure shows the probability to misclassify the operator P_e (the error probability) when applying the modal value and when applying the distance weighted rule against the number of neighbors. This image is based on the evaluation as done in [18] and shows that the distance-weighted rule in general outperforms the majority rule. Moreover, for the results shown in Figure 4.7 a test set of 30 assemblies has been applied. The optimal value for K using the weighted-distance rule can be found at a value larger than $\frac{\text{test set}}{2}$. The size of the test set can continuously increase, as all previously classified assemblies can be used as test set. Hence, the maximum size of the test set is

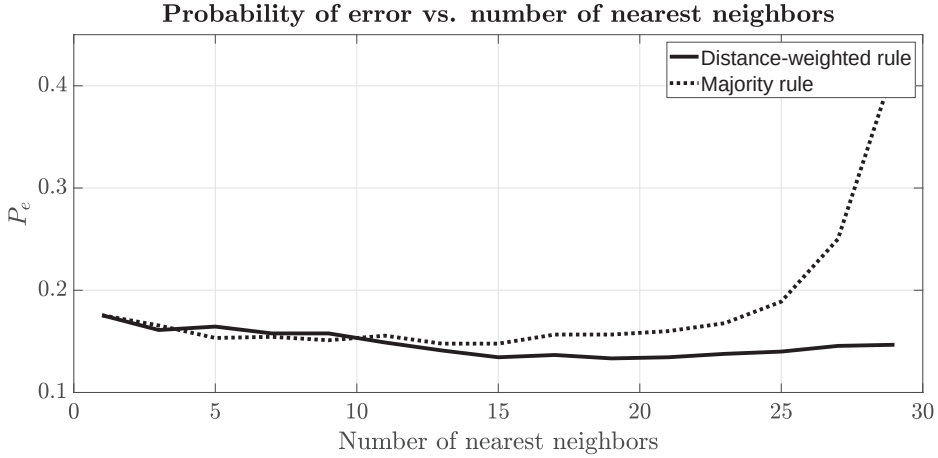


Figure 4.7: Probability of misclassification of the working operator when applying either the weighted distance measure or the majority rule (modal value).

equal to $n-1$. Increasing the test set causes the algorithm to become classified as semi-supervised learning. If the original test set is applied and not increased, this algorithm is classified as supervised learning. A generic description of this procedure is shown in Algorithm 3.

With the three above described classification algorithms, the system is able to recognize the currently working operator. The performance of the algorithms is dependent on the accuracy of the estimated predictions and the process time variance. Additionally, the K-nearest neighbors algorithm's performance is depending on the size of the test set and the value for K, defining the number of nearest neighbors. All three of the algorithms require a known amount of possible operators and an estimate of their average process times. Moreover, K-nearest neighbors requires a test set where measured process times are classified to their corresponding operators. All three algorithms classify the predicted process times to one of the known operators. Opposed to the online optimization procedure as described in Section 4.2 are the supervised learning algorithms not able to apply a new assembly order for an unknown operator. The flexibility to recognize and adapt to new, unknown operators is desirable, therefore, the next section discusses the possibilities of recognizing new operators, e.g., a third operator in the example of Figure 2.4 by applying a clustering algorithm known as K-means clustering.

Algorithm 3 Pseudo-code for computing the K-nearest neighbors classification algorithm

Require: Assembly number n, expected process time $\overline{\mathbf{PT}}(n+1)$, number of possible operators O, measured process times $\mathbf{PT}$ and corresponding operators operator($\mathbf{PT}$), current size of test set $S(n)$, number of neighbors K, and number of tasks T.
Ensure: Recognized expected Operator$(n+1)$.

procedure K-NEAREST NEIGHBORS CLASSIFICATION$(n, \overline{\mathbf{PT}}(n+1), \mathbf{PT}, \text{operator}(\mathbf{PT}), S, K, T)$
  $S(n) \leftarrow n-1$
  for all $t \in (1, 2, ..., T)$ **do**
    for all $i \in (1, 2, ..., n)$ **do**
      $D_t^i(n) \leftarrow \sqrt{(\overline{PT_t} - PT_t(i))^2}$
    end for
    repeat
      remove max $\mathbf{D_t}(n)$
    until size$(\mathbf{D_t}(n)) = K$
    for all $k \in (n(1), n(2), ..., n(K))$ **do**
      if $D_t^K \neq D_t^1$ **then**
        $w_t^k \leftarrow \frac{D_t^k - D_t^K}{D_t^k - D_t^1}$ ▷ (4.17)
      else
        $w_t^k \leftarrow 1$ ▷ (4.17)
      end if
    end for
    for all $o \in (1, 2, ..., O)$ **do**
      $W_t^o \leftarrow 0$
      Weight$^o \leftarrow 0$
      for all $k \in (n(1), n(2), ..., n(K))$ **do**
        if Operator$(k) = o$ **then**
          $W_t^o \leftarrow W_t^o + w_t^o + w_t^k$
        end if
      end for
      Weight$^o \leftarrow$ Weight$^o + W_t^o$
    end for
  end for
  Operator$(n+1) \leftarrow$ operator (min(Weight))
end procedure

4.4 Recognizing and Characterizing a New Operator

In the previous section, three supervised classification algorithms have been discussed. Each of these algorithms is computationally cheap enough to recognize the currently working operator in real time. However, the supervised classification algorithms are not able to define a new, unknown, operator. The ability to recognize and adapt to new operators is desirable to add additional efficiency. Therefore, this section describes how a new operator can be recognized.

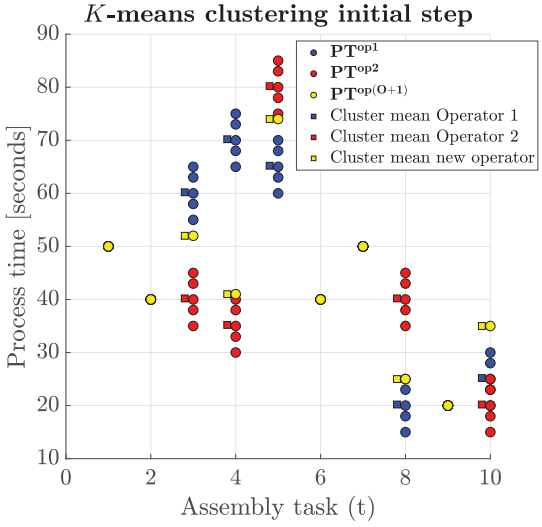
Clustering techniques are applied when data instances are to be divided into groups [2]. In the case of the collaborative assembly setups, these groups are operators. One of the simplest clustering techniques is called *K-means* clustering, which requires a specified number of clusters that are being sought denoted in parameter K [2]. All clusters are assigned to their closest cluster center computed using Euclidean distance (4.16). Next, the mean of the instances in each cluster is computed and finally the whole process is repeated with the new cluster centers [2].

For the assembly setup, the cluster means are equal to the mean process times of the operators, such as provided in Figure 2.4. Hence, each operator and it's labeled process times can be seen as a cluster. Therefore K in the K-means clustering algorithm is equal to the number of operators O. When aiming to recognize a new, unknown operator, the size of K can be increased. I.e., if a new operator is recognized, the total number of operators known by the system O is increased with one, hence, the number of clusters K is increased with one. Determining whether or not the number of operators has to be increased with one, can be done by optimizing the assembly order for measured process times after measuring process times which are notably different to previous measured process times (e.g., a difference larger than twice the standard deviation of the previously recognized operator), by solving the MILP as described in Chapter 3. However, optimizing the assembly order by solving the MILP can be very time consuming as the computation time increases with the assembly procedure complexity, such as shown in Figure 4.4. Therefore, the process time variance for which the optimal assembly order should be reevaluated, should be dependent on the assembly procedure complexity.

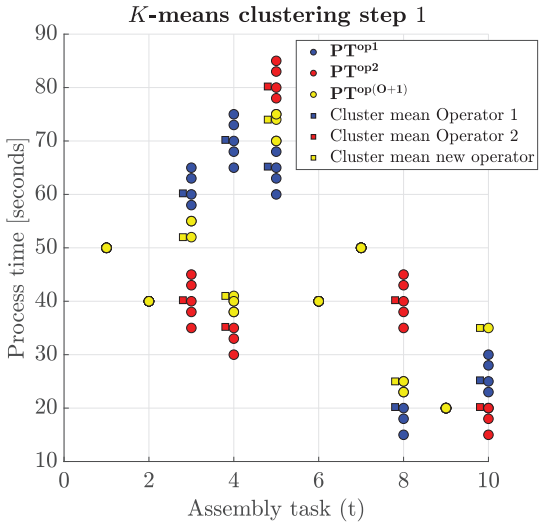
If the optimization results in a new optimal assembly order, the performance gain of applying that assembly order can be evaluated by examining the makespan when applying the new assembly order to the measured process times of period n ($x_{\mathrm{O+1}}^{C\max}(n)$) and examining the makespan when applying the assembly orders of the other operators to the measured process times of period n ($x_{\mathrm{o}}^{C\max}(n) \quad \forall o \in (1, 2, ..., O)$). If the makespan of the new assembly order is sufficiently lower than the makespan of the fastest 'original' assembly orders, the new assembly order is set as the new assembly order, linked to the new operator. Hence, the number of operators is increased with one ($O = O + 1$) (i.e., the value of K for the K-means clustering algorithm is increased with one).

Next, the K-means algorithm is applied to the measured process times with the new value of K. The initial mean values for the cluster centers, are equal to the mean values of the clusters before adding the new operator (i.e., $\mathbf{PT}^{\mathrm{op(o)}} \quad \forall o \in \{1, 2, ..., O\}$). The initial mean value for the new cluster is equal to the measured process times at assembly n, for which the new operator is recognized. Now the process times are assigned to the operator corresponding to the closest cluster center using Euclidean distance (4.16) for all measured process times for this assembly product so far. This can cause the example that a set of process times was previously assigned to operator 1 and is now assigned to operator O. With these possibly changed operator assignments, the new cluster centers are computed. Next, all sets of process times are assigned to their closest cluster center by applying (4.16) again. This process continues until there are no changes to the operator assignments anymore [2]. An example of this procedure is shown in Figure 4.8. Note that the cluster means in this figure are indicated by the square markers, on the left of the measured data of the corresponding cluster. For this example, only two steps are needed to complete the K-means clustering procedure. In addition, it is notable in Figure 4.8 that for Task 10, both Operator 1 and Operator 2 have measured process times of 23 seconds and 20 seconds. Firstly, the cluster means are computed in Figure 4.8a, including the cluster mean of the new instance. Next, all instances are allocated to their closest cluster means in Figure 4.8b. A few instances of task 3, 4, 5, and 8 are allocated to the new cluster. Finally, the new

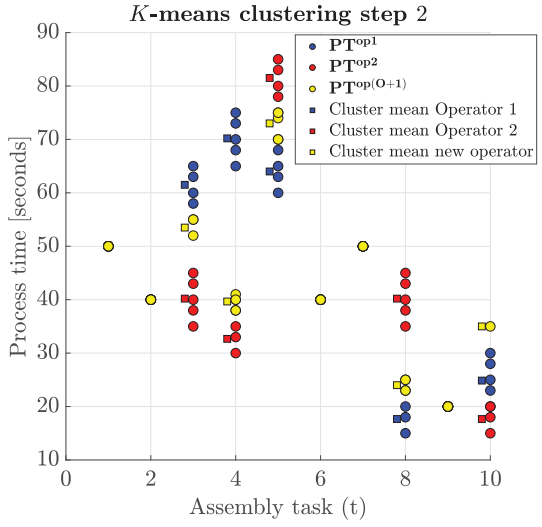
cluster means are computed in Figure 4.8c. All instances are now allocated to their closest cluster mean, hence, the procedure is finished. A generic description of the K-means procedure as described above can be found in Algorithm 4. It is desired to solve the MILP in parallel with the supervised classification algorithm. This allows the continuation of the assembly process with the previously known amount of operators.



(a) Compute the means of the clusters.



(b) Allocate the instances to the closest cluster mean.



(c) Compute the new cluster means.

Figure 4.8: Example of K-means clustering, where the clustering procedure is finished after two steps. Initially, the cluster means are computed as indicated in Figure 4.8a, next the instances are allocated to the closest cluster mean as shown in Figure 4.8b, and finally, the new cluster means are computed as can be seen in Figure 4.8c. With the new cluster means, all instances are allocated to the closest cluster mean, hence, the procedure is finished. Note that the cluster means are indicated with the square markers, on the left of the measured data of the corresponding cluster.

Four algorithms to estimate the preferred assembly order have been described in this chapter. Applying these algorithms to the expected process times as explained in Section 4.1 minimizes the makespan of the assembly procedure in real time. It can be concluded from Figure 4.4 that it is possible to optimize the assembly order online on the expected process times for products with relatively simple assembly procedures. If the number of assembly tasks increases or the number of precedence constraints decreases, the computation time to solve the MILP limits the ability to apply the optimization. The supervised classification algorithms discussed in Section 4.3 are able to decrease the makespan of the assembly by recognizing the currently working operator and applying the predetermined optimal assembly order corresponding to that operator. The online optimization procedure is not bounded by the knowledge of the known operators and their optimal assembly order. This allows a vast range of different assembly orders and insures minimization for an infinite amount of operators. Applying K-means clustering to the supervised classification algorithms adds the capability to learn to recognize and characterize a new operator. In order to gain insight into the possible performance gains by applying the algorithms, discrete event simulations are desirable. In the next chapter, the discrete event model is discussed. Additionally, the results of the applying above described algorithms to the discrete event model explained in Chapter 5 are given in Chapter 6.

Algorithm 4 Pseudo-code for computing the K-Means clustering algorithm

Require: Assembly number n, measured process times $\mathbf{PT}$ and corresponding operators operator($\mathbf{PT}$), number of tasks T, the current number of operators O, assembly repetitions before reevaluating N, and performance gain fraction threshold th.
Ensure: Number of known operators O.

procedure k-MEANS CLUSTERING($n, \mathbf{PT}$, operator($\mathbf{PT}$), T, N)
  Check $\leftarrow 0$
  for all $t \in (1, 2, ..., T)$ **do**
    if $PT_t(n) > \mu_{\mathbf{PT_t}} + C\sigma_{\mathbf{PT_t}}$ or $PT_t(n) < \mu_{\mathbf{PT_t}} - C\sigma_{\mathbf{PT_t}}$ **then**
      Check←Check+1
    end if
  end for
  if Check> 0 **then**
    Define assembly order for assembly n by solving the MILP as formulated in Chapter 3
  end if
  if new assembly order $\neq$ original assembly order and $\min(x^{C\max}_{o\in(1,2,...,O)}(n)) - x^{C\max}_{O+1}(n) > th \cdot \min(x^{C\max}_{o\in(1,2,...,O)}(n))$ **then**
    $O \leftarrow O + 1$
    New Operator $\leftarrow$ true
    Operator$(n) \leftarrow O$
  else
    New Operator $\leftarrow$ false
  end if
  if New Operator $=$ true or $n - n_{\text{New Operator}} >$ Reevaluate **then**
    repeat
      for all $o \in (1, 2, ..., O)$ **do**
        for all $t \in (1, 2, ..., T)$ **do**
          $PT_t^{\text{op}(o)} \leftarrow \frac{\sum_{\forall n\in(\mathbf{Operator}=o)}(PT_t(n))}{size(\mathbf{Operator}=o)}$
        end for
      end for
      for all $i \in (1, 2, ..., n)$ **do**
        for all $o \in (1, 2, ..., o)$ **do**
          $\mathbf{D}^{\text{op}(\mathbf{o})}(i) \leftarrow \sqrt{(\mathbf{PT(i)} - \mathbf{PT}^{\text{op}(\mathbf{o})}) \cdot (\mathbf{PT(i)} - \mathbf{PT}^{\text{op}(\mathbf{o})})}$
        end for
        Operator$(i) \leftarrow \min(\mathbf{D}(i))$
      end for
    until No Operator changes occur anymore
  end if
end procedure

Chapter 5

Discrete Event Model

In Chapter 2 the problem description and research question have been posed. To answer the posed research question, changes have to be made to the existing assembly procedure and new information has to be added to the system. Analyzing the effects of changes to the system can be done using discrete event simulations. Such simulation imitates the operation of the real-world system over time and can be used to estimate the measures of performance of the system with the simulation-generated data [19]. The changes that are made, are changes in the order in which the assembly tasks are performed. These changes are made if the applied algorithm described in Chapter 4 expects the makespan of the assembly can be shortened by applying a more suitable assembly order. This chapter describes the applied discrete event model written in the specification Chi 3, which is applied to analyze the effects of using the algorithms described in Chapter 4 to continuously analyze the expected process times and apply the expected suitable assembly order to the assembly procedure. Section 5.1 briefly describes how communication in Chi 3 works and why it is used. An explanation of the model of the human/cobot collaborative assembly setup can be found in Section 5.2.

5.1 Chi 3

The discrete event simulation is done using Chi 3, which is a specification developed with the focus of simulating operation systems, such as semiconductor factories, assembly and packaging lines, car manufacturing plants, steel foundries, metal processing shops, beer breweries, health care systems, warehouses, and order-picking systems. In Chi 3, a system is abstracted into a model, with cooperating processes, where processes are connected to each other via channels. The channels are used for exchanging material and information. Communication takes place in a synchronous manner, i.e., communication between a sending and a receiving process takes place only when both processes are able to communicate [20].

Models for the analysis of a system should be formal, easily writable, easily readable, and easily extendable. A model features a system and its control as a collection of parallel running processes, communicating with each other using channels [20].

The modeling in Chi 3 is aimed to provide readable models, for non expert readers. In order to ensure readability, some basic modeling techniques are applied. Firstly, the complexity of the model is moved to the functions in the model rather than the processes as much as possible. Functions are more transparent than processes, as they are not dependent on channels and synchronization [21]. The model should avoid deadlock and live-lock. Deadlock occurs when one of two processes remains blocked during synchronization or communication with the other process, live-lock occurs when a process is never blocked by a synchronization or communication action [21]. Library functions are used to increase the compactness of the model.

The next section explains how the human/cobot collaborative assembly line as shown in Figure 1.1 is modeled in Chi 3, using the above described modeling techniques.

5.2 Human/Cobot Collaborative Assembly Setup Model

The previous section provides the modeling and communication techniques applied in the Chi 3 model. This section gives an explanation on how this human/cobot collaborative assembly setup as shown in Figure 1.1 is modeled in Subsection 5.2.1. Moreover, Subsection 5.2.2 describes how the setup is extended to apply the algorithms as described in Chapter 4.

5.2.1 Model of The Current Assembly Setup

Firstly, the existing setup is modeled. In the current setup, assembly tasks can be performed by three resources: a human operator, a cobot, and a tester. One resource is commonly able to perform multiple assembly tasks. In Chi 3 it is common to model processes to which parts are send. For example, a part enters process A, the system is delayed with the required process time to finish process A after which the part is send to process B. The assembly tasks of the products assembled on the assembly setup only require one process. Each process is performed by one of the three resources. Each assembly procedure contains precedence constraints, as shown in Figure 2.1 and Figure 2.4. Some tasks can be processed immediately and some have to wait before others are finished. The assembly is finished when all assembly tasks are finished.

To model the current assembly setup, the resources are modeled as assembly stations that can process assembly tasks for which the station is a feasible resource. Hence, a distinction is made between the operator, the cobot, and the tester, resulting in process stations O, C, and T respectively. Each process station is able to process an assembly task if it is not currently working on an assembly task. Therefore, each process station is modeled to continuously request a new assembly task if, and only if, it is empty. All process stations are only allowed to process one assembly task at once, however, the different process stations are allowed to process in parallel. I.e., the operator can process an assembly task while the cobot is processing a different assembly task. If multiple tasks are to be processed by one resource, some tasks have to wait. Waiting is done in buffers. This allows two different modeling strategies. One would be to model one central buffer which contains all assembly tasks that need to be performed and sends tasks to the first feasible resource that requests a task if the precedence constraints are fulfilled. The other strategy is to use individual buffers for all resources, which contain only the tasks that are to be processed on that resource. The buffer then sends the first task in line to the resource if the resource requests a new task and the precedence constrains of the task are fulfilled. For this model, the latter of the two strategies is chosen, as the first strategy showed difficulties with sending tasks to only one resource and keeping the other resources empty. Even though this would be the preferred strategy as it fulfills the requirement to model easily readable as suggested in Section 5.1, the author lacked the computational skills to overcome this issue of applying one central buffer. Therefore, the seconds strategy is chosen. The two disadvantages of the seconds strategy is that it requires more modeling effort, as it requires multiple buffers instead of one buffer, and it requires a known resource assignment. Hence, if a tasks exist for which both the operator and the cobot are feasible resources, the first strategy would allow the first available resource to process the task while the seconds strategy requires a predetermined resource that processes the task.

Each task needs to be assigned to the buffer corresponding to the assigned resource before it is processed at the resource station. This can only be done if the assigned resource is known for all assembly tasks. Hence, each task contains a list of information of which the assigned resource is part. Moreover, this list contains the task number, the required processing time, and the precedence constraints for each assembly task. These tasks are generated in the generator process $G1$, which generates the assembly tasks for a product one by one in order of which they are processed. Again, two different strategies are possible for the communication of the assembly tasks towards the buffers. One would be to generate the tasks until all tasks of one assembly are generated and then send the tasks to the corresponding buffers, which request new tasks when empty. However, the generator process would then start to generate new tasks for the next assembly and if one of the buffers is empty before the total assembly is finished, it would already receive new assembly tasks from the generator process. This could cause a mix of assembly tasks of multiple assemblies to be present in the buffers. A different strategy is chosen to overcome this issue. For this strategy, two generator

processes $G1$ and $G2$ are applied. Process $G1$ remains to generate assembly tasks in the order of which they are processed. The tasks still contain a list with information of the task number, the assigned resource, the required process time, and the precedence constraints. The tasks is only generated if process $G2$ requests a new task. If a task is requested by $G2$, it is generated by $G1$ and immediately communicated to $G2$. Process $G2$ receives the task information and stores it in a resource depending tuple. Hence, if a task is assigned to the operator, the information of the task is stored in the tuple containing only task information of tasks assigned to the operator. Process $G1$ continuous to process as long as $G2$ requests new tasks. Process $G2$ stops requesting new tasks when it has received the final task of the assembly. Next, $G2$ communicates the information tuples to the corresponding buffers. Now, $G2$ waits with requesting new tasks until it has received new requests from all buffers. The buffers request new tuples when they are empty, hence, when all buffers are empty and therefore the final task is being processed, $G2$ requests new tasks from $G1$.

The model is not yet complete. Currently, the tasks are communicated to the buffer corresponding to their assigned resource. The resource station then process the tasks first in first out (FIFO). Some tasks are bounded by precedence constraints, as is shown in Figure 2.1 and Figure 2.4. Therefore, situations occur in which the operator can not start the next task until the cobot is finished and vice versa. The current model does not communicate the precedence constraints between the different buffers yet, hence, the human is not aware of the tasks that are finished by the cobot and vice versa. A central precedence constraint buffer BPC is added. At the moment a resource finishes a task, it communicates the task number to BPC. This buffer updates the precedence constraints, by removing the finished tasks from the list of precedence constraints for all resources. It then communicates the updated precedence constraints to the resource dependent buffers.

An assembly is finished when all assembly tasks are processed. Finished assemblies are gathered at

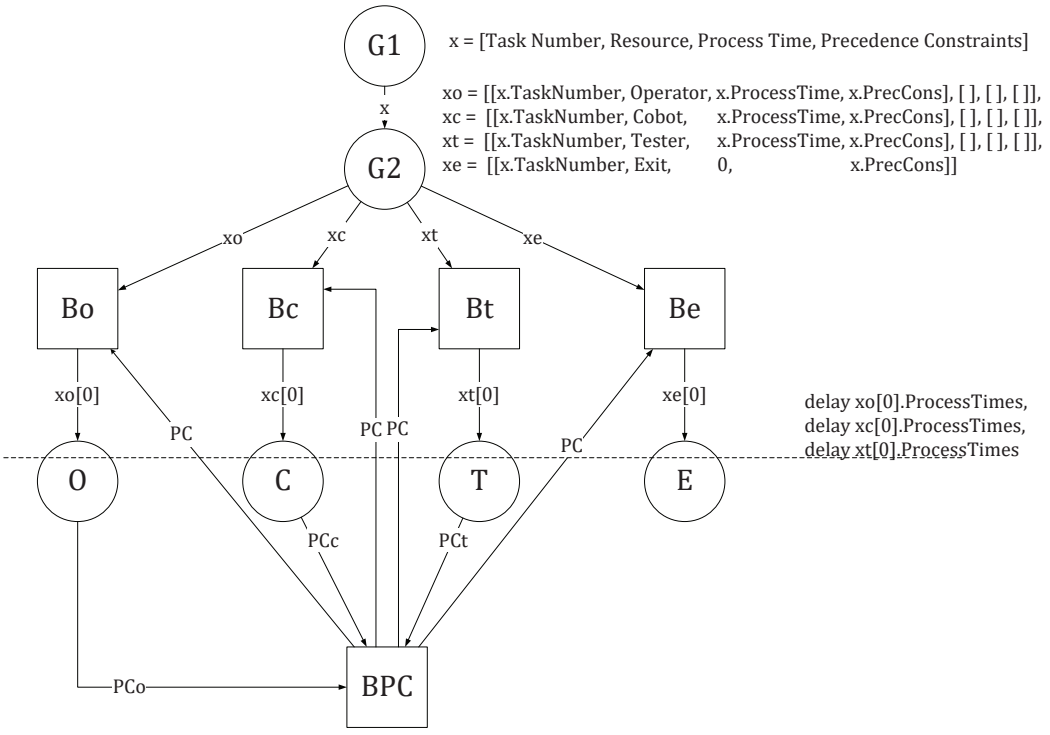


Figure 5.1: Schematic overview of the communication within the discrete event model of the current assembly setup using tuples containing the necessary process information, buffers to store future assembly steps and resource stations to process the assembly tasks.

exit process E. This process counts the number of assemblies it has gathered and tells the setup to stop generating new tasks when the desired amount of assemblies is attained. This exit process requires a task assigned to the exit process, hence, $G2$ requests new tasks from $G1$ until the final real task is generated plus an exit task is generated. This exit task is always generated last, thus $G2$ knows when to stop requesting new tasks if it has received the exit task. Moreover, the exit task has all real assembly tasks as precedence constraint, therefore, it can not start before the final real assembly task is finished. The process time of the exit task is equal to zero. Generator process $G2$ only requests new task from $G1$ if it has received requests from all buffers. The buffer Be corresponding to the exit task has to be empty before $G2$ requests new tasks from $G1$. This results in only one assembly being processed at once.

A schematic overview of the communication within the discrete event model of the current assembly setup can be found in Figure 5.1. The next section is devoted to extending this model into a model capable of applying the algorithms described in Chapter 4.

5.2.2 Model of The Extended Setup

The model of the currently existing assembly setup has been described in the previous subsection. However, the discrete event model is not only used to analyze the current setup, but to analyze the effects of applying the algorithms described in Chapter 4 to the current setup as well. The extension of the model is described in this section.

All algorithms described in Chapter 4 require the expected process time as input data. These expected process times are computed using the measured process times and the previously expected process times. Therefore, the measured process times are stored in the model. The process times are stochastic, hence, they are not known in advance. The process times are measured during processing tasks and are communicated to the central precedence constraint buffer BPC. All resources communicate with BPC causing it to be a useful buffer to store the measured process time data. The model does not require any additional buffers or processes.

The output of all supervised classification algorithms discussed in Chapter 4 is the operator corresponding to the expected process times. Thus, the measured process times stored in BPC are applied to compute the expected process times for the upcoming time period. The output of the online optimization procedure is the optimal assembly order corresponding to the expected process times.

Generator process $G1$ generates tasks in the order in which they are to be assembled. An optimal assembly order is predetermined for each operator possibly recognized by the supervised classification algorithms. Hence, when applying one of the supervised classification algorithms, the generator process has to know which operator is currently recognized. When applying the online optimization, the optimal assembly order has to be known by the generator process.

A set of functions is added to the model to compute the desired assembly order. First of all, a function is added which computes the expected process times based on the measured process times and the previously expected process times. Next, the supervised classification algorithms are added to the model. All functions require the expected process time as input variable. All supervised classification functions result in a recognized operator. Finally, a function is called which computes the desired assembly order based on the recognized operator. The online optimization procedure is not modeled in Chi 3, but in Matlab. Studying the effects of applying the online optimization procedure is done by applying the measured process times from the discrete event simulations and the expected process time computed by the function in the model as input for the online optimization. The online optimization has the assembly order as output, together with the completion times of the assembly tasks, hence, the cycle time of each assembly.

The functions to compute the desired assembly order can either be called from BPC or from $G1$. In the first case, the assembly order has to be communicated to $G1$. In the latter, the measured

process times have to be communicated to $G1$. A third option is to call the functions to compute the expected process time and corresponding recognized operator in BPC and communicate the recognized operator to $G1$. Then $G1$ has to call the function compute assembly order corresponding to the recognized operator. All three options obtain the same results. The decision can be made based on the preference of communication. For this research, the second option is applied, hence, all functions are called from $G1$ after receiving the process times from BPC. BPC communicates the measured process times when the set of precedence constraints is empty, hence, the exit task is the upcoming task to be processed. A schematic overview of the communication within the extended discrete event model can be found in Figure 5.2 below.

With the described discrete event simulations, the effects of applying the makespan minimization techniques of Chapter 4 can be simulated. The results of these simulations are discussed in Chapter 6.

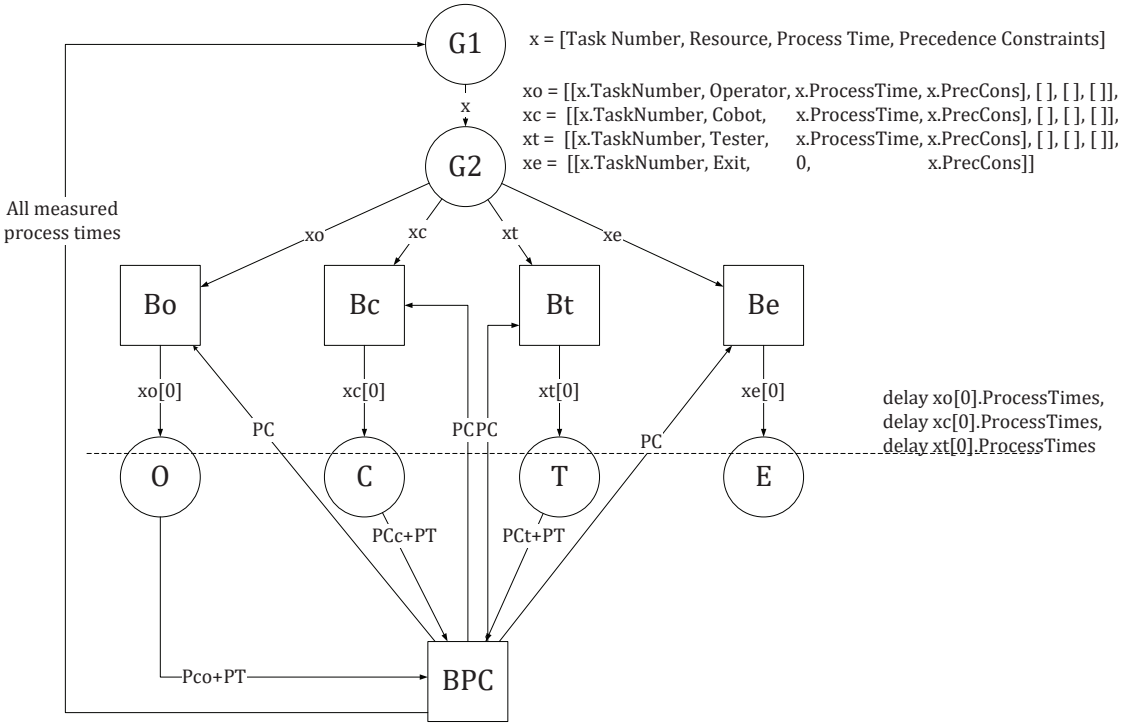


Figure 5.2: Schematic overview of the communication within the extended discrete event model in which PT denotes the measured process times communicated to and stored in BPC.

Chapter 6

Results

The goal of this research is to formulate a suitable optimization procedure to continuously minimize the cycle time of each product assembly such that the makespan of the total assembly procedure is minimized by optimizing the assembly order of the individual assemblies of products depending on the operator's process times per assembly task, subjected to the products assembly precedence constraints and resource assignment constraints. In Chapter 4, four different makespan minimization algorithms were proposed. All algorithms use the expected process times based on a weighted moving average of the previously measured process times as input variable. The previous chapter discussed the applied discrete event model to simulate the assembly of products. This model requires the assembly order as input together with the precedence constraints, the process time distributions and the resource assignments. The assembly order is the only input variable when simulating the example discussed in Section 2.2. The precedence constraints, process time distributions and resource assignments are assumed to be fixed constants.

The proposed algorithms are compared in this chapter to answer the research question. The goal is to find the most suitable machine learning algorithm fitted for the assembly setup. First, the effectiveness of the supervised classification algorithms is examined for a large scope of process time variances on the assembly of the example product as described in Section 2.2 by visualizing the working operator and the recognized operator. This study is performed to gain insight into the differences of the effectiveness of the algorithms and the effects of low and high process time variances on the algorithms. The simulation settings, expectations, and results of this analysis are provided and discussed in Section 6.1. Next, the average cycle time reduction reached by applying the different supervised classification algorithms and reached by applying online optimization opposed to only applying the assembly order corresponding to Operator 1 or Operator 2 are discussed in Section 6.2. Furthermore, the difference between the best possible solution (the optimal assembly order when the true process times are known in advance), the supervised classification algorithms and the online optimization are shown and discussed in the same section in order to add numerical values to the difference between the maximum cycle time reduction and the obtained cycle time reduction per makespan minimization procedure. These results are shown for the same scope of process time variances as the results of Section 6.1, because the effectiveness of the makespan minimization procedures are expected to be dependent on the process time variances. All these previously described simulations and analysis are discussed to gain insight into the most suitable makespan minimization procedure for the evaluated assembly setup.

Section 6.3 provides visualizations comparable to those in Section 6.1 for recognizing a new operator of which the system has no prior knowledge. The capabilities of recognizing new operators by applying the K-means algorithm as described in Section 4.4 is examined and discussed, followed by an analysis of the sensitivity of the effectiveness of the K-means algorithm to the input parameters of the algorithm.

6.1 Currently Working Operator vs. Recognized Operator

This study is performed to gain insight into the differences of the effectiveness of the algorithms and the effects of low and high process time variances on the algorithms. All three supervised classification algorithms are examined on the same data set. First, the simulation parameters are provided in Subsection 6.1.1. The expectations of the results are discussed in Subsection 6.1.2. The results are visualized in images showing the real working operator at a given assembly number n and the recognized operator during the same assembly. The results are shown and discussed in Subsection 6.1.3. The conclusions that can be drawn from the results are discussed in Subsection 6.1.4.

6.1.1 Simulation Parameters

The output of the three algorithms described in Section 4.3 is the recognized working operator for assembly $n + 1$, based on the expected process times computed as described in Section 4.1. The Euclidean distance algorithm of Subsection 4.3.1 and the modal closest algorithm of Subsection 4.3.2 require a mean process time per operator in order to provide feedback to the new expected process times. For the example of Section 2.2, the mean process times are known. They are provided in Figure 2.4. The K-nearest neighbors algorithm of Subsection 4.3.3 requires either a given test set or to use a different supervised classification algorithm for the first assemblies in order to create a test set.

The test set is predefined for analyzing the effectiveness of the K-nearest neighbors algorithm in this section. Essentially, during the first 30 assemblies, the operator recognized by the K-nearest neighbors algorithm is equal to the real working operator in order to create the test set.

The parameters used to examine the effectiveness of the operator recognition supervised classification algorithms are:

- the number of operators $O = 2$;
- operators switch after 15 assemblies;
- the initial K nearest neighbors test set size $S(0) = 30$ (one full set of assemblies for both operators) and the test set increases during the simulation;
- the process time variance of the tasks assigned to the operator is increased after 60 assemblies, applying a gamma distribution with a mean equal to $\mathbf{PT}$ and an increasing variance from $PT_t^{0.6}$ to $PT_t^{1.8}$ for $t \in \{1, 2, ..., T\}$;
- the process time variance of the tasks assigned to the robot is assumed to be equal to 0;
- the total number of assemblies in the simulation $N = 420$.

An indication of the applied process times can be found in Figure 6.1, which shows the measured process time of task 3 for the analyzed set of process time variances.edmc This scope of process time variances is used because it contains both low process time variance and extremely high variance for which the different operators are not distinguishable anymore by looking at the plot. Hence, it allows to examine the effectiveness of the algorithms for a large scope of process time variances.

6.1.2 Expectations

All three supervised algorithms are expected to recognize the working operator swiftly for low process time variances, with little to no errors. Moreover, all algorithms are expected to show an increased amount of errors for increased process time variances. The Euclidean distance algorithm is expected to be the most sensitive to higher process time variances as one outlier can cause the assembly to recognize the wrong operator. The modal closest algorithm is expected to cope better with higher process times, because one outlier only effects the recognized operator on one task, but this can be filtered out by the recognized operators for the other tasks. Further increased process time variances are expected to cause an increased amount of errors for the modal closest algorithm. The K-nearest

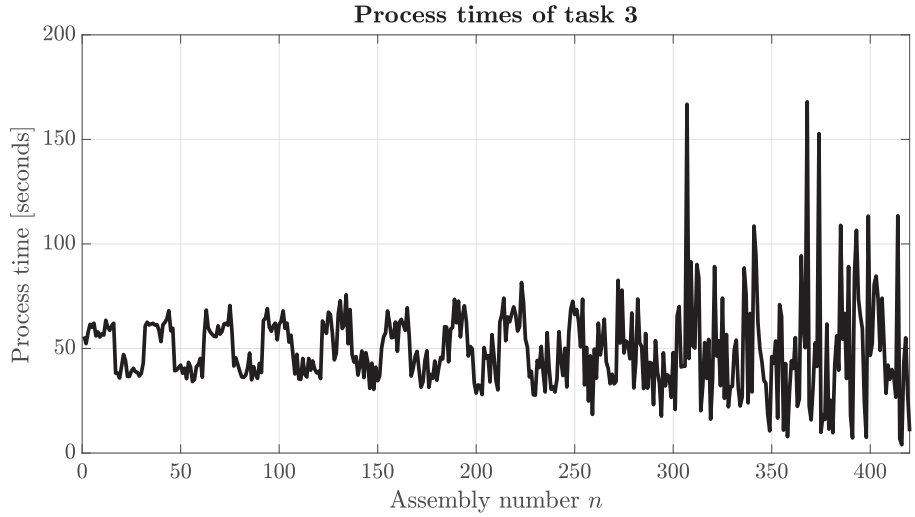


Figure 6.1: Measured process times of task 3 over 420 assemblies, with an increasing process time variance after every 60 assemblies. The operators switch after 15 assemblies.

neighbors algorithm is expected to outperform the other two algorithms, as it applies knowledge of all previously measured process times and applies the K nearest located process times to decide which operator is recognized. This algorithm is expected to cope with higher process time variations than the other two algorithms. For low process time variances, the expected difference is negligible, causing the Euclidean distance to be the most suitable algorithm for low process time variances as it is the simplest to implement.

6.1.3 Results

The results of the simulation are visualized in Figure 6.2, Figure 6.3, Figure 6.4. For each algorithm, the recognized operator at assembly n is shown by the dotted red line. The real operator working at assembly n is shown by the solid blue line.

An operator switch cannot be predicted, therefore, if an operator switch occurs at assembly n, the first possible assembly to which the new operator's assembly order is applied is equal to $n + 1$. The recognized operator in Figure 6.2, Figure 6.3, and Figure 6.4, shown in the dotted red line, shows the assembly n at which the recognized operator is applied. Hence, optimally the dotted red line shows operator switches as close to (but not on) the blue line. The first operator switch for the K-nearest neighbors algorithm, at $n = 16$ does occur directly at the blue line. This happens because a test set is created from $n = 1$ to $n = 30$, hence, this immediate operator switch happens because the system is told about the planned operator switch.

All three algorithms react within three assemblies to the operator switches until the simulation exceeds 230 assemblies, where the modal closest algorithm shows the first wrongly detected operator switch. Just after the 300th assembly, all three algorithms show more regular errors. It remains hard to distinguish which of the three algorithms performs better at those process time variances. At process time variance equal to $PT_t^{1.8}$, the K-nearest neighbors algorithm shows less errors than the other algorithms.

6.1.4 Conclusions

Figures 6.2, 6.3, and 6.4 provide insight into the speed and accuracy of operator recognition by the supervised classification algorithms for different process time variances. Based on the figures, one might have a slight preference for the Euclidean distance algorithm opposed to what was expected as

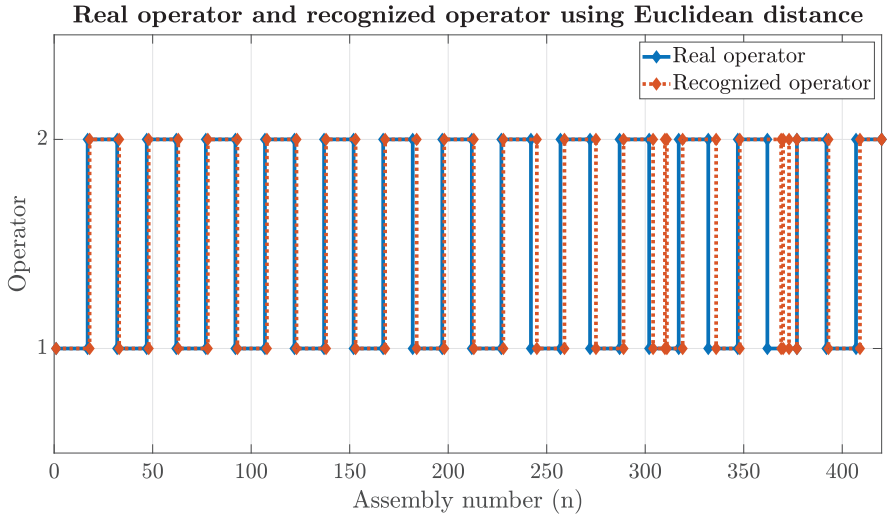


Figure 6.2: Applying the Euclidean distance algorithm as explained in Subsection 4.3.1.

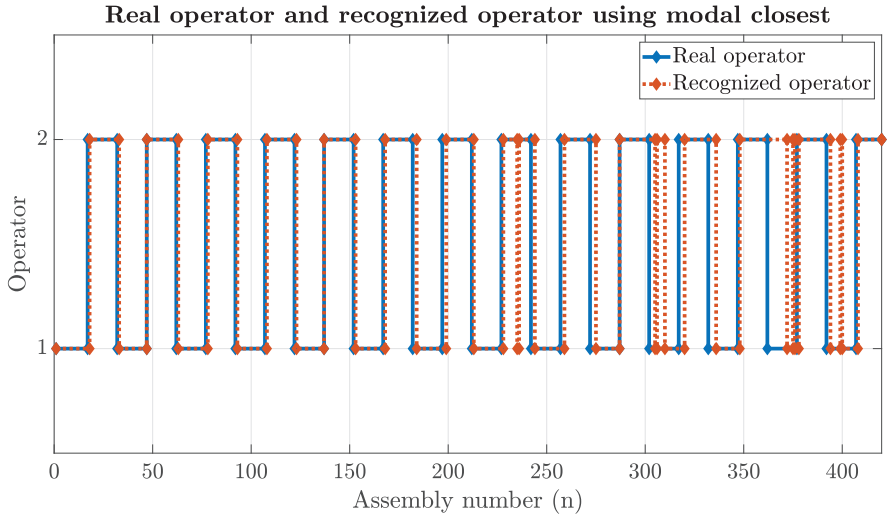


Figure 6.3: Applying the modal closest algorithm as explained in Subsection 4.3.2.

discussed in Subsection 6.1.2. The Euclidean distance algorithm shows comparably accurate results as the Modal closest and K-nearest neighbors algorithm, however, it is the easiest to implement. As stated in Section 2.3, the algorithm which is the easiest to implement is preferred at equal results. K-nearest neighbors shows some better results at process time variances equal to $PT_t^{1.8}$, however, those process time variances are extremely large, as can be seen from Figure 6.1, and therefore very unlikely to occur in real world scenarios.

Even though Figure 6.2, Figure 6.3, Figure 6.4 provide some insight into the speed and accuracy of operator recognition, they do not provide any insight into the results of minimizing the makespan of the assembly. Therefore, it is not yet possible to decide which algorithm fits the assembly setup best. Section 6.2 is devoted to gain insight into the reachable cycle time reduction compared to the assembling without makespan minimization procedure.

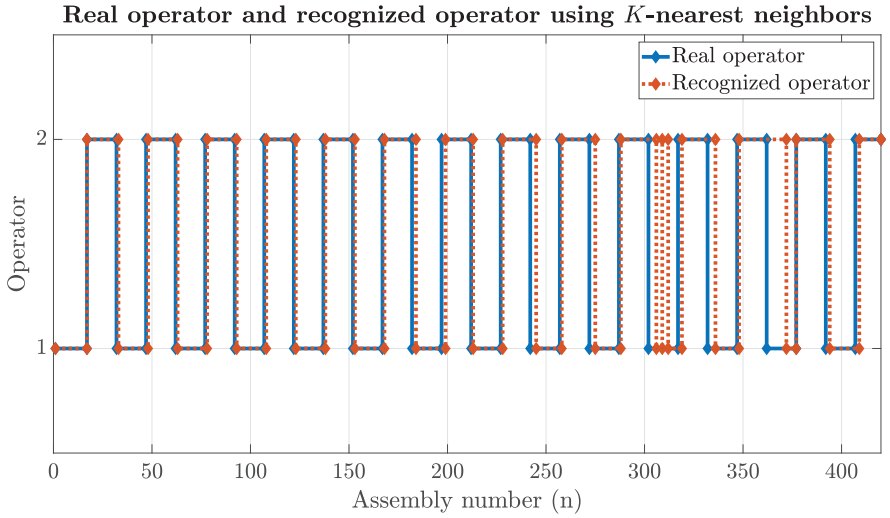


Figure 6.4: Applying the K-nearest neighbors Algorithm as explained in Subsection 4.3.3.

6.2 Cycle Time Reduction Results

This study is performed to add numerical values to the obtainable average cycle reductions by applying the makespan minimization procedures as discussed in Chapter 4. All three supervised classification algorithms are examined together with the online optimization procedure. All are examined on the same data set in order to get a fair and true difference between the results of the procedures. First, the simulation parameters are provided in Subsection 6.2.1. The expectations of the results are discussed in Subsection 6.2.2. The results are visualized in images showing the real working operator at a given assembly number n and the recognized operator during the same assembly. The results are shown and discussed in Subsection 6.2.3. The conclusions that can be drawn from the results are discussed in Subsection 6.2.4.

6.2.1 Simulation Parameters

For this simulation, the following parameters have been used:

- the number of operators remain $O = 2$;
- operator switches remain to happen after every 15 assemblies;
- the initial K-nearest neighbors test set size remains $S(0) = 30$, however, a new test set is generated for each process time variance;
- the process time variance of the tasks assigned to the operator is increased after 300 assemblies, applying a gamma distribution with a mean equal to **PT** and an increasing variance from $PT_t^{0.6}$ to $PT_t^{1.8}$ for $t \in \{1, 2, ..., T\}$;
- the process time variance of the tasks assigned to the cobot remain 0;
- the total number of assemblies in the simulation $N = 2100$.

Operators are recognized by applying the expected process times computed using the Kalman filter as explained in Section 4.1. The online optimization is performed on the same expected process times. The makespan is measured over 300 assemblies for each process time variances, with operator switches after every 15 assemblies. The makespan of those 300 assemblies is divided by 300 to find the average cycle time per assembly, dependent on the production time standard deviation. Additionally, the best possible makespan is computed using online optimization on the measured process times. Thus, the best possible solution is practically unattainable, but the objective is to obtain results as close to the best possible solution as possible.

6.2.2 Expectations

It is expected that all makespan minimization procedures reduce the average cycle time severely. However, based on the results of Subsection 6.1.3, it is expected that all three supervised classification algorithms show equal reductions for low process time variances. Moreover, it is expected that the Euclidean distance and the modal closest algorithm show less cycle time reduction than the K-nearest neighbors classification algorithm at medium and high process. For the Euclidean distance algorithm this expectation is based on the function to classify the process times, as it can be easily influenced by one large outlier in process times. For the modal closest algorithm this expectation is based on the results shown in Subsection 6.1.3. Moreover, based on the results shown in Subsection 6.1.3 and the function to classify the process times, K-nearest neighbors is expected to outperform the other two supervised classification algorithms.

The online optimization procedure is expected to show the largest average cycle time reduction, because it is not bounded by predetermined assembly orders. Hence, it can apply the optimal assembly order for the expected process times of the assembly.

Finally, it is expected that the average cycle time reductions are highest for low process time variances and decrease with increasing process time variance. A clear intersection is expected for which applying any of these makespan minimization procedures becomes counter productive.

6.2.3 Results

Figure 6.5 shows a plot of the average measured cycle times when applying the different makespan minimization procedures, applying the optimized assembly order corresponding to Operator 1, applying the optimized assembly order corresponding to Operator 2, and applying the best possible assembly order. The best possible assembly order is computed using online optimization with the measured process times known in advance.

The average cycle time differences between the supervised classification algorithms are extremely small for low process times. Figure 6.6 is plotted to show the difference between the supervised classification algorithms in more detail. In this figure it is more clear to see that K-nearest neighbors slightly outperforms the other two supervised classification algorithms for all analyzed process time variances.

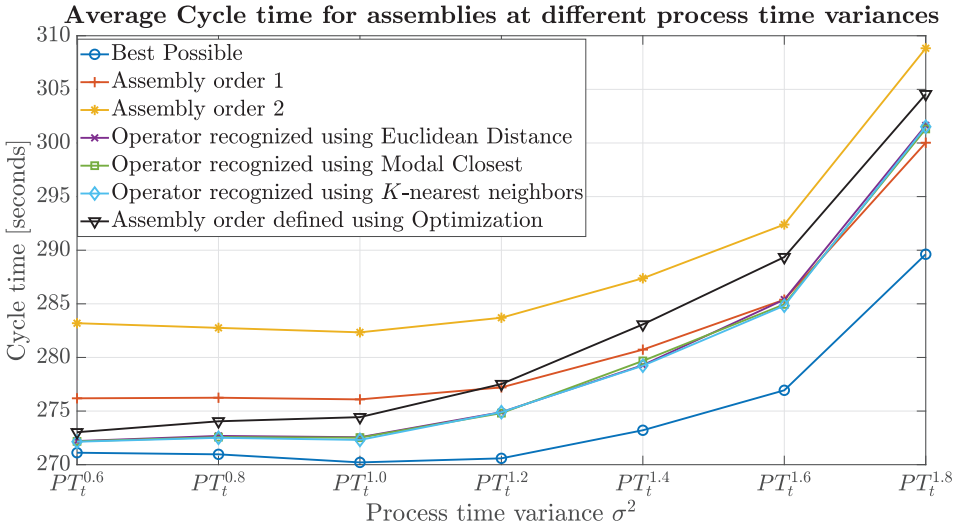


Figure 6.5: Plot of the average measured cycle times when applying the makespan minimization procedures, only the optimized assembly order for Operator 1, only the optimized assembly order for Operator 2, and the best possible solution (when all process times are known in advance).

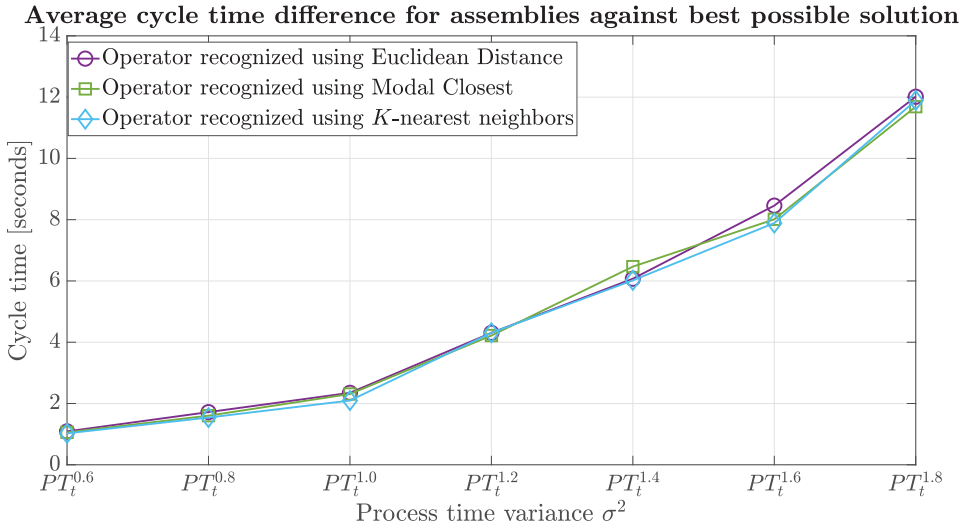


Figure 6.6: Plot of the difference between the average measured cycle times when applying the supervised makespan minimization procedures and the best possible solution.

At higher process time variances such as $\sigma^2 = PT_t^{1.6}$, Figure 6.5 shows that the cycle time reduction between the supervised classification algorithms and applying the assembly order corresponding to Operator 1 has decreased to zero.

The cycle time reduction when applying online optimization is smaller than the cycle time reduction of the supervised classification algorithms. This is true for all examined process time variances. Moreover, the cycle time reduction between the online optimization and applying the assembly order corresponding to Operator 1 has been reduced to zero for $\sigma^2 = PT_t^{1.2}$.

The maximum cycle time reduction in this simulation is obtained by the algorithms at $\sigma^2 = PT_t^{0.6}$ and it is equal to 4 seconds. This is equal to an average cycle time reduction of 1.5%.

6.2.4 Conclusions

The online optimization was expected to outperform the supervised classification algorithms, however, Figure 6.5 shows otherwise. This is caused by the expected process times which are applied as input for the optimization. The expected process times are not equal to the true process times and increasing process time variance increases the error of the expected process times. Therefore, the online optimization computes sub optimal assembly orders. The online optimization procedure outperforms the supervised classification algorithms if, and only if, assembly orders are computed which are different from the assembly orders corresponding to the two operators and these assembly orders fit better to the true process times. The supervised classification algorithms only apply the two assembly orders corresponding to the operators. It can be concluded that the latter is more suitable, especially for high process time variances, as it is less sensitive to the quality of the competed expected process times.

A cycle time reduction of 1.5% was attained during this simulation. This required relatively low process time variances. The cycle time reduction decreases with an increased process time variance. The results of the different supervised classification algorithms are very small. However, the K-nearest neighbors algorithm tends to outperform the other algorithms slightly. It is therefore recommended to implement the K-nearest neighbors algorithm.

6.3 New Operator Recognition

This study is performed to gain insight into the applicability of K-means clustering on the assembly setup in order to recognize new operators of which the system has no prior knowledge. All three supervised classification algorithms are extended with K-means clustering and the same data set and parameters are applied for all algorithms. The results are compared in order to obtain information about which algorithm is the most suitable to extend with K-means clustering. Moreover, the clustering algorithm's sensitivity to the input parameters is shown and discussed.

First, the simulation parameters for the K-means clustering applicability analysis are provided in Subsection 6.3.1. Next, the expectations of the results are discussed in Subsection 6.3.2. The results of the simulation are shown and discussed in Subsection 6.3.3. The clustering algorithm's sensitivity to the input parameters is shown and discussed in Subsection 6.3.4. Finally, the conclusions are discussed in Subsection 6.3.5.

6.3.1 Simulation Parameters

For the initial simulation the parameter settings are set to:

- initial number of operators known by the system $O(o) = 2$;
- operator switches happen after 15 assemblies in the order operator 1 $\rightarrow$ operator 2 $\rightarrow$ operator 3 $\rightarrow$ operator 1 $\rightarrow$ operator 2 $\rightarrow$ operator 3 $\rightarrow$ ...;
- the process times of the new operator are $\mathbf{PT}^{\mathbf{op3}} = [50, 40, 80, 50, 90, 40, 50, 60, 20, 15]$ seconds;
- the initial kNN test set is equal to $S(0) = 30$;
- the process time variances are equal $PT_t^{0.6}$ for all $t \in \{1, 2, ..., T\}$;
- the process time variances of the tasks assigned to the cobot are equal to 0;
- the total number of assemblies in the simulation $N = 270$;
- the first possible assembly at which a new operator is allowed to be recognized is $n = 31$, hence, when the kNN test set is completed;
- a new assembly order is proposed when $PT_t(n) > PT^{\text{op}(o)} + C\sigma_{\mathbf{PT_t}}$ or $PT_t(n) < PT^{\text{op}(o)} - C\sigma_{\mathbf{PT_t}}$ with $t \in \{1, 2, ..., T\}$, $o \in \{1, 2, ..., O\}$ and $C = 2$;
- the threshold performance gain th at which a new proposed assembly order is accepted is set to 10 seconds;
- the cluster centers are reevaluated when a new operator is recognized and every 50 assemblies after recognizing a new operator.

The algorithms are aware of the process times and optimal assembly orders for Operator 1 and Operator 2. The algorithms have no knowledge of the process times and optimal assembly order of Operator 3.

6.3.2 Expectations

In Section 6.1 and Section 6.2 it was concluded that the performance differences between the three supervised classification algorithms are very small. The same is expected for the applicability of extending the algorithm with the K-means algorithms in order to recognize new operators.

6.3.3 Results

Figure 6.7 shows that applying the K-means algorithm to recognize the new operator by extending the Euclidean distance algorithm results in the system immediately recognizing a new operator at $n = 31$, as the new operator starts at that assembly number and the process times of that operator are severely different to the process times of the other operators and the process time variances are small. Exactly the same holds for applying the modal closest algorithm in Figure 6.8. Just after the 120th assembly and at the 215th assembly both algorithms show errors in recognizing operator switches. Applying the K-nearest neighbors algorithm requires an additional constraint to the K-means and K-nearest neighbors algorithms: after recognizing a new operator, the system applies the Euclidean distance algorithm for the next 10 time periods in order to generate a new test set corresponding to the new operator. Figure 6.9 shows that the K-nearest neighbors algorithm initially recognizes the new operator perfectly. However, when Operator 3 starts working for the second time,

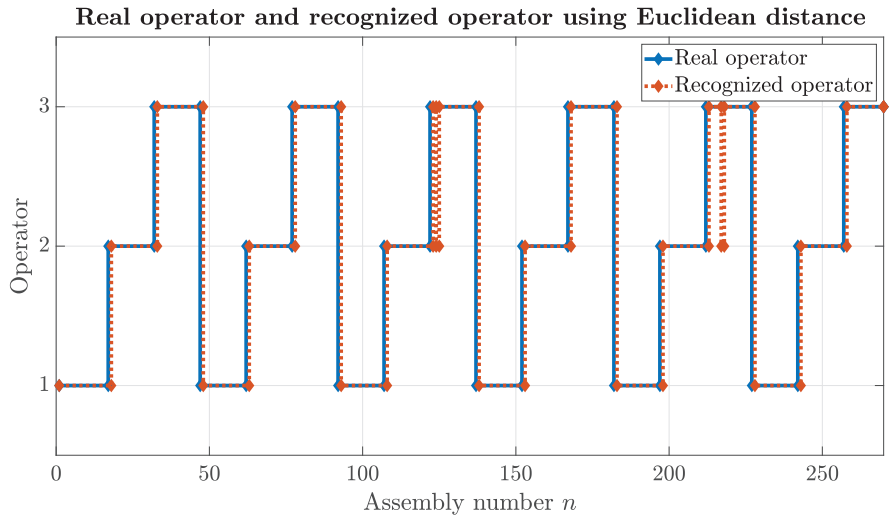


Figure 6.7: Recognizing a new operator by extending the Euclidean distance algorithm with the K-means clustering algorithm.

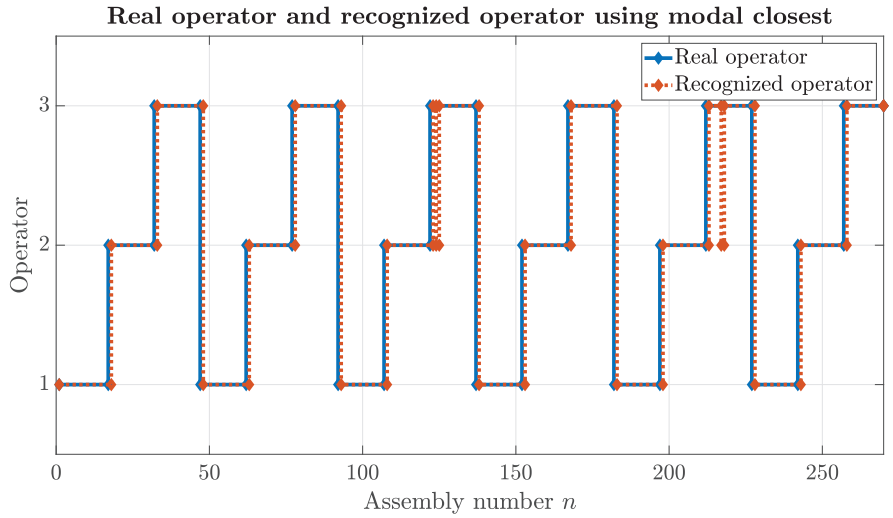


Figure 6.8: Recognizing a new operator by extending the modal closest algorithm with the K-means clustering algorithm.

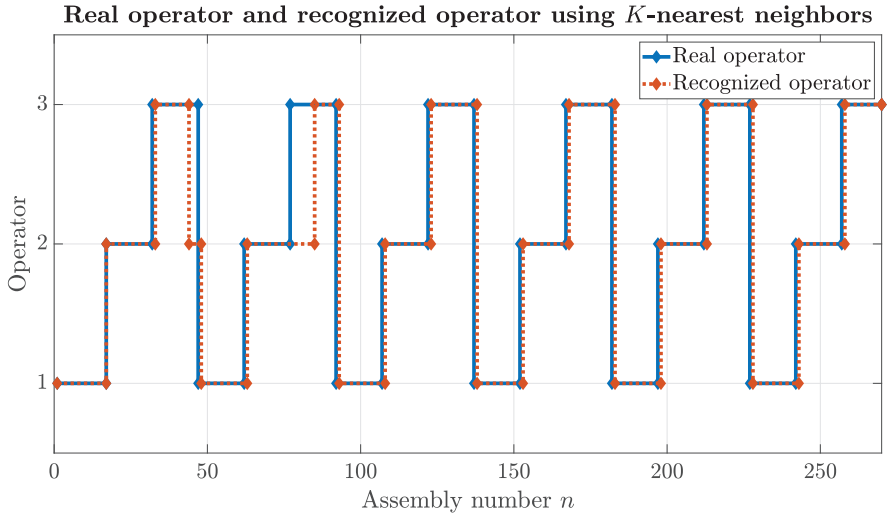


Figure 6.9: Recognizing a new operator by extending the K-nearest neighbors algorithm with the K-means clustering algorithm.

the K-nearest neighbors algorithm has difficulties with detecting the new operator. However, when the 81st assembly occurs, the means of the classes are reevaluated, causing the quality of the test set and the means to improve and causing operator recognition to improve.

6.3.4 Sensitivity and Limitations

Evaluating the possibility to recognize a new, unknown operator is done with low process time variances ($\sigma^2 = PT_t^{0.6}$). The next simulations show the algorithms' sensitivity to parameter settings. The K-nearest neighbors algorithm has shown to be the most suitable procedure as discussed in Section 6.2. Therefore, the next simulations have been done by applying K-means to the K-nearest neighbors algorithm.

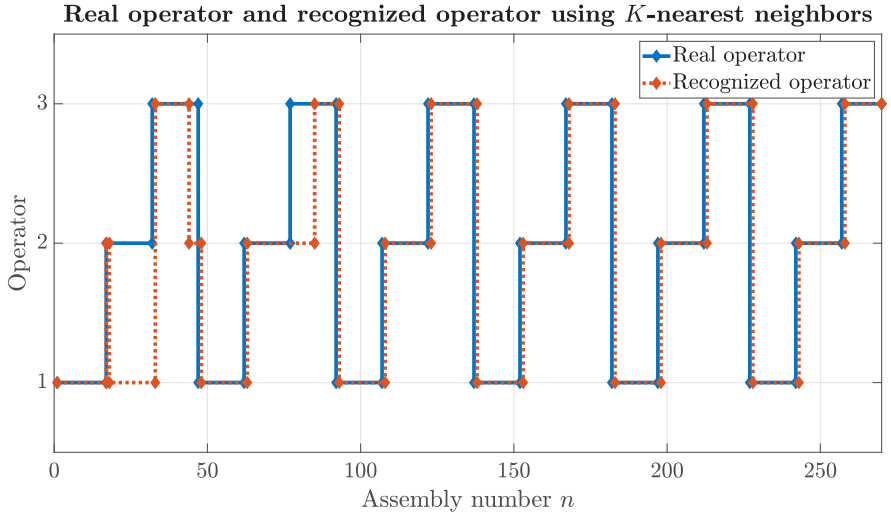


Figure 6.10: Allowing the first new operator to be recognized after 17 assemblies.

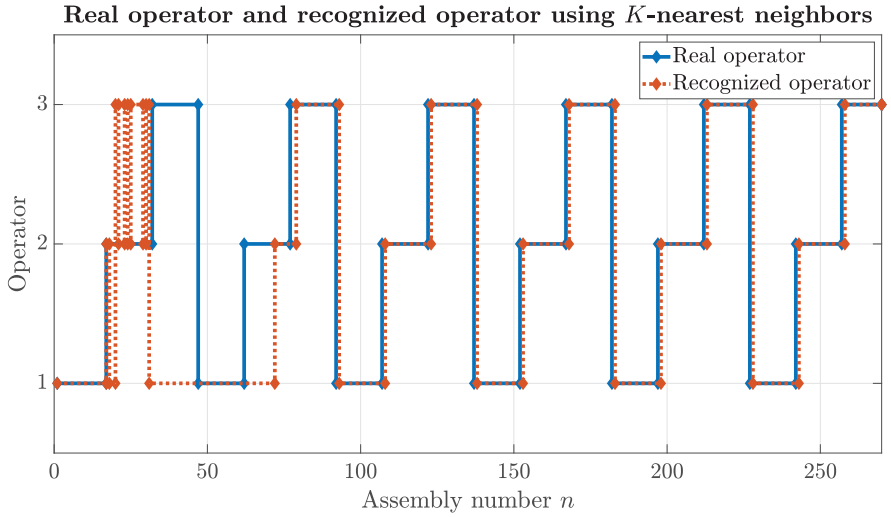


Figure 6.11: The performance gain threshold equal to 5 seconds.

First of all, the results of the K-means algorithm are sensitive to the moment at which the first new operator is allowed to be recognized. In Figure 6.10 the first new operator is allowed to be recognized at assembly $n = 17$, hence, during the time at which Operator 2 is assembling, Operator 3 can be recognized. This also causes the K-nearest neighbors test set to consist of 15 assembly process times labeled to Operator 1 and only 2 assembly process times labeled to Operator 2. The cluster centers have been reevaluated 50 assemblies after the new operator was recognized. This figure shows that at first the test set has to little process time data labeled to Operator 2 to recognize Operator 2. However, after recognizing Operator 3, the cluster means are reevaluated, causing the quality of the test set to increase and containing more data labeled to Operator 2.

Changing the performance gain threshold parameter from 10 seconds to 5 seconds causes the risk of recognizing an outlier in process times as a new optimal assembly order. This is simulated an imaged in Figure 6.11. The image shows that after the test set is completed at $n = 17$, the system

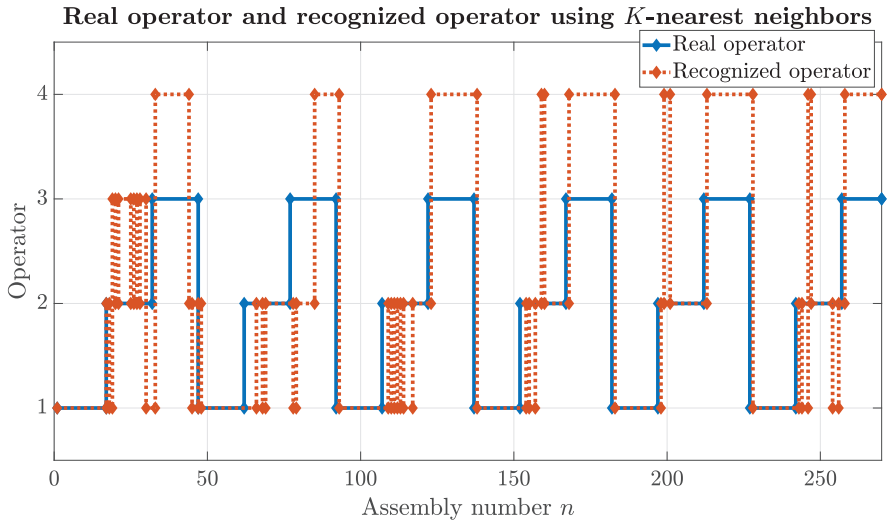


Figure 6.12: Process time variance $\sigma^2 = PT_t^{1.0}$.

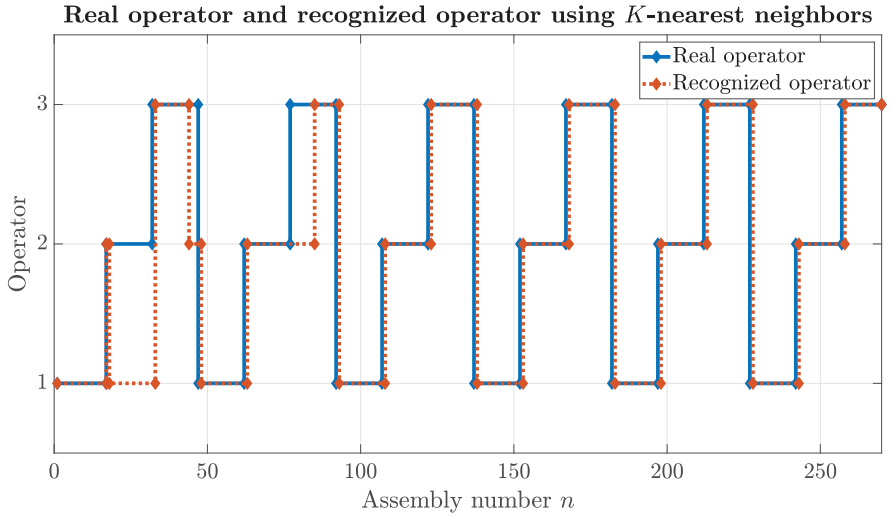


Figure 6.13: The performance gain threshold equal to 10 seconds.

has difficulties with recognizing the right operator. However, 50 assemblies after recognizing a new operator, the cluster means are reevaluated, causing the test set to be improved and the operators to be recognized as expected.

Increasing the process time variance also increases the difficulty to recognize a new operator, as can be seen by the results of Figure 6.12, where the process time variances are increased from $\sigma^2 = PT_t^{0.6}$ to $\sigma^2 = PT_t^{1.0}$. This image shows that an increased process time variance also increases the difficulty of recognizing the right operator. In this case, an additional operator class has been defined, while no additional operator is really working. The performance gain threshold is increased back to 10 seconds for the results of Figure 6.13, showing that this solves the issue of recognizing the additional operator. Unfortunately, higher process time variances increase the parameterization difficulties up to a point where K-means is not usable anymore in the way explained and discussed in this report.

6.3.5 Conclusions

It can be concluded from the results discussed in Subsection 6.3.3 that all three supervised classification algorithms are applicable to be extended with K-means clustering. This allows the capabilities to recognize new operators with which the assembly setup has no prior experience. However, Subsection 6.3.4 shows that K-means clustering is sensitive to the applied input parameters. The results of the attainable effectiveness of K-means clustering and the recognition of new operators requires additional research.

The results overall have shown that the makespan of the assembly can be minimized by applying a suitable assembly order. Supervised classification algorithms are very applicable for this matter as they show near optimal results, especially when process time variances are not extremely large. The K-nearest neighbors algorithm is the most suitable algorithm, however, applying the Euclidean distance algorithm or the modal closest algorithm shows similar results. A properly parameterized K-means clustering algorithm is applicable to recognize and define new operators. All three supervised classification algorithms are applicable to be extended with K-means clustering. All of the proposed algorithms are easy to implement, allowing the makespan minimization procedure to be implemented with low effort.

Chapter 7

Conclusions and Recommendations

The previous chapters have been devoted to explaining the existing situation and stating the research question, defining the approach to answer the research question, explaining the applied techniques in detail and showing the results. This research aimed to formulate a suitable optimization procedure to continuously minimize the cycle time of each product assembly such that the makespan of the total assembly procedure is minimized by optimizing the assembly order of each individual product assembly depending on the operator's process times per assembly task, subjected to the product's assembly precedence constraints and resource assignment constraints. This chapter provides the conclusions and recommendations obtained during this research in Section 7.1 and Section 7.2 respectively.

7.1 Conclusions

A set of makespan minimization tools has been provided, all searching for a suitable assembly order based on the expected process times computed by applying a weighted moving average (EWMA). First, a mixed integer linear programming (MILP) optimization has been proposed. Solving the optimization results in the optimal assembly order corresponding to the input process times, resulting in a decreased makespan compared to applying only one assembly order. However, the applicability of online optimization is limited by the computation times, depending on the products' complexity. Moreover, the accuracy of the expected process times is dependent on the process time variances. Therefore, sub optimal assembly orders are found by the optimization, decreasing the effectiveness of adjusting the assembly order to minimize the makespan.

Three supervised classification algorithms have been proposed to find a suitable assembly order based on the expected process times: the Euclidean distance algorithm, the modal closest algorithm, and the K-nearest neighbors algorithm. The first two algorithms aim to recognize the working operator by computing the distance between the expected process times and the mean process times corresponding to the known operators. The latter aims to recognize the working operator by computing the distances between the expected process times and the process times of its K nearest neighbors in the data set of measured process times, analyzing those distances and examining the labeled operators of the neighbors using a distance-weighted function.

Simulations have proved the effectiveness of the supervised classification algorithms opposed to online optimization. The supervised classification algorithms apply the same amount of assembly orders as known operators opposed to a vast amount of possible assembly orders of the online optimization. Therefore, the supervised classification algorithms are less sensitive to the expected process time computation accuracy. The K-nearest neighbors algorithm is concluded to be the recommended algorithm over the Euclidean distance and modal closest algorithms as it in general outperforms the latter slightly. At increased process time variances, the performance difference between the modal closest algorithm and the Euclidean distance and K-nearest neighbors algorithms increases.

The K-means clustering algorithm is applicable to apply for recognizing new operators. It can be

difficult to parameterize the clustering algorithm because of the sensitivity to parameters clarifying when a optimal assembly order is labeled as a new operator. Nevertheless, K-means clustering does recognize new operators even when the parameters are not perfectly optimized. The results of the K-means clustering tend to improve over time, as more data can be used to characterize the clusters more accurately.

7.2 Recommendations

The possible performance gains by alternating the assembly order for each assembly depending on the operator's process times are discussed in this report. It can be concluded that the attainable performance gains are small. Though, this research could be used as stepping stone for future research in this topic. For example, it might be very beneficial to start assembly on the next product before the assembly on the current product is finished instead of assembling products in sequence. This can be applied to remove idle time of the operator and the robot but requires practical tests or an extension of the discrete event model to simulate. Additionally, assemblies for which the utilization of the robot is much higher than the utilization of the operator could be done by using multiple robots. This increases the complexity of the optimization, thus increases the computation time, but also it provides opportunities for additional performance gains. These topics remain to be open for future research.

Bibliography

[1] N. Singh, "A matheuristic for scheduling automated guided vehicles in a multi-tenant settings," series master theses operations management and logistics, Eindhoven University of Technology, Eindhoven, The Netherlands, August 2019. Department of Industrial Engineering & Innovation Sciences, Operations Planning Accounting & Control.

[2] I. Witte, E. Frank, M. Hall, and C. Pal, *Data Mining: Practical Machine Learning Tools and Techniques, Fourth Edition*. 50 Hampshire Street, 5th Floor, Cambridge, MA, 02139, United States: Morgan Kaufmann, Elsevier Science and Technology, 4th ed., 2016.

[3] P. McCorduck and C. Cfe, *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. CRC Press, 2004.

[4] Y.-Y. Chen, Y.-H. Lin, C.-C. Kung, M.-H. Chung, and I.-H. Yen, "Design and implementation of cloud analytics-assisted smart power meters considering advanced artificial intelligence as edge analytics in demand-side management for smart homes," *Sensors (Basel).*, vol. 19, May 2019.

[5] C. Bishop, *Pattern Recognition and Machine Learning*. Information Science and Statistics, Springer, 2006.

[6] W. Hopp and M. Spearman, *Factory Physics*. Irwin/McGraw-Hill series in operations and decision sciences, McGraw-Hill, 3 ed., 2008.

[7] M. Sugiyama, ed., *Introduction to Statistical Machine Learning*. Boston: Morgan Kaufmann, 2016.

[8] V. Nasteski, "An overview of the supervised machine learning methods," *HORIZONS.B*, vol. 4, pp. 51–62, 12 2017.

[9] T. S. Madhulatha, "An overview on clustering methods," *IOSR Journal of Engineering*, vol. 2, pp. 719–725, April 2012.

[10] B. A. McCarl and T. H. Spreen, *Applied Mathematical Programming Using Algebraic Systems*. 2003. Texas A&M University, Department of Agricultural Economics, http://http://agecon2.tamu.edu/people/faculty/mccarl-bruce/books.htm.

[11] M. B. Perry, "The exponentially weighted moving average," *Wiley Encyclopedia of Operations Research and Management Science*, February 2011. The University of Alabama, Department of Information Systems, Statistics and Management Science.

[12] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *Internation Journal of Forecasting*, vol. 20, pp. 5–10, 2004.

[13] P. Zarchan and H. Musoff, *Fundamentals of Kalman Filterering: A Practical Approach*. American Institute of Aeronautics and Astronautics, 2000.

[14] R. E. Kalman and R. S. Bucy, "New Results in Linear Filtering and Prediction Theory," *Journal of Basic Engineering*, vol. 83, pp. 95–108, 03 1961.

[15] A. Harvey, *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, 1990.

[16] J. Dunik, M. Simandl, and O. Straka, "Methods for estimating state and measurement noise covariance matrices: Aspects and comparison," *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 15, 01 2009.

[17] A. Laub, "A schur method for solving algebraic riccati equations," *IEEE Transactions on Automatic Control*, vol. 24, pp. 913–921, December 1979.

[18] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, pp. 325–327, April 1976.

[19] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-event system simulation; 5th ed.* Upper Saddle River, NJ: Pearson, 2010.

[20] I. Adan, A. Hofkamp, and J. Rooda, "Chi 3 tutorial." Eindhoven University of Technology, Department of Industrial Engineering & Innovation Sciences, Operations Planning Accounting & Control, October 8 2012.

[21] J. P. van den Berk, "Analysis of the intel five-machine six step mini-fab," Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004. Department of Mechanical Engineering, Systems Engineering Group, SE 420383.

Technische Universiteit Eindhoven University of Technology

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

05/02/2020

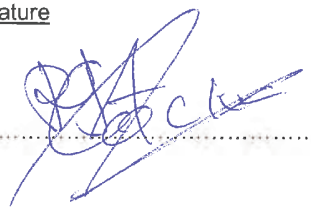
Name

R. J. A. Cocks

ID-number

0971042

Signature



Submit the signed declaration to the student administration of your department.

[i] See: http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/
The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.
More information about scientific integrity is published on the websites of TU/e and VSNU