CHALMERS

Implementation of a Framework for Restart after Unforeseen Errors in Manufacturing Systems

Master's Thesis in Systems, Control and Mechatronics

SHILAN PARSAEIAN

Department of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2014
Master Thesis EX051/2014

Abstract

Error recovery in manufacturing systems, which can be divided into error detection, fault diagnosis, error correction and restart, is a time-consuming and complicated procedure. This work focuses on the restart phase and aims to test a new restart method to answer whether this theoretical method can help operators and speed up the error recovery procedure. The method restarts the system from an operator desired *restart state* and re-synchronizes the manufacturing system with the control system, whereby the nominal production can be continued. Moreover, during the recovery procedure, an added HMI interface supported the operator actions. Additionally, implementation of this method does not impose any additional hardware on the manufacturing system.

These finding shows that the method is applicable in practice. Using this method, there is no need to start the corrupted manufacturing procedure from the beginning, thus, restart using this method saves time and money. Moreover, this method can restart the system from unforeseen errors and it is easy to perform. Having a complete knowledge of a manufacturing system, this method with small modifications in the corresponding control system, can be implemented for the manufacturing system.

Acknowledgements

I would like to express my special appreciation and thanks to my supervisor Professor Dr. Martin Fabian, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a researcher. Your support on the research way has been priceless. I would also like to thank my advisor, Patrik Bergagård for the useful comments, remarks and engagement through the learning process of this master thesis, and for serving as my committee members even at hardship. I would especially like to thank P.S.L. lab supervisors, Per Nyqvist and Hans Sjöberg. You both have been there to support me when I was implementing the program and collecting data for my master thesis.

A special thanks to my family. Words cannot express how grateful I am to my mother Shahla, and my father Ali for all of the sacrifices that you've made on my behalf. Your prayer for me was what sustained me thus far. I would also like to thank all of my friends, Amir Shehni, Cristian Bogdan Czegledi, and Sarmad Riazi, who supported me in writing, and incited me to strive towards my goal. At the end I would like express appreciation to my beloved husband Afshin who spent sleepless nights with and was always my support in the moments when there was no one to answer my queries.

Shilan Parsaeian, Gothenburg 2014/4/27

Contents

1 Introduction **1**

2 Background **5**
- 2.1 A sequence of operation . . . 7
- 2.2 Restart framework . . . 8
  - 2.2.1 Restart state calculation . . . 10

3 Implementation **13**
- 3.1 General implementation . . . 13
  - 3.1.1 A single operation implementation . . . 14
- 3.2 Adding restart facilities . . . 17

4 Conclusion **19**

5 Future developments **20**

Bibliography **22**

A Case study **23**
- A.1 Experimental example description . . . 23
- A.2 Case study SOP . . . 25
- A.3 Implementation of an operation . . . 29
  - A.3.1 Robot program . . . 30
  - A.3.2 State machine . . . 33
  - A.3.3 Conditions and action . . . 35
  - A.3.4 Busy . . . 36
  - A.3.5 Enable . . . 38
- A.4 Restart from an operation . . . 40

List of Figures

2.1 The state concept for a single operation 5
2.2 The control system states against the physical system states 6
2.3 An error p_e is a state of the physical system that is unexpected from the control system point of view at that moment 7
2.4 An SOP example consists of four operations 8
2.5 An operation state diagram. Pre-condition and post-condition are abbreviated as Pre and Post. i, e, and c are the state suffixes for an initial, an executing, and a finished state, respectively. 8
2.6 The control system state diagram of SOP presented in Figure 2.4 9
2.7 The bottom plane represents the nominal production before the restart, while the upper plane represents the execution after restart. q_rs is the restart state, q_i is the initial state, q_c is the final state where the nominal production is done, and q_e and p_e are the error states of the control system and the physical system, respectively. . 10
2.8 All possible placement events assuming there is no placement constraints 11
2.9 The specification automation when the operations A and B can be executed and restarted in any order. 12
2.10 The specification automation when the operation B is conditionally restartable. . . . 12

3.1 A simple platform for theoretical implementation 14
3.2 The state transition diagram of O_1 14
3.3 The operation block inputs are pre- and post- conditions and its output is a command. This command triggers pre-action in the physical system. 15
3.4 An operation state diagram 15
3.5 An operation state is a function of pre-condition, post-condition and its previous state 15
3.6 If an operation is in **executing** state, its pre-action is triggered by sending a command. 15
3.7 A single operation block while pre-condition and post-condition are also derived. . . 16
3.8 A complete operation block diagram. 16
3.9 Designed HMI screen with opened Restart panel 17
3.10 Instruction menu helps the operator to satisfy the nominal and the re-execution requirements. 17

A.1 The manufacturing system installed in the PSL lab at Chalmers University of Technology 24
A.2 The fixture, where model car is assembled 25
A.3 The final product is just made of the base and the roof plates. 26
A.4 SOP of the case study 27
A.5 R_1's safe positions are shown by solid red circles, and R_2's safe positions are shown by solid black circles. 29
A.6 The operation **R1_PlaceBottem** implementation in FBD language 31
A.7 The control and physical systems connection during operation execution 32
A.8 The manufacturing system consists of the control system and the physical system . . 32
A.9 The main module of the robot program 34
A.10 **GOR_UPPDRAG_Shilan** which is only a switch case of **PrgNr** 34
A.11 **R2WagToFixture** is a robot program 34
A.12 The state machine block diagram in FBD language 35
A.13 Inside view of the state machine. The state is updated according to the last state value, pre- and post-conditions. 36
A.14 The status of the control and the physical system before and after executing the **R1_PlaceBottom** operation 37
A.15 The operation **R1_PlaceBottom**'s pre-condition 37
A.16 The operation **R1_PlaceBottom**'s post-condition 37
A.17 The control system sends a relevant command during operation execution 37
A.18 The Robot busy status 38
A.19 The control system stops sending command if either an error occurs or the robot is busy. 39
A.20 The operation **R1_PlaceBottom** implemented using the port **EN** 39
A.21 Error detection mechanism 40
A.22 Alarm is triggered either when the operator clicks on the stop icon, or when **TimeErr** occurs. 40
A.23 The restart block assigns the operation state according to the selected restart state. 41
A.24 LiftUpBottom is selected as the restart state. 41
A.25 The dynamic menu and the static menu when **LiftUpBottom** was selected as a restart state. 42
A.26 Assesment of the correct position for restart. 43

List of Tables

1.1 Different restart methods and their drawbacks. 3

2.1 Possible restart states for different potential error states 11

A.1 List of operations executed during the nominal production of the case study 28
A.2 Handshaking and safe position 30
A.3 Parameters changes during operation execution 33
A.4 State values and the numbers allocated to them. 35

1

Introduction

Imagine you started your washing machine one hour ago, and the laundry is now 80% done. Suddenly, the electricity goes out, and the machine stops. When the electricity comes back, the machine starts the whole procedure from the beginning. This means that the machine must repeat 80% of the whole job. Consequently, 80% of the whole electricity and the whole time used in a normal laundry on the first try is wasted.

The electricity interrupt in the washing machine is an example of an *error* occurrence which is defined as a difference between what was expected (e.g. normal laundry) and what really happened (e.g. electricity interrupt). This interrupt is an example of errors that occur in factories, where an error occurrence may force operators to unload the whole manufacturing system. In addition, a manufacturing system, which is composed of a variety of different machines with different manufacturing tasks that are run by different operators, is more complicated and difficult to handle than a laundry machine. Consequently, error occurrence in such a system for lack of operator support makes manufacturing even more complicated and time consuming.

Some system designers provide manufacturing systems with mechanisms which can detect and correct special errors. For those manufacturing systems, if some unforeseen errors happen during manufacturing, there is no other way than to start from the beginning. Moreover, even if an operator does not have to start manufacturing from the beginning, there is typically no systematic aid to help him/her to resume the manufacturing procedure.

Most error related stoppages in a manufacturing system are due to hardware [1], such as tool breakage. Replacement of broken tools or defective products unloading from a manufacturing system may require operator interference. This interference may lead an operator to change the physical positions of machines in order to get access to the broken tool and perform repair. In fact, after these manual changes, it is very unlikely that the actual machine position becomes equal to the expected position [2]. Thus, even if the operator repairs the error, normal production cannot be immediately resumed.

Error recovery as defined in [3] is the process by which the system is recovered from an abnormal condition so that manufacturing can continue. Similar to [4], this process can be divided into four major activities: *detection* of any difference between what is expected and what actually happened;

diagnosis of the original problem; *correction* of the original problem and removing its effects; and *restart* of the manufacturing, which is a process to start the normal production [1]. In this work, it is assumed that the relevant mechanisms for error detection and diagnosis exist and are implemented in the *controller* part of the manufacturing system which controls resources of the manufacturing system. Also, the operator is supposed to be responsible for performing the error correction phase. Thus, the focus of this work is on the restart phase.

In [5] and [1], error recovery methods are partitioned into two major categories: off-line and on-line methods. Off-line methods use some mechanisms (hardware or software) embedded in the manufacturing system before starting manufacturing. And then during manufacturing, when an error occurs, without operator interference, these mechanisms repair the system. The term *off-line* comes from the fact that these mechanisms are designed and implemented off-line. These mechanisms may provide the manufacturing system with redundant tools or redundant programs, such that if an error occurs during application of a program or a tool, the redundant program/tool will replace the defective program/tool [6]. Obviously, these methods need full information about the system. On-line methods are methods that decide how to correct the error after error detection [7]. Since the on-line methods analyse error during manufacturing and then produce a sequence of actions to correct the error, these methods require more powerful controller.

Another classification is presented in [4] and [5]. According to this classification, error recovery techniques are divided into three methods:

1. Programming language construct
2. Knowledge based
3. Graph based

Programming language constructs mostly deal with exception handling methods, which are augmented programs to transfer the control of the program to the proper exception handler after error occurrence [4]. In [8], an investigation of exception handling in several programming languages and their application in robot programming is presented. This type of error handling mostly leads to huge software overhead to detect and correct a limited set of expected errors.

Knowledge based methods are typically on-line, or a mix of off-line and on-line approaches. In [7] a knowledge based method is presented in which all of the calculations are implemented in on-line mode. When an error occurs, several tests are run to investigate the original reason of error occurrence. These tests result in a rule table including faulty manufacturing system behavior; the error's original reason, the fault; and how to recover the manufacturing system from them. Initially, there is no look-up table and the set of rules are gradually created during deployment and maintenance.

In [9] and [10], knowledge based mixed methods are presented in which a table of possible error causes and effects is prepared off-line and then after an error occurrence in on-line mode, the control system looks up in the table to find the reason and the most reasonable way to recover the manufacturing system from the error. Thus, this is an off-line planning and an on-line investigation. Since, these error recovery methods are defined depending on the error type, if there is dependency between errors (e.g. error **A** and **B** are cause and effect of each other), diagnosis of the original error would be difficult. Also, knowledge based methods need large amounts of memory capacity of memory to store the information, something that prohibits their inclusion in the typical shop-floor control system.

Another knowledge based mixed method is illustrated in [11] to recover a dancer robot from an error that occurs during ballroom dance in which a human male dancer leads a female robot dancer. Based on interactive force/movement applied by the human and the pre-defined rules of dancer movement, the robot estimates her partner's next movement. If the robot makes a mistake, it detects the error while her partner is doing the next movement. Consequently, the robot can according to the human leg's motion correct its estimation and then continue dancing correctly.

Graph based methods are mostly off-line methods that provide the manufacturing system with some alternative production paths [12] to produce the same product, and then when an error has occurred in one production path, the graph based method recovers the manufacturing system by switching the production path to the alternative path. Examples of these methods are backward error recovery [8], which restarts the manufacturing system from earlier activities and then repeats activities where the error happened (if possible); or rescheduling [13], which provides (with on-line or off-line planning) a sequence of operations to return the manufacturing system to the nominal production [10], [5]. The inclusion of the necessary information typically leads to a huge system. Table 1.1 summarizes the methods.

Table 1.1: Different restart methods and their drawbacks.

Method	Tools	Drawbacks
Programming language construct	Exception handler	Extra code
Knowledge based system	Rule	Assume foreseen errors
	Reasoning	Needs large resources
	Separated from the program	Needs full information about the application
Graph based approaches	Petri net	Needs full information about
	Automata	faulty and non-faulty behavior

The method implemented in this work [14] is a graph based method that combines off-line and on-line methods to reach a better performance. This method was presented in [1] and was then extended with implementation details in [15]. This work aims to verify that the theoretical idea presented in [14] can help operators and speed up recovery process.

After error correction, [14] restarts the system from a desired operation, a *restart state*. This method guarantees that it synchronizes the manufacturing system with the control system. Compared to the presented methods in the literature, to use the method [14] very few changes are required to a manufacturing system. Moreover, there is no redundancy or restart path added to the manufacturing system, and consequently, complexity is reduced in relation to previous works. Also, the method is not designed based on the error type, but rather on the selected restart state, so the control system can be recovered from unforeseen errors by means of this method.

The idea presented in [14] is more flexible than the idea of [15], and can be applied to almost every industrial manufacturing system. Moreover, [14] limits the restart procedure to the defected machines, so the operator cannot decide about the other machines state during restart procedure.

The rest of this report is organized as follows: In Chapter 2 a review of preliminaries and the main idea of [14] are presented. A theoretical implementation of the idea using block diagram language is described in Chapter 3. Finally, in Appendix A, the real implementation of the work is discussed in detail.

2

Background

A *manufacturing system* is composed of a *control system* and a *physical system.* The physical system is the set of all machines, tools and products controlled and supervised by the control system [16]. *Process planning* is an important activity to develop a new factory, during which tasks that should be executed to produce the intended product by the manufacturing system are planned. These tasks are modelled by *operations* [16].

Each operation of the control system is modelled by an automaton [16] with three *states.* The three operation states respectively denote that the operation is either *initial* (not started), *executing*, or *finished.* Figure 2.1 shows the state concept for a single operation, in which $O_K^{\uparrow}$ is an event to trigger execution and $O_K^{\downarrow}$ is the completion event.

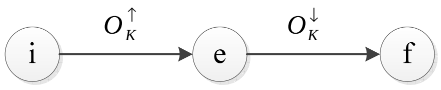


Figure 2.1: The state concept for a single operation

A set of conditions which must be true just prior to operation execution is called a *pre-condition.* Thus, a pre-condition is a set of conditions that must be satisfied to change the operation state from **i** to **e** ($O_K^{\uparrow}$ in Figure 2.1). Tools limitation, conditions on operation execution order, and sensor values are examples of pre-conditions.

Every operation execution changes some parameters in the physical system. Parameters in the manufacturing system that are changed by an operation execution form the *post-condition* of the operation. In other words, a post-condition is a set of conditions that must be satisfied before changing the operation state from **executing** to **finished** (the completion event, $O_K^{\downarrow}$, in Figure 2.1). Signals sent by the manufacturing system to announce that the operation execution in the physical system has finished are integral parts of the post-condition.

Pre-condition satisfaction assures the control system that the physical system is ready to start the execution of the specific operation whose pre-condition is satisfied. The control system can then send a command to the manufacturing system to start the execution. A set of actions executed in the physical system, as a result of sending a command, is called a *pre-action*. Running an executing code in the physical system, sending and receiving data are pre-action examples.

A *production plan* can be viewed as a trajectory between a start point (q_i in Figure 2.2) where none of the operations have been started to execute, to a final point (q_c in Figure 2.2) where a sub-set of the operations have been executed. Sometimes this is called a *Sequence of OPeration* or an *SOP*. Operations are sorted by an SOP according to the physical system facilities. For example, if two operations **A** and **B** can be executed at the same time, the SOP includes this possibility. But the control system typically starts one operation each time. Thus, what we plan is that the operation **A** is started first and then the operation **B**. A *nominal production* is a production according to how we plan to produce the product.

The control system is also modelled by an automaton composed of states, which are called *control system states*. Each control system state expresses that some operations have not been started yet (they are in their *initial state*), some operations are executing (they are in their *executing state*), and the rest of the operations have been finished (they are in their *finished state*) [16]. For example, in the control system trajectory, the start point (q_i in Figure 2.2) is a state where no operations have started yet and the final point is a state where a sub-set of the operations have finished (q_c in Figure 2.2) and no operations are executing.

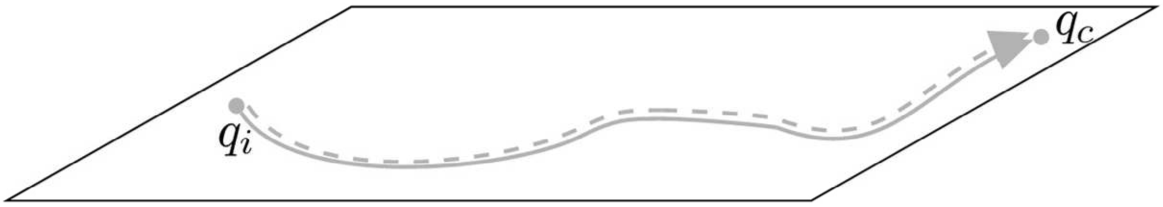


Figure 2.2: The control system states against the physical system states

Also, every factory machine is in a specific state in every moment; the combination of all machines' current state make up the *manufacturing system state*. In nominal production, the physical system state always matches the control system state. The control system states and the physical system states are shown by solid line and hatched line in Figure 2.2. In this way, an error during the nominal production is a state of the physical system that is unexpected from the control system point of view at that moment. This incorrect physical system state is called an *error state*. Such a state represents that the physical system did not continue according to its control system state trajectory but deviated from the nominal production trajectory. In Figure 2.3, the error state is depicted by p_e.

When the monitoring system (the operator and/or some sort of sensors installed in the factory) detects the error, the manufacturing system must be recovered to resume the nominal production. Error recovery which aims to return the physical system to the nominal production is composed of four steps [3]:

1. Error detection

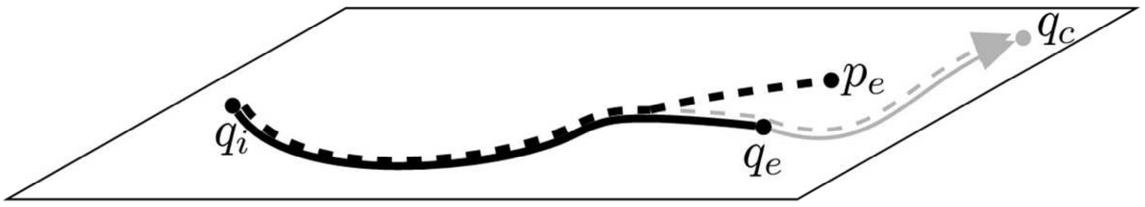


Figure 2.3: An error p_e is a state of the physical system that is unexpected from the control system point of view at that moment

2. Error diagnosis
3. Error correction
4. Restart

This project supposes that there are modules embedded in the control system which detect, and diagnose the error, and that the operator corrects the error before starting the restart phase. Thus, the main emphasis in this work is on the restart module. In the following section, the SOP and the restart state calculation are reviewed.

2.1 A sequence of operation

Each product design addresses its application. For example, a tire is round to speed up a car. This means that, it is unlikely to reach the intended speed with square tires. Thus, the product application introduces several *design constraints*. From the manufacturing point of view, there are also several constraints to be satisfied [17]. Tools or resources are crucial elements for determining these *manufacturing constraints*. As an example, a lack of tools to weld connection points limits the number of concurrent welding operations.

Manufacturing constraints and design constraints, collectively called *nominal requirements*, specify how to execute the operations defined during process planning. For instance, if placing part A hides connection point B, firstly a robot must weld the connection point B and then place part A. An SOP is a sequence of operations that are sorted in a special order to satisfy these constraints. Figure 2.4 gives an SOP of 4 operations, O_1, O_2, O_3, and O_4. The black solid circle, inside each operation block, is the post-condition symbol. O_2 and O_3 are executed in *parallel*. This means that they are executed independently, e.g because they are executed by two different machines. Parallel sequences are illustrated by double bars in SOP. Also, these two operations are in *straight sequence* of execution with O_1 and O_4 which specifies that O_1 must be started at first, then O_2 and O_3 are executed in parallel, finally O_4 is executed.

As mentioned earlier, every operation, from start until completion, passes three states: **initial**, **executing**, and **finished**. Figure 2.5, which describes the behaviour of an operation, is the *state diagram* of the operation. O_k is an operation and i, e, and c are the state suffix for an **initial**, an **executing**, and a **finished** state, respectively. Also, *Pre* and *Post* are abbreviations of pre-condition and post-condition respectively.

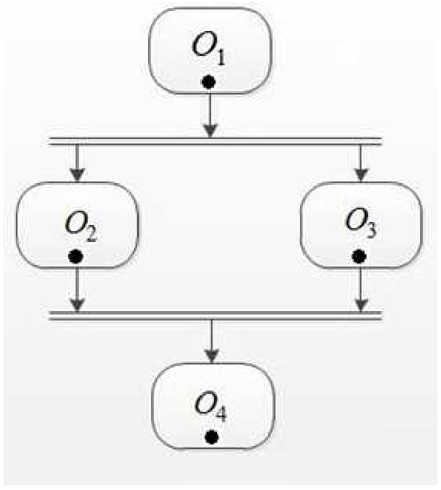


Figure 2.4: An SOP example consists of four operations

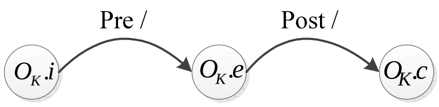


Figure 2.5: An operation state diagram. Pre-condition and post-condition are abbreviated as Pre and Post. i, e, and c are the state suffixes for an initial, an executing, and a finished state, respectively.

The state diagram of the control system is obtained from synchronization of the operation state diagrams. Therefore, the state diagram of SOP presented in Figure 2.4 can be given by Figure 2.6.

2.2 Restart framework

When the monitoring system (operators and/or some sort of sensors installed in the factory) detects a difference between the current manufacturing system state and the expected control system state, an alarm will warn of an abnormal situation.

Roughly 60% of all stoppages in factories result from human errors, and about 15% of all stoppages are consequences of tool breakage [18]. Also, most errors are related to the physical system [1]. These sort of errors need an operator to perform repair, which may lead to corrupting the physical system state. The operator changes the physical system state but the control system remains in the same state as when the error occurred. Then the control system has a misconception about the physical system state. To remove the difference, it may be necessary to change both the control system state and the physical system state.

This problem may be solved by definition of a reference table so that the control and the physical system states are changed to a *restart state* [14] selected from a reference list. A *restart state* is a control system state from which the stopped production can continue. In Figure 2.7, this restart state is shown by q_{rs}. It is important to mention that it is not necessary to select restart state from the nominal production states. The only condition is that the last state that we want to reach in the nominal production (the marked state) must be reachable from the restart state. In Figure 2.7

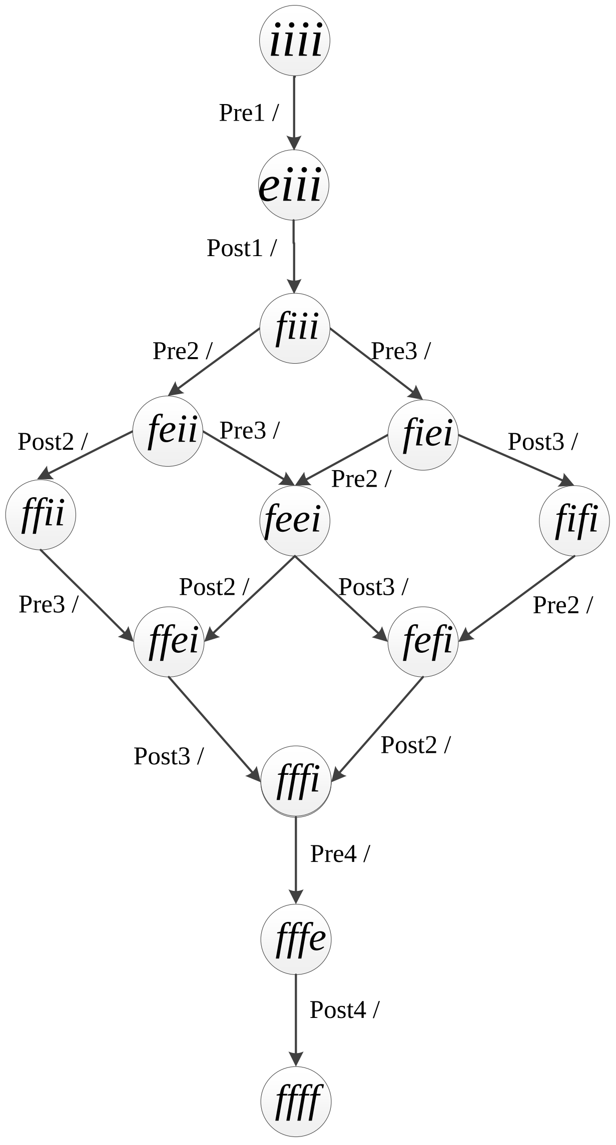


Figure 2.6: The control system state diagram of SOP presented in Figure 2.4

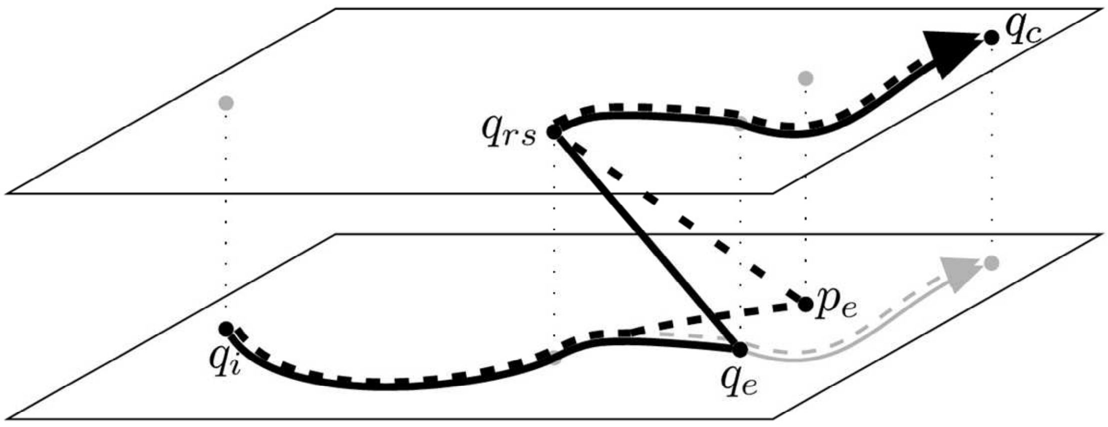


Figure 2.7: The bottom plane represents the nominal production before the restart, while the upper plane represents the execution after restart. q_rs is the restart state, q_i is the initial state, q_c is the final state where the nominal production is done, and q_e and p_e are the error states of the control system and the physical system, respectively.

q_c which is the marked state in this SOP, is reachable from q_{rs}.

The restart states set, according to the nominal requirements and *re-execution requirements* is calculated off-line. Re-execution requirements specify under what circumstances the operation can be restarted. As an example, a gluing or a welding operation typically cannot be re-executed. In this way, the restart states set can vary depending on the corrupted operation, not on the particular error.

2.2.1 Restart state calculation

This section describes how to calculate the proper restart state list for each operation. The restart states list is calculated based on nominal and re-execution requirements. We illustrate this method by a simple example. Also, it is necessary to point out that, in this work we assume that the restart state list is already calculated to implement in the control system. Thus, we do not need to implement any mechanism to calculate these states.

Let A and B be the only operations of the control system. We assume that they are independent, so they are executed in parallel (Figure 2.8). Considering that each operation must pass three states before completion (**initial**, **executing** and **finished**), the corresponding control system has nine (3×3) states.

Errors only occur during the operation execution period, because an error is an incorrect change and the system state only changes during execution. For this reason, **executing** states are said to be *potential error states*, which means that there is a possibility of deviation from the nominal production only in these states. These states are shaded in gray in Figure 2.8. Generally each control system state including at least one operation in **executing** state is a potential error stare. Then, the restart states are only calculated for the potential error states.

The error recovery procedure can change the system control state to either its earlier states (*backward error recovery*) or later states (*forward error recovery*). An event by which a potential

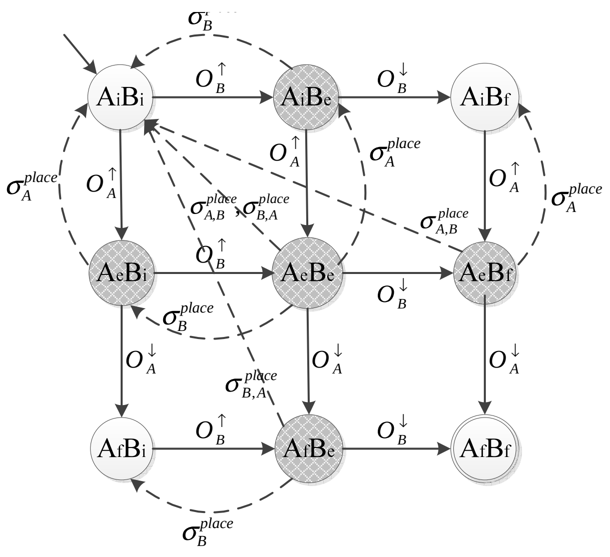


Figure 2.8: All possible placement events assuming there is no placement constraints

error state is transferred to a restart state is called a *placement event* and is represented by $\sigma_{\alpha,\,\beta}^{place}$ in the state diagram. Here, α is an operation corrupted by an error and β is a set of operations that may be restarted along with α. Figure 2.8 gives all possible placement events assuming there is no placement constraint.

Table 2.1: Possible restart states for different potential error states

Corrupted operation	Potential error operation	Restart state
A	A_eB_i	A_iB_i
	A_eB_e	$A_iB_i,\ A_iB_e$
	A_eB_f	$A_iB_f,\ A_iB_i$
B	A_iB_e	A_iB_i
	A_eB_e	$A_iB_i,\ A_eB_i$
	A_fB_e	$A_fB_i,\ A_iB_i$
A and B	A_eB_e	A_iB_i

The main criteria for selection of an appropriate restart state are nominal and re-execution requirements. If there is a constraint on restarting B in Figure 2.8, this limits the number of possible restart states. Table 2.1 shows possible restart states (when there are no constraints) for different potential error states. As an example, if there is a requirement that B can only be restarted when A is restarted before, B is *conditionally restart-able*, because it is possible to have no restart state. Figure 2.9 illustrates a specification automaton when A and B can be executed and restarted in any order.

Figure 2.10 shows the specification automaton when B is conditionally restartable. This means

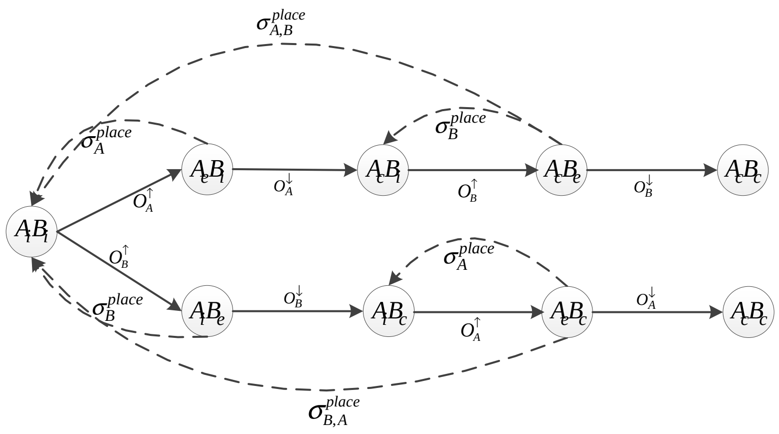


Figure 2.9: The specification automation when the operations A and B can be executed and restarted in any order.

that, they can be executed in any order but restarted in a special order, first A and then B. As an example, the restart state A_iB_i for the potential error state A_iB_e is forbidden, because A has not been restarted or even executed yet. There are two possible restart scenario depending on which operation is started first.

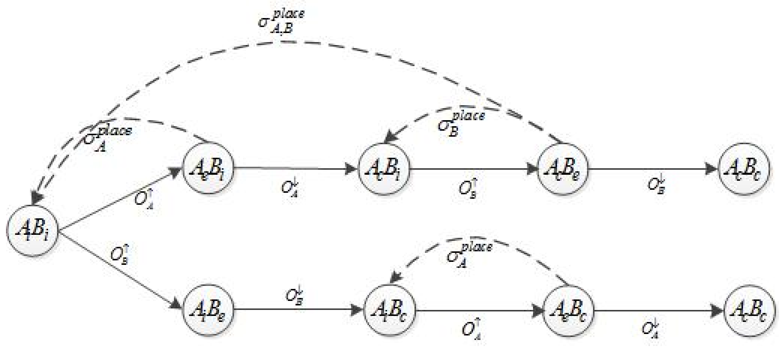


Figure 2.10: The specification automation when the operation B is conditionally restartable.

If the operation A is executed first, the control system state is changed from A_iB_i to A_eB_i in which A can be restarted because there is no constraint on A's restart. When B is executed and the control system state is changed from A_cB_i to A_cB_e in which if A was restarted before, B can be restarted.

There is another scenario in which B is started first and can never be restarted. Executing B changes the control system state from A_iB_i to A_iB_e in which there is no possibility of restarting B, because A has not been executed yet. Also, when the operation A is executed and the system control state is changed from A_iB_c to A_eB_c, the operation A can be restarted but there is no way to restart B.

3

Implementation

There are different ways of implementing a unique idea. This variety is due to variation in programming tools and programmers writing style. Generally, depending on programmer preferences, programming can be performed in different ways which all lead to the same result. The way suggested here to theoretically implement the idea presented in [14] is a general way and easy to follow.

The implementation is represented by block diagram to prepare the reader for the actual implementation, which is based on P.L.C. (Programmable Logic Controller) programming and is presented in Appendix A. Thus, in this chapter, regardless of the implementation details, we discuss the suggested way from the general point of view and implementation details are presented in Appendix A.

The implementation theory is presented in two steps, first implementation of the whole work regardless of the restart framework, and then the restart framework which is added to the first part. For ease of explanation, in this chapter, the implementation is presented for an simple control system.

3.1 General implementation

Implementation of a general control system as a platform to implement the restart framework in Section 3.2 is presented in this section. The selected control system is a simple system composed of a **start** command, a **stop** command, an **Alarm**, and a simple SOP, which are shown in Figure 3.1. The **start** and **stop** key starts and stops, respectively, the procedure defined by the SOP, and the **Alarm** light is considered to be a warning of an abnormal situation. The SOP is composed of 4 operations, O_1, O_2, O_3, and O_4. Operations O_2 and O_3 are executed in parallel and they are in straight sequence of execution with O_1 and O_4, as shown in Figure 3.1.

The state diagram of this SOP has already been presented in Figure 2.6. Figure 3.2 gives part of this state diagram which shows the state transition from the **initial** state of the operation O_1 to its **finished** state. Once an operation pre-condition is satisfied, the operation state changes from **initial** to **executing**. At the beginning of execution, the control system sends a command

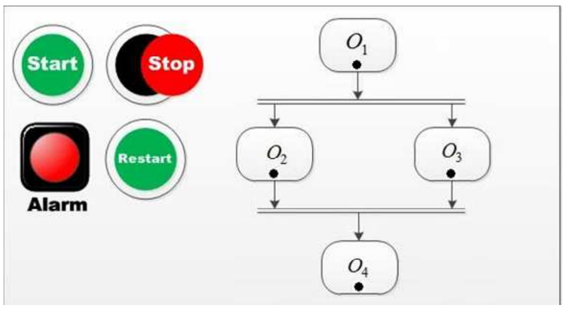


Figure 3.1: A simple platform for theoretical implementation

to start the corresponding pre-action in the manufacturing system. Post-condition satisfaction terminates the operation execution that precedes changing the state from **executing** to **finished**. This procedure is shown in Figure 3.2.

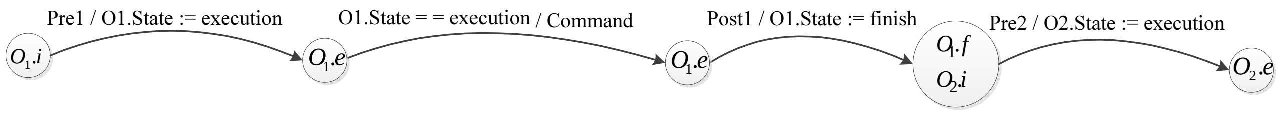

Pre1 / O1.State := execution
O1.State = = execution / Command
Post1 / O1.State := finish
Pre2 / O2.State := execution
$O_1.i$
$O_1.e$
$O_1.e$
$O_1.f$
$O_2.i$
$O_2.e$

Figure 3.2: The state transition diagram of O_1

The main part of this section is dedicated to implementation of this state diagram, which is performed in two steps: first a sample operation is implemented and then, operations, which are all implemented in the same way, are conditioned to execute according to the SOP presented in Figure 3.1.

3.1.1 A single operation implementation

Each operation in the control system has some specifications that vary from operation to operation. The operation specifications are its pre-condition, post-condition, state, command, and pre-action. If an operation is implemented by a block diagram, an operation block, the block should handle all these specifications. As mentioned earlier, an operation state is affected by its pre- and post-condition satisfaction, so their satisfaction is defined as inputs to the operation block (Figure 3.3). The control system changes the operation state based on these two conditions.

Since we are going to implement a single operation in this section, Figure 3.4 emphasizes O_1's state diagram compared with Figure 3.2. Obviously, implementation of a **state machine** is the first requirement to implement this state diagram. The state machine is a part of the control system that is responsible for the calculation of the operation state. As the nominal production starts, this system sets all the operation states to the **initial** state. Once the manufacturing system meets an operation pre-condition, the state machine changes the operation state from the **initial** state to the **executing** state. In the **executing** state, as the post-condition is satisfied, the operation state changes to the **finished** state.

Thus, the next state is a function of the current state, the pre-condition, and the post-condition. Figure 3.5 shows the block diagram for the state machine. The pre-condition and the post-condition

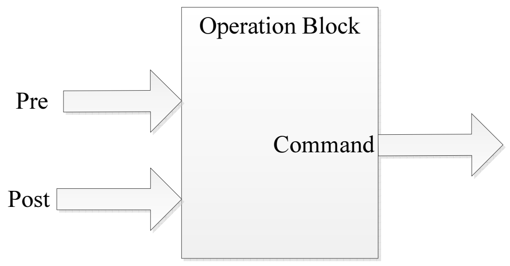


Figure 3.3: The operation block inputs are pre- and post- conditions and its output is a command. This command triggers pre-action in the physical system.

$$\begin{tikzcd}
O_1.i \arrow[r, bend left, "Pre1 / O1.State := execution"] & O_1.e \arrow[r, bend left, "O1.State == execution / Command"] & O_1.e \arrow[r, bend left, "Post1 / O1.State := finish"] & O_1.f
\end{tikzcd}$$

Figure 3.4: An operation state diagram

are inputs to the state machine and state value is input/output for this block.

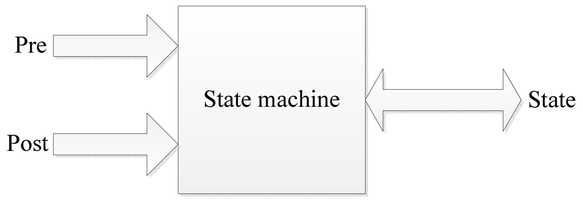


Figure 3.5: An operation state is a function of pre-condition, post-condition and its previous state

Whenever the operation state equals **executing**, the control system should send a command to trigger the corresponding pre-action in the physical system area. Figure 3.6 shows a *command box* that is a block to energize a pre-action depending on the operation state.

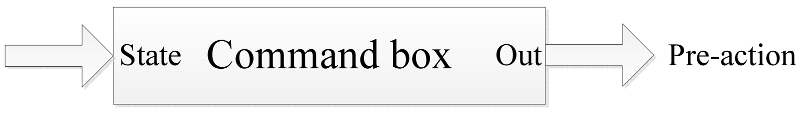


Figure 3.6: If an operation is in **executing** state, its pre-action is triggered by sending a command.

The last step of the operation implementation includes pre-condition and post-condition calculation such that all the constraints imposed by the design and the manufacture are covered.

Pre-condition is composed of conditions which must be satisfied prior to operation execution. As an example, sensor values, tool limitations and the other prerequisites that are necessary to start an operation constitute pre-conditions.

Also, an operation execution results in some changes in the manufacturing system, such as sensor outputs, handshaking signals, etc. Post-conditions are applied to these changes. Figure 3.7 shows the operation block while pre-condition and post-condition are also fed. Finally, the operation block is given in Figure 3.8.

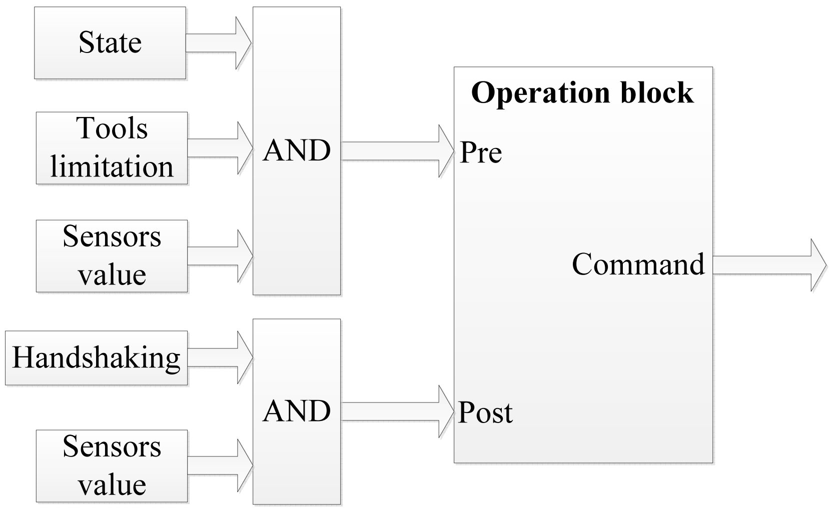


Figure 3.7: A single operation block while pre-condition and post-condition are also derived.

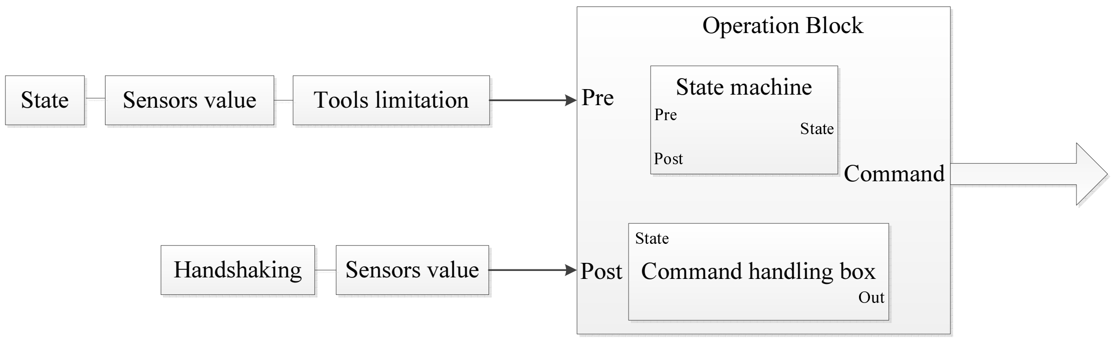


Figure 3.8: A complete operation block diagram.

3.2 Adding restart facilities

When something triggers the alarm, the nominal production is stopped and the recovery procedure, to resume the nominal production, will come to the manufacturing system's aid. Considering that the error state is the current control system state, an HMI panel containing all possible restart states appears (Figure 3.9). This panel includes restart states reachable from the error state, and these states have been calculated off-line. Then, the operator should select a restart state (q_{rs}, Figure 2.7) from the list and then, to show that his/her selection is finalized, click on the **OK** icon.

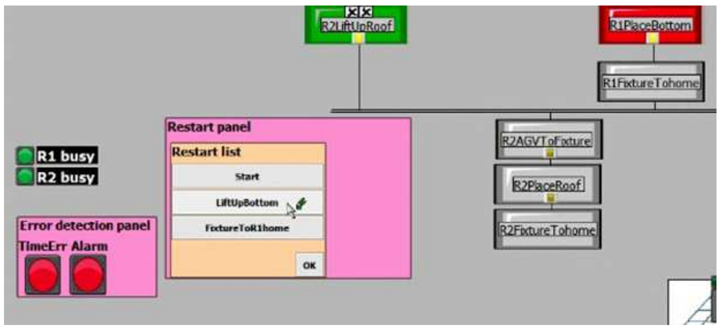
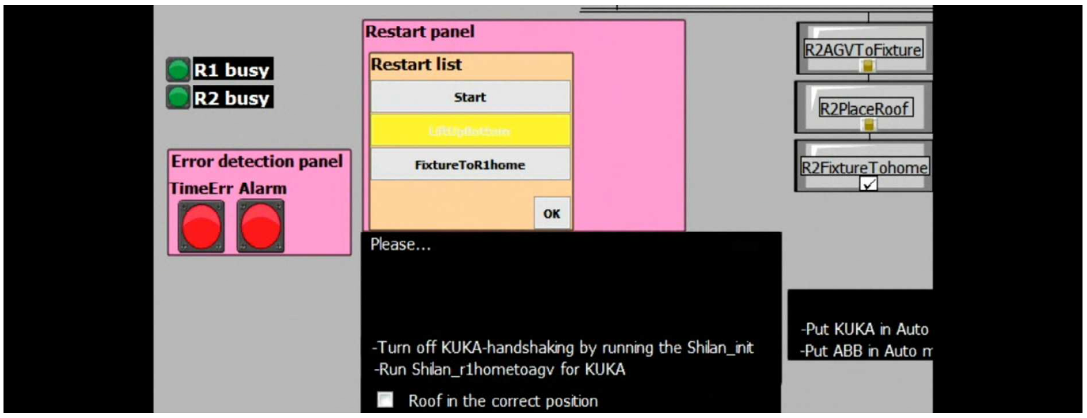


Figure 3.9: Designed HMI screen with opened Restart panel



Please...

-Turn off KUKA-handshaking by running the Shilan_init
-Run Shilan_r1hometoagv for KUKA

☐ Roof in the correct position

-Put KUKA in Auto
-Put ABB in Auto m

Figure 3.10: Instruction menu helps the operator to satisfy the nominal and the re-execution requirements.

Next, another HMI panel containing instructions to help the operator opens. Instructions to move the physical system from p_e to the selected restart state, q_{rs} (Figure 2.7), appears (Figure 3.10). This instruction list have also been calculated off-line and is prepared to satisfy the nominal and the re-execution requirements. By means of these instructions, the operator moves the physical system. This move reresents the dashed line from p_e to q_{rs} (Figure 2.7). When all the instructions have been executed, the physical system is correctly placed in the restart state, otherwise, the operator is not permitted to restart the manufacturing system. In other words, execution of these instructions are considered to be a requirement of the restart phase. Since all of these instructions are either sensor, handshaking, or check-box related, their states are measurable and used as re-execution requirements.

Finally, if all requirements are satisfied, the nominal production can be resumed. When the operator clicks on the **Restart** icon, the selected restart state is loaded into the control system state. This is the last step in the error recovery procedure and the physical system and the control system are now in corresponding states. Once this change is performed, the nominal production is resumed.

4

Conclusion

In this work, implementation of restart idea discussed in **??** was presented. In this implementation, every operation of the nominal production is dedicated a state, a pre-condition, a postcondition, and a command to start the pre-action. An operation is initialized in its **initial** state, and its pre-condition satisfaction changes the operation state from **initial** to **executing**. Being in the **executing** state without error occurrence, the operation block sends a command to start the pre-action in the physical system. Finally, post-condition satisfaction in the **executing** state changes the operation state to the **finish** state.

This project planned to help the operator such that defined states, and menus popping up in each step of the restart procedure finally lead the operator to move the system from the error state to the restart state. The operator selects an intended restart state and then the control system provides him/her with an instruction list which includes instructions to move the physical system to the appropriate safe position for restarting in the intended restart state. After this step, the operator can be assured that the manufacturing system continues with the nominal production by pressing the **Restart** key.

5

Future developments

In this work, it was assumed that the relevant mechanisms for error detection and diagnosis existed and were implemented in the control system. Also, the operator was supposed to be responsible for performing the error correction phase. The focus of this work was just on the remaining part, the restart phase. Even the described restart procedure in this work is a time consuming procedure. The operator stops the whole nominal production and selects a restart state from the restart list. Then, he/she must move the physical system to satisfy conditions mentioned in the instruction list and finally the nominal production is restarted. What if the operator only interferes the defective part of the nominal procedure and correct the error without stopping the whole procedure. In fact, the operator becomes a part of the physical system who can manually correct the error. Thus, the operator can interfere the defective part of the nominal production while the other parts are running. This is a combination of automated and manual nominal productions. In this way, there is no need to stop the whole procedure and there is a collaboration between human and machines.

Bibliography

[1] K. Andersson, B. Lennartson, M. Fabian, Restarting manufacturing systems; restart states and restartability, Automation Science and Engineering, IEEE Transactions on 7 (3) (2010) 486–499.

[2] P. Loborg, A. Törne, Towards error recovery in sequential control applications, in: Proc 6:th int. Symp. on Robotics in Manufacturing, 1996, pp. 377–383.

[3] C. Syan, Y. Mostefai, Status monitoring and error recovery in flexible manufacturing systems, Integrated Manufacturing Systems 6 (4) (1995) 43–48.

[4] P. Loborg, Error recovery in automation—an overview, Presented at the AAAI-94 Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems Stanford, Ca, USA.

[5] E. Adamides, E. Yamalidou, D. Bonvin, A systemic framework for the recovery of flexible production systems, International Journal of Production Research 34 (7) (1996) 1875–1893.

[6] F.-T. Cheng, H.-C. Yang, J.-Y. Lin, Development of holonic information coordination systems with failure-recovery considerations, Automation Science and Engineering, IEEE Transactions on 1 (1) (2004) 58–72.

[7] A. Kazarov, A. Radu, L. Magnoni, G. Miotto, Use of expert system and data analysis technologies in automation of error detection, diagnosis and recovery for atlas trigger-daq control framework, in: Real Time Conference (RT), 2012 18th IEEE-NPSS, 2012, pp. 1–5.

[8] I. J. Coxand, N. Gehani, Exception handling in robotics, IEEE Transactions on Automation Science and Engineering 22 (3) (1989) 43–49.

[9] M. H. Lee, D. P. Barnes, N. W. Hardy, Knowledge based error recovery in industrial robots, in: Proc. 8th. Int. Joint Conf. Artificial Intelligence, 1983, pp. 824–826.

[10] F. Noreils, R. Chatila, Plan execution monitoring and control architecture for mobile robots, Robotics and Automation, IEEE Transactions on 11 (2) (1995) 255–266.

[11] T. Takeda, Y. Hirata, K. Kosuge, Hmm-based error recovery of dance step selection for dance partner robot, in: Robotics and Automation, 2007 IEEE International Conference on, 2007, pp. 1768–1773.

[12] M. Alcaraz-Mejia, E. Lopez-Mellado, A. Ramirez-Trevino, Redundancy based controller reconfiguration for fault recovery of manufacturing systems, in: Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on, 2007, pp. 128–133.

[13] M. Tittus, S.-A. Andreasson, A. Adlemo, J.-E. Frey, Fast restart of manufacturing cells using restart points, in: Proc 4:th World Automation Congress, WAC2000, 2000.

[14] P. Bergagård, M. Fabian, Error recovery for systems modeled as sequences of operations using supervisory control theory, Submitted for possible journal publication.

[15] K. Andersson, B. Lennartson, P. Falkman, M. Fabian, Generation of restart states for manufacturing cell controllers, Control Engineering Practice 19 (9) (2011) 1014 – 1022.

[16] P. Bergagård, Modeling and analysis of restart, transport, and resource allocation in manufacturing systems using sequences of operations, Licentiate thesis, Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden (September 2012).

[17] K. Bengtsson, Flexible design of operation behavior using modeling and visualization, Ph.D. thesis, Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden (September 2012).

[18] T. Sata, H. Hiraoka, M. Miki, S. Takata, Functions required for advanced flexible manufacturing systems, Robotics and Computer Integrated Manufacturing 3 (4) (1987) 417–421.

A

Case study

In this section, the manufacturing system where this project was implemented is introduced. Also, the nominal production in this manufacturing system will be explained and finally, the restart mechanism implementation is discussed. This paper's case study is a manufacturing system to assemble toy cars implemented in the Production System Laboratory (PSL) at Chalmers University of Technology. The corresponding control system is a SIEMENS programmable logic controller, PLC, SIMATIC S7-300. The physical system includes 4 robots and the corresponding robot controllers; a fixture; a wagon; and 4 small toy cars.

A.1 Experimental example description

The example presented here, i.e. this work's case study, is a part of the laboratory car factory implemented in the PSL lab. This selected manufacturing system consists of two robots R_1 and R_2; a wagon, which carries different car parts; a roof plate and a base plate; a fixture; two sensors for the base plate (**Fixture_bak_botten** and **Fixture_fram_botten**); and two sensors for the roof plate (**Fixture_bak_tak** and **Fixture_fram_tak**) (Figures A.1 and A.2). Also, it must be mentioned that the final product is just made of the base and the roof plates (Figure A.3).

Due to safety, two automatic clamp locks are installed on the fixture. Once robot **R1** places the bottom plate on the fixture, the control system locks the clamp locks. When the clamp locks are open, **Fixtur_cylinder_bak_uppe** and **Fixtur_cylinder_fram_uppe**, which are variables connected to the open status of these clamps, are True. Otherwise, if these clamps are locked, **Fixtur_cylinder_bak_nere** and **Fixtur_cylinder_fram_nere** are True and the others (**Fixtur_cylinder_bak_uppe** and **Fixtur_cylinder_fram_uppe**) are False. There is no state in-between for these sensors, because clamps are either opened or closed, thus, there is no case that all four are false.

In order to validate a new mechanism, we must use it. Thus, some events that lead to restarting the nominal production are considered in this implementation. There are two main events that lead to stop and consequently restart of the nominal production: an operator request or an error occurrence.

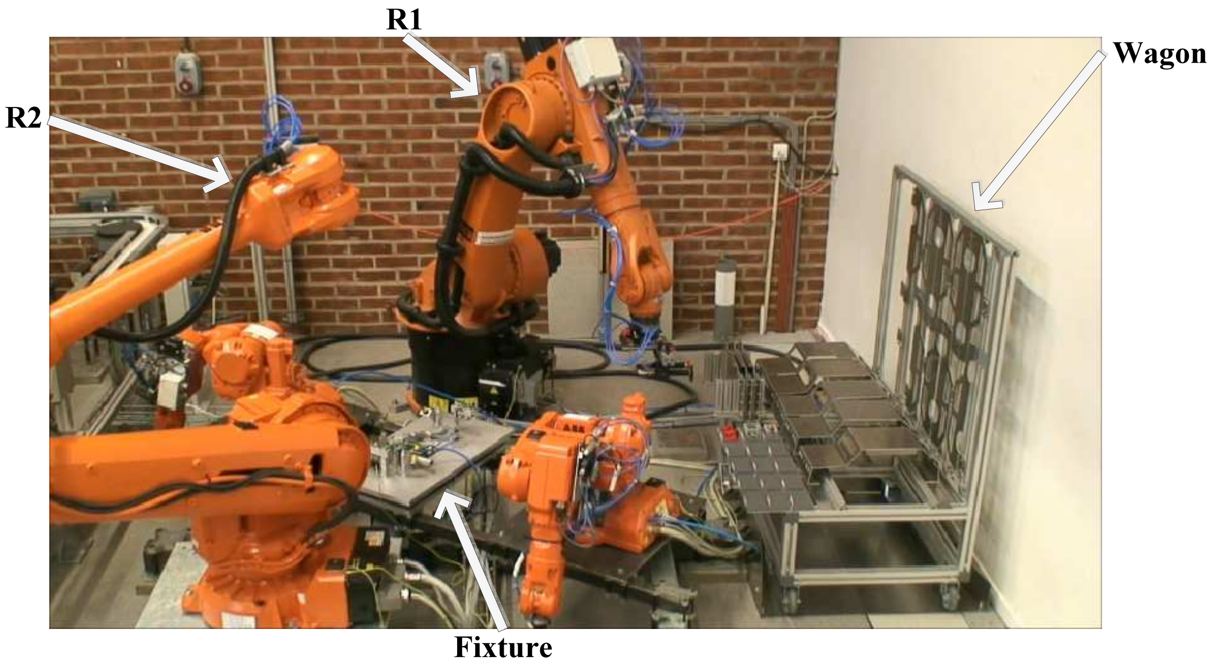


Figure A.1: The manufacturing system installed in the PSL lab at Chalmers University of Technology

Three of the operations are equipped with stop possibility due to operator request. This means that the operator can deliberately stop the procedure during execution of these operations. Also one of these three operations is provided with a simple error detection mechanism. Using this detection mechanism, the control system can stop the corrupted procedure. Thus, there are only two operations equipped with the stop possibility and there is one operation equipped with both the stop possibility and the error detection mechanism. Two errors are considered that corrupt the nominal production:

1. picking no base plate from the wagon (**LiftError**)
   The robot R_1 fails in lifting the base plate up from the AGV.

2. dropping the base plate (**DropError**)
   The robot R_1 drops the base plate on its way from the wagon to the fixture.

these both lead to the same failure in the manufacturing system, i.e. there is no base plate on the fixture. Thus, the embedded error detection mechanism, which is designed to detect these errors, checks the availability of the base plate on the fixture.

Since, the nominal production stops due to either operator request or error detection, all of the operations equipped with theses are considered as potential error states and there is an allo-

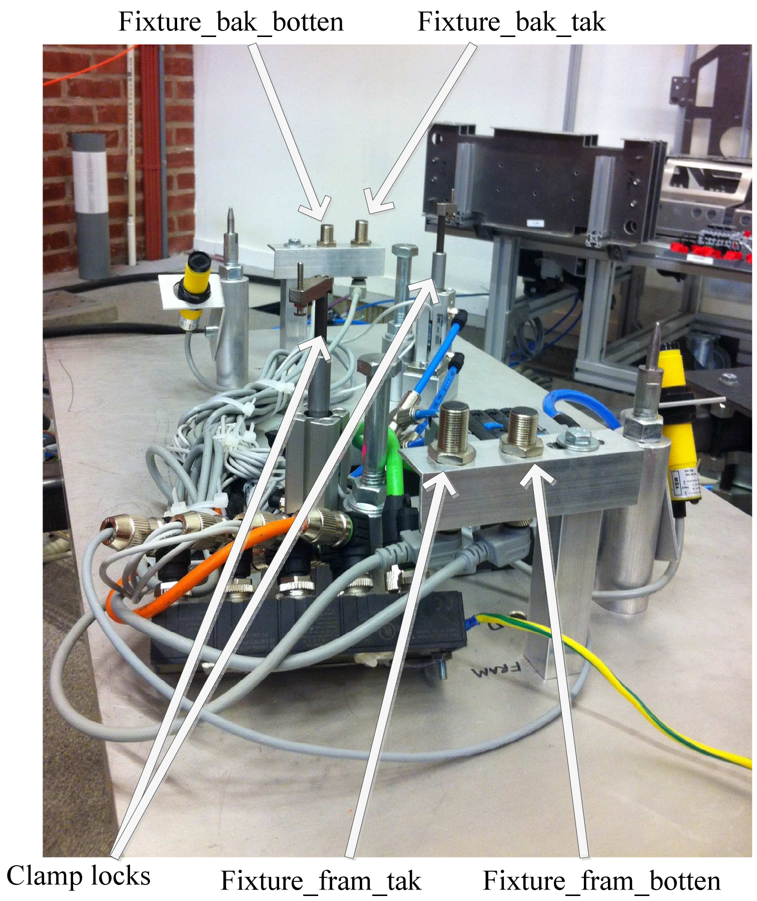


Figure A.2: The fixture, where model car is assembled

cated restart list for each of the three operations equipped with stop possibility or error detection mechanism. As mentioned earlier, a restart list is a collection of restart states from which the manufacturing system has to be restarted after an error occurrence. These restart lists (one list per each of three operations) are calculated off-line.

A.2 Case study SOP

The nominal production is that: First, R_1 picks up the base plate from the wagon. Next, R_2 picks up the roof plate from the wagon and R_1 puts the base plate on the fixture at the same time. Finally, R_2 puts the roof plate on the fixture. A sequence of operations (SOP) that leads to this nominal production is shown in Figure A.4. This SOP is designed such that nominal production requirements and physical system limitations are satisfied. These operations are explained in detail

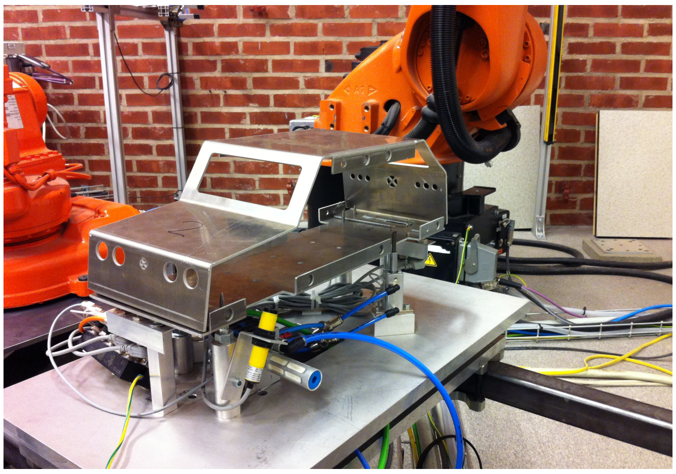


Figure A.3: The final product is just made of the base and the roof plates.

in Table A.1. Solid black circles in Figure A.4 represent the post-conditions which are replaced by yellow lights on the HMI screen.

The first operation in the SOP of Figure A.4 is **R1_HomeToWag**. This is an operation to move R_1 from R1Hemmalage to R1Wag (Figure A.5). R1Hemmalage and R1Wag are called *safe positions*. A safe position is a physical position/location where a robot finishes an operation, and waits for a new command. These positions are safe, because there is no collision risk. These positions should be determined before manufacturing is started.

Since every operation is started and finished at a safe position, Figure A.5 shows that each robot is assigned 4 safe positions. Safe positions for R_1 are located at its home position, R1Hemmalage; top of the wagon position, R1Wag; between the wagon position and the fixture position, R1Inbetween; and the fixture position, R1Fixture. The safe positions of R_2 are located at its home position, R2Hemmalage; top of the fixture position, R2Wag1; between the wagon position and the fixture position, R2Wag2; and the fixture position, R2Fixture. Figure A.5 approximately shows where these safe positions are located. In this figure, R_1's safe positions are shown by solid red circles, and the safe positions for R_2 are shown by solid black circles. Also, the operations to move robots between every two safe positions are mentioned. The double-circled positions show the robot positions when the nominal production starts.

As mentioned, a handshaking signal is a signal raised by the physical system to show that an operation execution has been completed in the physical system area. Every operation has its handshaking signal and these signals are shown by solid black circles in Figure A.4. In this report, operations, handshaking signals, and safe positions have similar names. Then, to avoid confusion, **operations**, *handshaking signals*, and safe positions are denoted by **Bold**, *Italic*, and Normal letters, respectively.

Table A.2 lists handshaking signals and their corresponding operations. The physical system and the control system communicate by handshaking signals. These signals are outputs from the physical system and inputs to the control system. So, in this work, *IN* is selected as the middle name of handshaking signals in the control system and *OUT* is selected as their middle name in the physical system. For example, the handshaking signal to show **R1_HomeToWag** completion

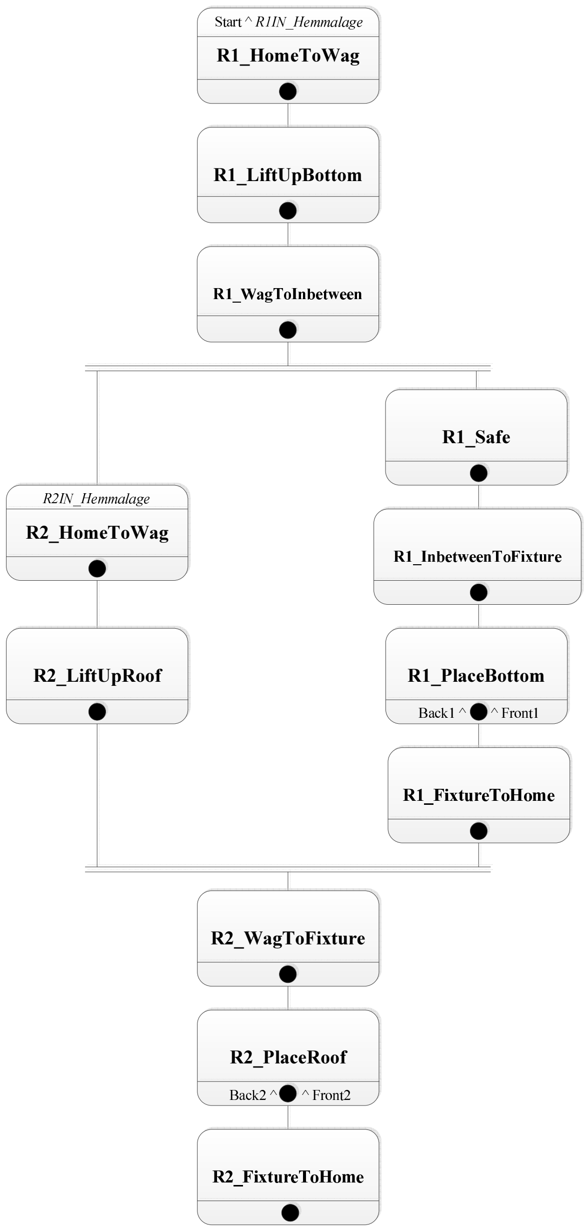


Figure A.4: SOP of the case study

Table A.1: List of operations executed during the nominal production of the case study

Operation	Description
R1_HomeToWag	An operation to move R_1 from R1Hemmalage to R1Wag
R1_LiftUpBottom	An operation to lift up the base plate from the wagon or do nothing when **LiftError** happens
R1_WagToInbetween	An operation to move R_1 from R1Wag to R1Inbetween
R1_Safe	An operation to do nothing or drop the base plate when **DropError** happens
R1_InbetweenToFixture	An operation to move R_1 from R1Inbetween to R1Fixture
R1_PlaceBottom	An operation to place the base plate on the fixture
R1_FixtureToHome	An operation to move R_1 from R1Fixture to R1Hemmalage
R2_HomeToWag	An operation to move R_2 from R2Hemmalage to R2Wag1
R2_LiftUpRoof	An operation to move R_2 from R2Wag1, lift up the roof plate from the wagon and then move R_2 to R2Wag2
R2_WagToFixture	An operation to move R_2 from R2Wag2 to R2Fixture
R2_PlaceRoof	An operation to place the roof plate on the fixture
R2_FixtureToHome	An operation to move R_2 from R2Fixture to R2Hemmalage

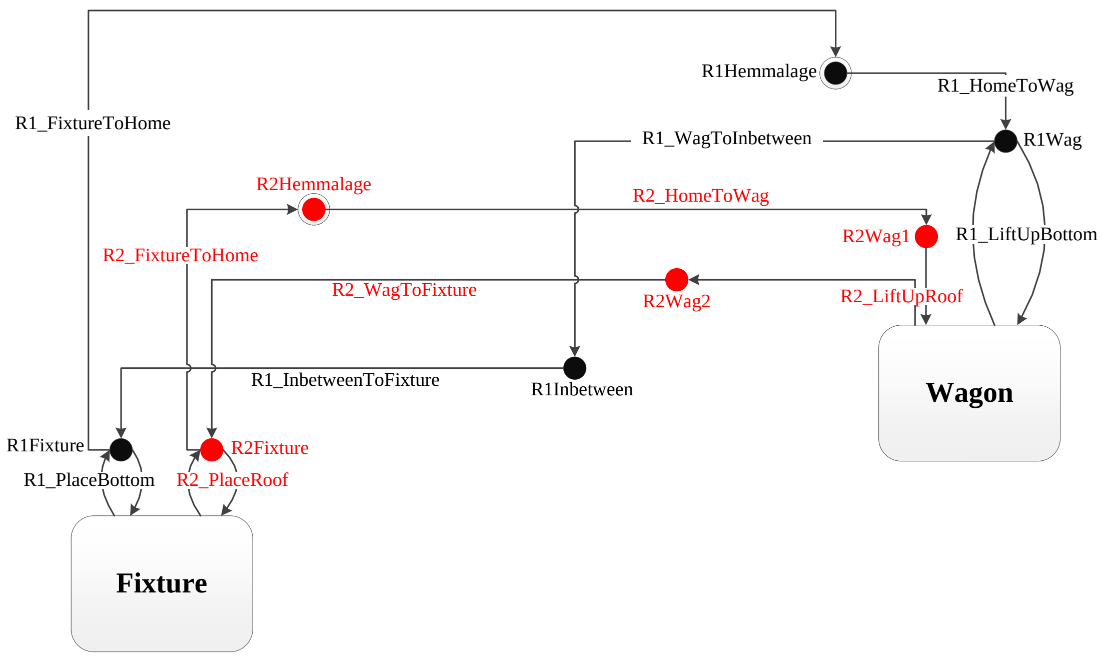


Figure A.5: R_1's safe positions are shown by solid red circles, and R_2's safe positions are shown by solid black circles.

is called:

1. In the physical system: *R1OUT_HomeToWag*
2. In the control system: *R1IN_HomeToWag*

It is important to note that, since operations are finished at safe positions, the manufacturing system raises handshaking signals at safe positions. In the next section, laboratory results from the implementation of the operation **R1_PlaceBottom** are given.

A.3 Implementation of an operation

R1_PlaceBottom is an operation to place the bottom plate on the fixture. This is executed by R_1. R_1 starts from the safe position R1Fixture, places the bottom plate on the fixture, and moves then back to R1Fixture. Figure A.6 shows this operation which is implemented in the Function Block Diagram (FBD) language. This is an operation block of the control system to run R1_PlaceBottom in the physical system. This block is supposed to send a command towards the robot controller in the physical system to start the operation and receive the sensor values from the physical system

Table A.2: Handshaking and safe position

Handshaking signal	Operation
R1IN_Hemmalage	**R1_FixtureToHome**
R1IN_HomeToWag	**R1_HomeToWag**
R1IN_LiftUpBottom	**R1_LiftUpBottom**
R1IN_WagToInbetweenPos	**R1_WagToInbetween**
R1IN_Safe	**R1_Safe**
R1IN_InbetweenPosToFixture	**R1_InbetweenToFixture**
R1IN_PlaceBottom	**R1_PlaceBottom**
R2IN_Hemmalage	**R2_FixtureToHome**
R2IN_HomeToWag	**R2_HomeToWag**
R2IN_LiftUpRoof	**R2_LiftUpRoof**
R2IN_WagToFixture	**R2_WagToFixture**
R2IN_PlaceRoof	**R2_PlaceRoof**

to analyze the operation state accordingly. When the pre-condition was satisfied, a number called the program number was sent to the robot controller. The robot controller includes several robot programs that are numbered the same as the programs. The robot program, to respond the control system, calls the robot program whose number was received. After terminating the robot program, the physical system raises the handshaking signal and then the control system monitors the post-condition value and determines whether the corresponding operation execution was finished or not (Figure A.7). The following sections explain implementation of the described procedure. First, the robot controller and its communication with the physical and the control systems are discussed.

A.3.1 Robot program

When a robot leaves a safe position, the manufacturing system resets the corresponding handshaking signal. Therefore handshaking signals are continuously updated according to the robot position.

Figure A.8 shows how the control system and the manufacturing system communicate with each other. In addition to the control system that is the central controller, there is a local controller

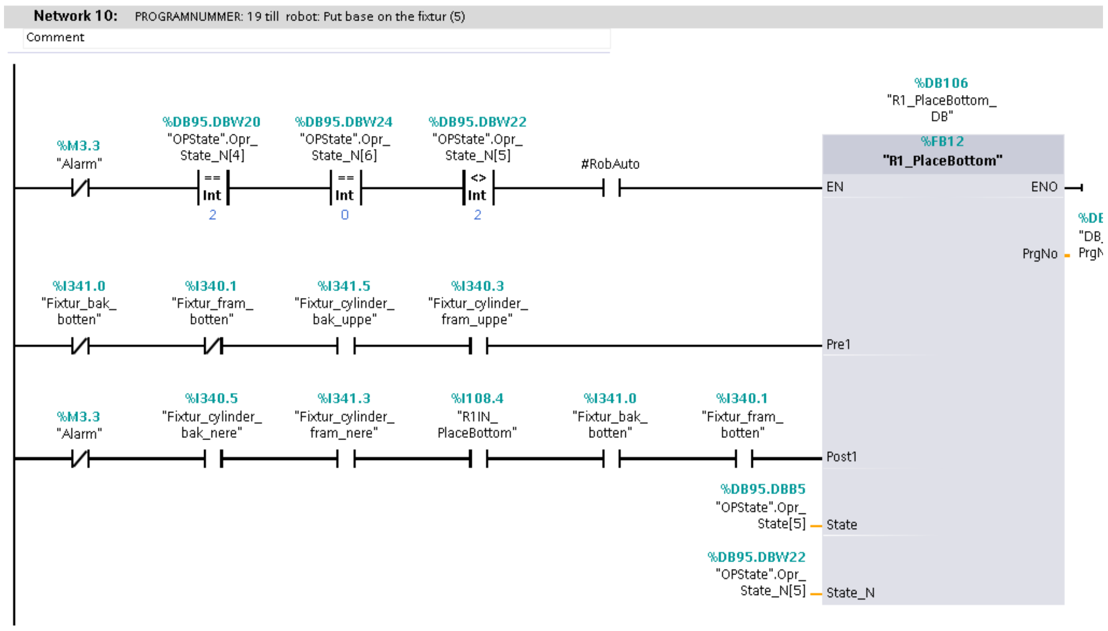


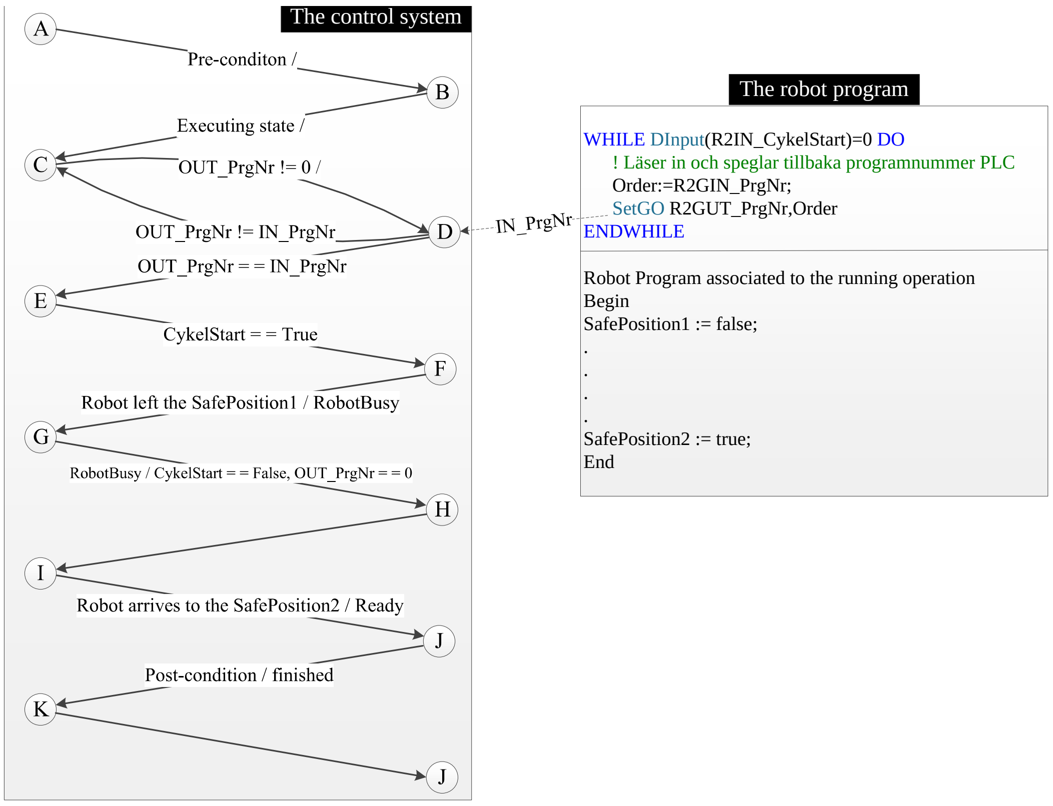
Figure A.6: The operation **R1_PlaceBottem** implementation in FBD language

for every robot in the manufacturing system. This controller is called a *robot controller*. When an operation is implemented in the control system, its equivalent *robot program* (or programs) should be implemented in the robot controller to communicate with the corresponding operation. The robot executes the robot program as a result of operation execution in the control system.

In this work, a number is assigned to each operation which is called a *program number* (shown by the variable **PrgNr**). Each operation is identified by its program number, the robot controller also knows these numbers. To execute an operation in the manufacturing system, the control system should send the operation program number to the robot controller. Robot controller always monitors **PrgNr**.

Figure A.7 and Table A.3 explain how **PrgNr** is exchanged during operation execution. When an operation is in its **executing** state (the state **B** of the left diagram in Figure A.7), the operation block sends the program number (**OUT_PrgNr**) to the robot controller. Since no operation maps to program number 0, sending a program number means that **OUT_PrgNr** is not zero. The parameter **OUT_PrgNr** of the operation block is connected to the parameter **IN_PrgNr** of the robot controller. To make sure that the robot received the same number as the program number, time the robot sends back the same program number. If these two numbers are equal, the control system sets the parameter **CykelStart** to **True**, which permits the robot to move and execute the corresponding robot program.

During execution of a robot program, the robot is busy (**RobotBusy** == **True**) and the robot controller cannot accept a new **IN_PrgNr**. When the robot arrives at a safe position, the robot status is changed to ready and if its post-condition is satisfied, the operation state is changed to



```
WHILE DInput(R2IN_CykelStart)=0 DO
    ! Läser in och speglar tillbaka programnummer PLC
    Order:=R2GIN_PrgNr;
    SetGO R2GUT_PrgNr,Order
ENDWHILE
```

```
Robot Program associated to the running operation
Begin
SafePosition1 := false;
.
.
.
.
SafePosition2 := true;
End
```

Figure A.7: The control and physical systems connection during operation execution

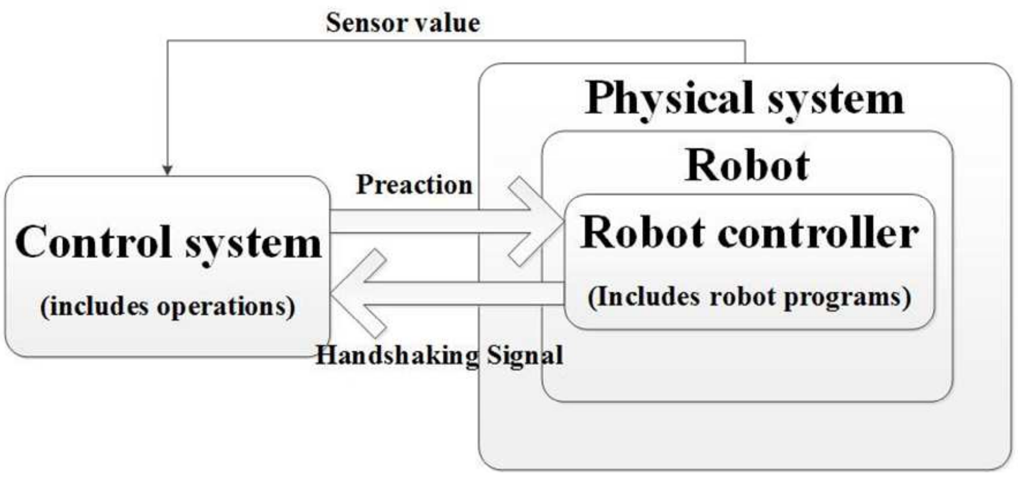


Figure A.8: The manufacturing system consists of the control system and the physical system

finished.

Table A.3: Parameters changes during operation execution

State name	State value	Robot Position	Cykel_ Start	Robot_ State	OUT_ PrgNr	IN_ PrgNr
A	OP1.init	SafePosition1	False	Ready	0	0
B	OP1.exe	SafePosition1	False	Ready	0	0
C	OP1.exe	SafePosition1	False	Ready	110	0
D	OP1.exe	SafePosition1	False	Ready	110	110
E	OP1.exe	SafePosition1	True	Ready	110	110
F	OP1.exe	?	True	Ready	110	110
G	OP1.exe	?	True	Busy	110	110
H	OP1.exe	?	False	Busy	0	110
I	OP1.exe	SafePosition2	False	Busy	0	110
J	OP1.exe	SafePosition2	False	Ready	0	110
K	OP1.fin	SafePosition2	False	Ready	0	110
L	OP1.fin	SafePosition2	False	Ready	0	0

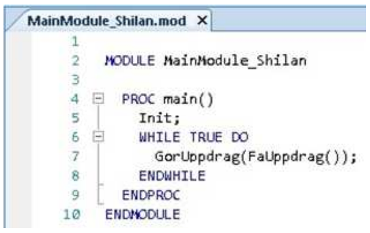
MainModule in Figure A.9 is the main program of the robot controller. Besides the **Init** module that resets variables to run a new procedure, there is a *while* loop monitoring the **PrgNr** continuously. As a result of receiving a new **PrgNr**, the robot controller runs the **GorUppdrag** module (Figure A.10), which is only a switch case of **PrgNr**. According to this selection in the **GorUppdrag** module, the relevant robot program start to run. Figure A.11 shows a robot program to move R_2 from the wagon position to the fixture position, R2WagToFixture.

As mentioned before, handshaking signals are switches to show the operation termination and consequently the robot position. If a robot has finished an operation and consequently has moved to a safe position, the relevant handshaking signal is true. When a robot leaves a safe position, the manufacturing system resets the corresponding handshaking signal. Therefore handshaking signals are continuously updated according to the robot position.

The robot program **R2WagToFixture** is a robot program that to move R_2 from the safe position R2Wag2 to the safe position R2Fixture. In Figure A.11, this robot program resets the handshaking signal *R2UT_LiftUpRoof*, as soon as the robot leaves the safe position R2Wag2 (Figure A.5). Also, this robot program raises *R2UT_WagToFixture* when the operation finishes and the robot is located at the safe position R2Fixture. In the next section, to continue the implementation of the operation **R1_PlaceBottom**, we discuss the state machine implementation.

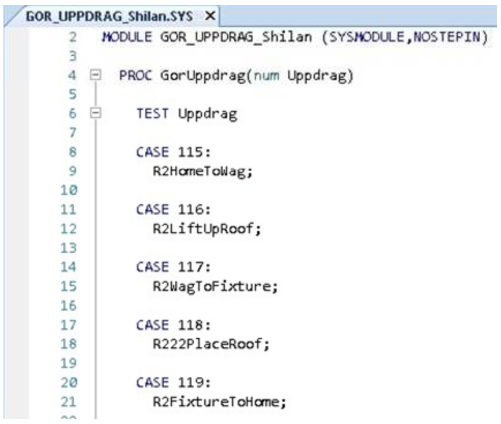
A.3.2 State machine

A basic state machine to calculate the operation state has been implemented in Section 3.1.1 (Figure 3.5), the same block is used in the PSL lab (Figure A.12).



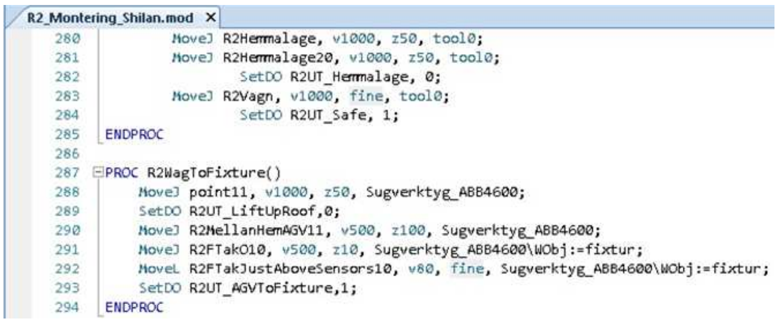
```

MODULE MainModule_Shilan

  PROC main()
    Init;
    WHILE TRUE DO
      GorUppdrag(FaUppdrag());
    ENDWHILE
  ENDPROC
ENDMODULE
```

Figure A.9: The main module of the robot program



```
MODULE GOR_UPPDRAG_Shilan (SYSMODULE,NOSTEPIN)

  PROC GorUppdrag(num Uppdrag)

    TEST Uppdrag

    CASE 115:
      R2HomeToWag;

    CASE 116:
      R2LiftUpRoof;

    CASE 117:
      R2WagToFixture;

    CASE 118:
      R222PlaceRoof;

    CASE 119:
      R2FixtureToHome;
```

Figure A.10: GOR_UPPDRAG_Shilan which is only a switch case of **PrgNr**



```
            MoveJ R2Hemmalage, v1000, z50, tool0;
            MoveJ R2Hemmalage20, v1000, z50, tool0;
                SetDO R2UT_Hemmalage, 0;
            MoveJ R2Vagn, v1000, fine, tool0;
                SetDO R2UT_Safe, 1;
ENDPROC

PROC R2WagToFixture()
    MoveJ point11, v1000, z50, Sugverktyg_ABB4600;
    SetDO R2UT_LiftUpRoof,0;
    MoveJ R2MellanHemAGV11, v500, z100, Sugverktyg_ABB4600;
    MoveJ R2FTakO10, v500, z10, Sugverktyg_ABB4600\WObj:=fixtur;
    MoveL R2FTakJustAboveSensors10, v80, fine, Sugverktyg_ABB4600\WObj:=fixtur;
    SetDO R2UT_AGVToFixture,1;
ENDPROC
```

Figure A.11: R2WagToFixture is a robot program to move R_2 from the safe position R2Wag2 to the safe position R2Fixture. First, the robot leaves the safe position R2Wag2, then the handshaking signal **R2Ut_LiftUpRoof** is reset.

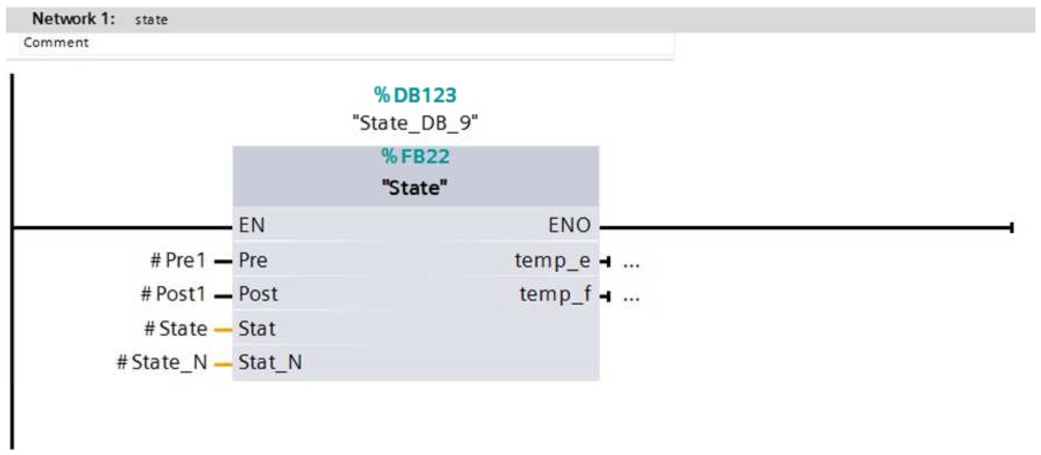


Figure A.12: The state machine block diagram in FBD language

Since, the operation state is a function of the previous state, the pre-condition, and the post-condition, the state value is an In/Out variable for the state machine block (Figure A.13). It must be mentioned that this is a general state machine for every operation. Since operations state has three possible values, the state is not a boolean variable. This variable is evaluated based on comparison and for convenience of comparing, it is saved as character (State) as well as integer value (State_N). Table A.4 lists the valid state values based on operation states.

Table A.4: State values and the numbers allocated to them.

Operation state	**State**	**State_N**
initial	i	0
executing	e	1
finished	f	2

A.3.3 Conditions and action

When R_1 starts to execute **R1_PlaceBottom**, there is no plate on the fixture and clamp locks are open. Thus, output values of 4 plate sensors (**Fixture_fram_tak**, **Fixture_fram_botten**, **Fixture_bak_tak** and **Fixture_bak_botten**) are False, and output values of **Fixture_cylinder_bak_uppe** and **Fixture_cylinder_fram_uppe** are True. The control and the physical systems status before **R1_PlaceBottom** execution is shown in Figure A.14. This figure shows that the operation **R1_InbetweenTo Fixture** must be finished before **R1_PlaceBottom** is started. Summarizing these conditions, **R1_Place Bottom**'s pre-condition is given in Figure A.15.

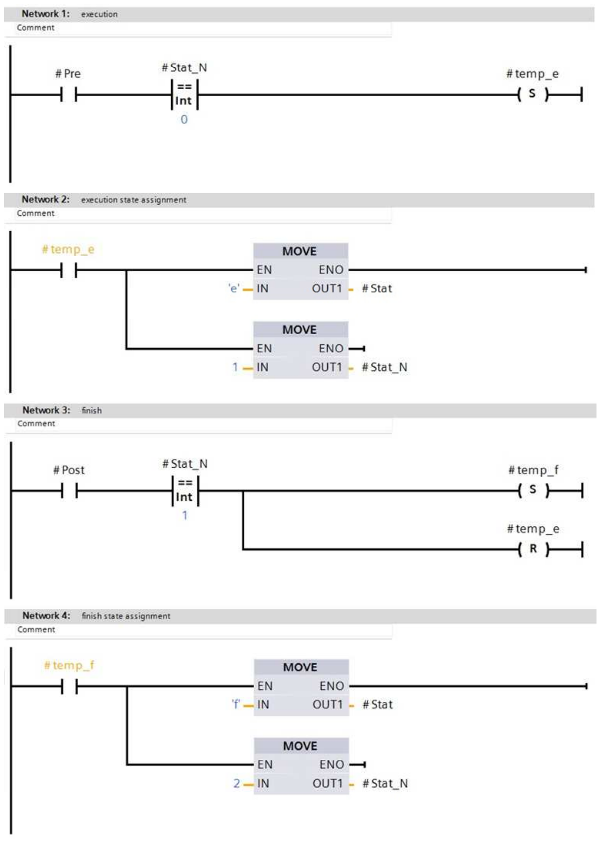


Figure A.13: Inside view of the state machine. The state is updated according to the last state value, pre- and post-conditions.

As soon as fixture's sensors, **Fixture_fram_botten** and **Fixture_bak_botten**, sense the bottom plate, clamp locks will be closed. Moreover, when R_1 comes back to R1Fixture, the physical system raises the handshaking signal (Figure A.14). These conditions constitute the post-condition of **R1_PlaceBottom** which is given in Figure A.16.

To start the corresponding pre-action in the physical system, the control system sends a relevant command during operation execution (Figure A.17). A command is a non-zero integer that the physical and the control systems use to differentiate operations, and consequently, their corresponding robot programs. When the physical system takes action, the command will be set to zero (section A.3.1).

A.3.4 Busy

No robot in the physical system can execute two different tasks at the same time. Thus, while a pre-action is running in the physical system, the control system stops sending the operating robot a new command. While the robot state under this circumstance is *Busy*, raising the handshaking signal shows the termination of this running operation, and changes this state to *Ready*.

Each robot in the manufacturing system has its own **Busy** signal. When **Busy** is True, the

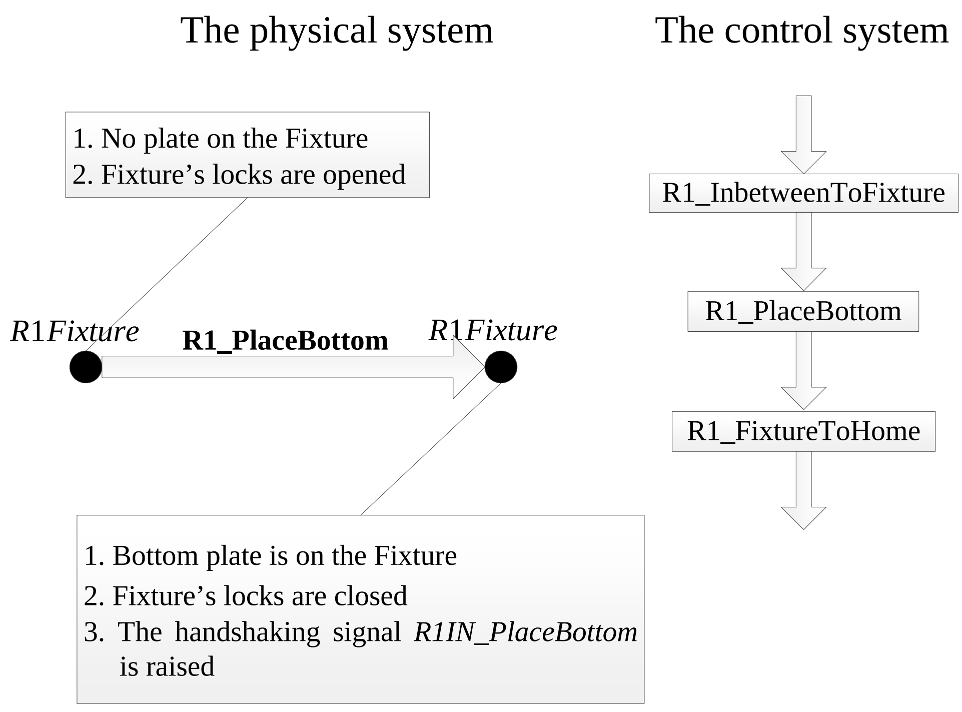

The physical system
The control system
1. No plate on the Fixture
2. Fixture's locks are opened
R1Fixture
R1_PlaceBottom
R1Fixture
1. Bottom plate is on the Fixture
2. Fixture's locks are closed
3. The handshaking signal R1IN_PlaceBottom is raised
R1_InbetweenToFixture
R1_PlaceBottom
R1_FixtureToHome

Figure A.14: The status of the control and the physical system before and after executing the **R1_PlaceBottom** operation

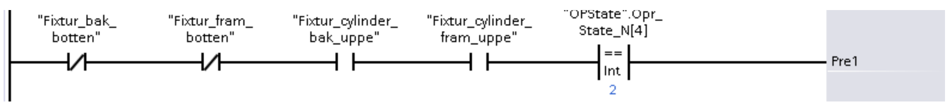

"Fixtur_bak_ botten"
"Fixtur_fram_ botten"
"Fixtur_cylinder_ bak_uppe"
"Fixtur_cylinder_ fram_uppe"
"OPState".Opr_ State_N[4]
==
Int
2
Pre1

Figure A.15: The operation **R1_PlaceBottom**'s pre-condition

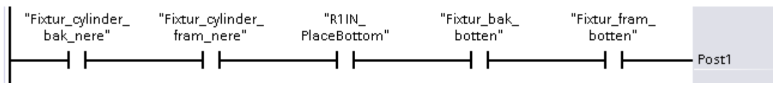


Figure A.16: The operation **R1_PlaceBottom**'s post-condition

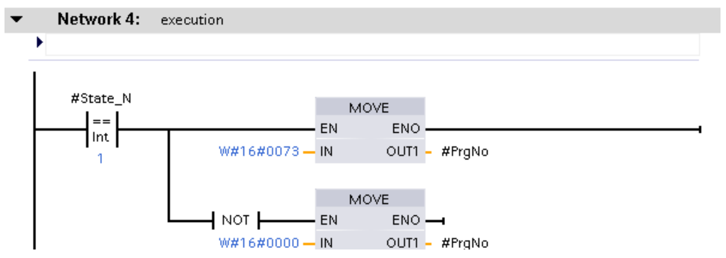


Figure A.17: The control system sends a relevant command during operation execution

robot is executing a pre-action. In other words, when a handshaking signal is True (Figure A.18), i.e. the robot is ready, the control system sends a new command to be executed.

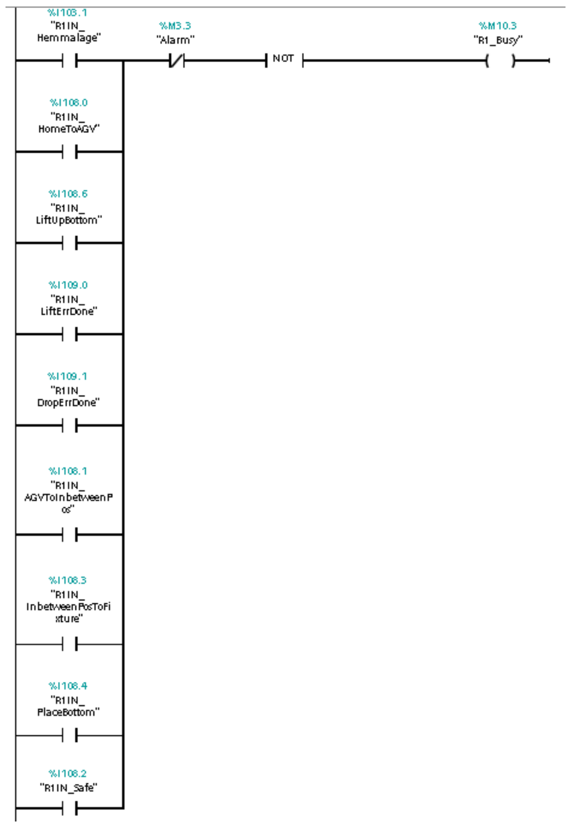


Figure A.18: The Robot busy status

Figure A.19 depicts that if any system error happens ("DB_Mode_Stn3".CykelStp is True), **Alarm** is triggered, or in automatic mode the robot is busy, the control system stops sending commands.

A.3.5 Enable

To ensure that only one operation command is connected to the command input port of the robot at any time, only one function block must be enabled every time. Otherwise, this leads to confusion and the robot cannot identify which command must be run now. Considering the operations sequence in the SOP, we decided that enabling operations according to SOP is the best way. Consequently, the operation block is allowed to communicate with the physical system, if and only if its earlier operation was finished and its later operation is still in its initial state. In other words, this function block is only **enabled** during this period of time.

Every function block is either enabled or disabled at any time. If **EN** is True, the operation block's inputs/outputs are connected and can receive/send value, otherwise, the block is disconnected from external variables. Figure A.20 shows the operation **R1_PlaceBottom**'s function

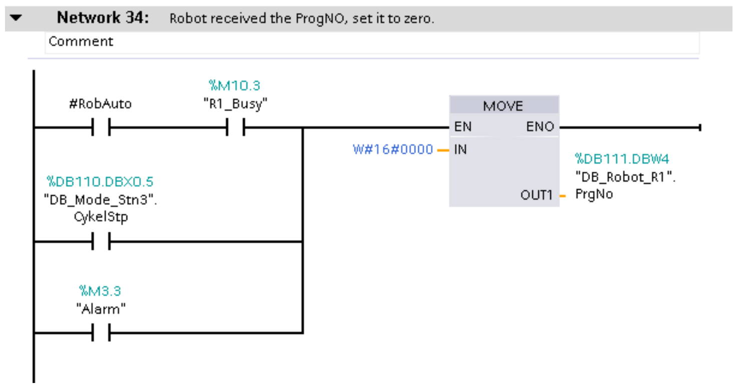


Figure A.19: The control system stops sending command if either an error occurs or the robot is busy.

block enabled by **EN**.

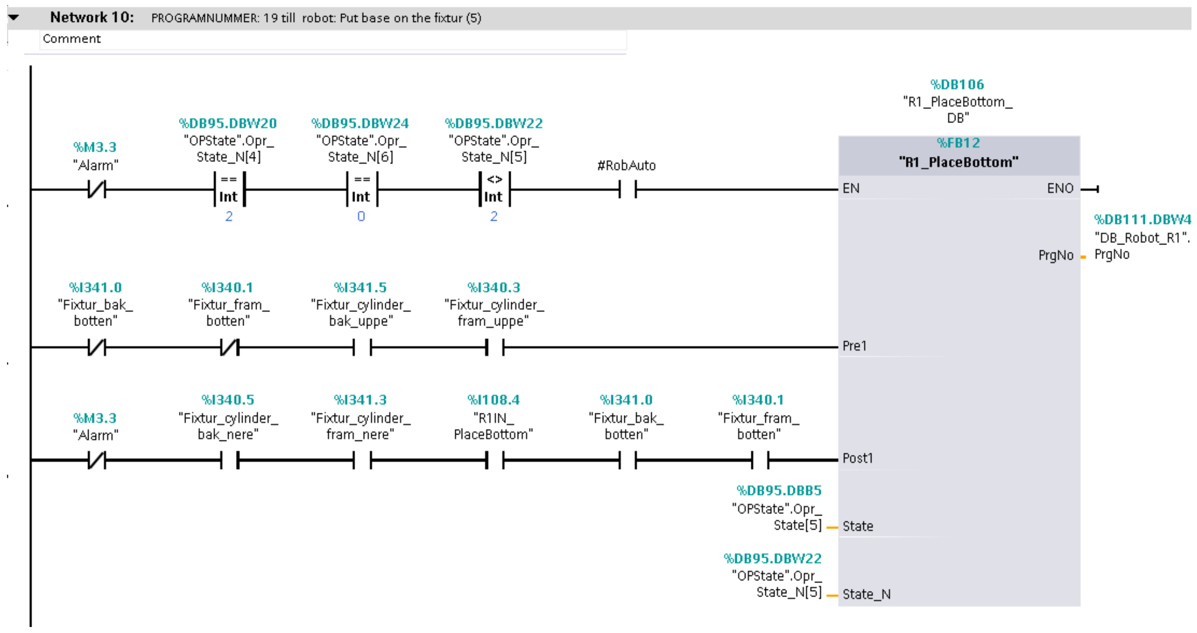


Figure A.20: The operation **R1_PlaceBottom** implemented using the port **EN**

Usually, two separate systems control each machine in a manufacturing system: a *local controller* and a *central controller*. The local controller is located beside the machine in the physical system. This controller is only used for the particular machine. On the other hand, there is a central controller used to control all machines in the manufacturing system. Consequently, there are two different control modes: the manual and the automatic mode; either the operator locally controls the machine by local controller in manual mode, or the central controller (the control system) controls the whole manufacturing system in the automatic mode. In this work, **RobAuto** is a variable showing the robot control mode. When **RobAuto** is true, the robot is in the automatic mode. Running a procedure, machines must be in the automatic control mode. Then, running an operation block in the control system, **RobAuto** is required to be True.

A.4 Restart from an operation

In this section, to verify the restart facility, we make an assumption that an error has occurred and the robot has not lifted up the bottom plate. Since the detection mechanism is only implemented in the operation **R1_PlaceBottom**, this is the first and the only operation of SOP to detect the error. This mechanism is discussed below.

As mentioned earlier, there are two sensors installed on the fixture (**Fixture_bak_botten** and **Fixture_fram_botten**) to sense the bottom plate. These sensors output values affect **R1_Place Bottom**'s post-condition. Sensing the bottom plate, the control system locks the clamp locks. Then, see Figure A.20, **R1_PlaceBottom**'s post_condition includes these clamp locks status.

Assuming that **R1_PlaceBottom** is in its **executing** state, this state should change to **finished** within six seconds of raising the handshaking **R1IN_PlaceBottom** by the physical system, otherwise, an error called a **TimeErr** is triggered by the control system (Figure A.21). This is the implemented error detection mechanism in this project. When Alarm is triggered, the restart menu pops up on the HMI screen.

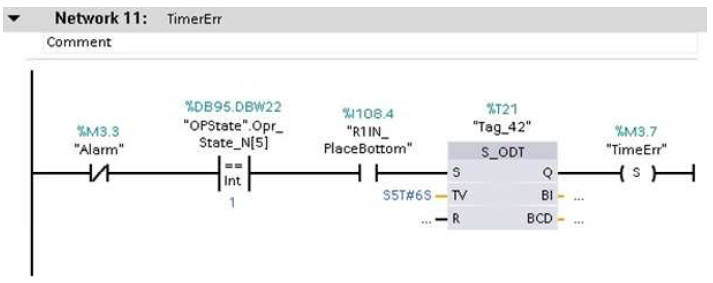


Figure A.21: Error detection mechanism

Figure A.22 shows that **Alarm** is triggered either when **TimeErr** occurs, or when the operator clicks on the Stop button. The Stop button is only enabled during execution of operations O_2, O_4 and O_6, since it was determined that only these were relevant for the simulated error case.

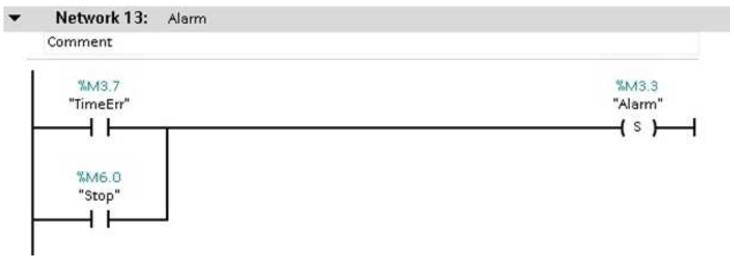


Figure A.22: Alarm is triggered either when the operator clicks on the stop icon, or when **TimeErr** occurs.

Figure A.23 shows the restart block whose variables are changed in the HMI screen. This block just assigns the right value to the state value according to the selected restart state. Selecting the **LiftUpBottom** as the restart state from the restart list (Figure A.24) triggers **RestartSel2**, and clicking on the **OK** button (to finalize the selection) triggers **OperatorOK**. This is shown by the green line in Figure A.23, which represents that the value is now true.

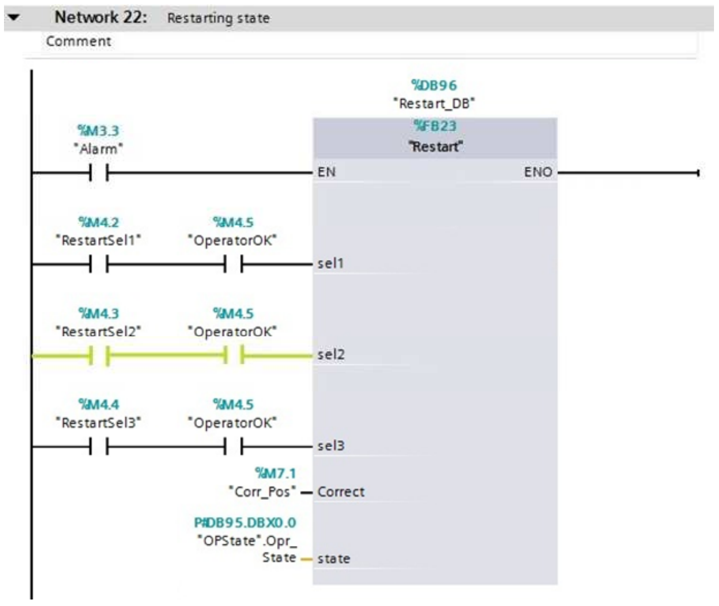


Figure A.23: The restart block assigns the operation state according to the selected restart state.

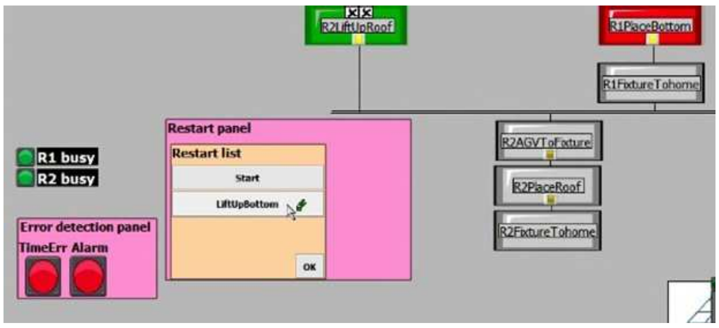
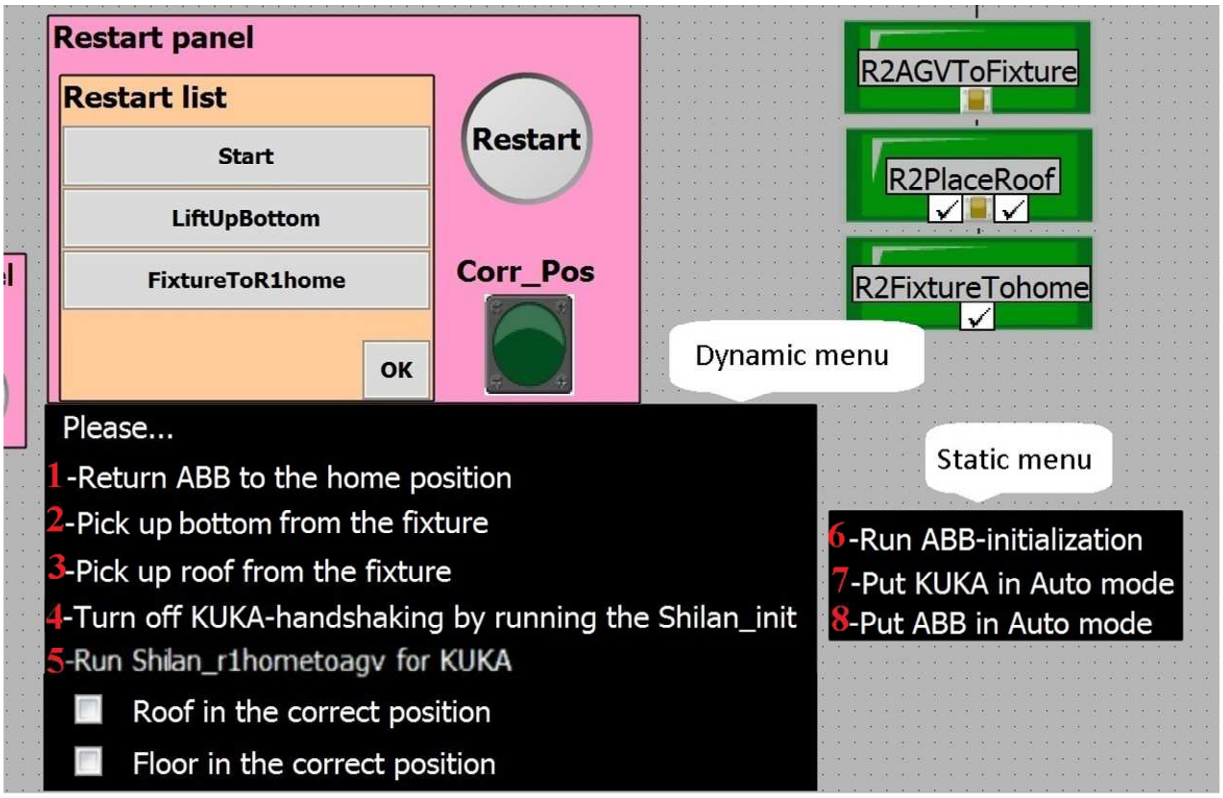


Figure A.24: LiftUpBottom is selected as the restart state.

On the other hand, once the operator finalizes his/her selection, the instruction menu pops up (Figure A.25). As shown in Figure A.5, to execute the operation **R1_LiftUpBottom**, the physical system must be in the safe position **R1Wag**. Then to re-execute this operation, the operator must place the physical system in this position.

The instruction list is divided into two menus: the *dynamic* menu and the *static* menu. The dynamic menu depends on which restart state has been selected, while the static menu is always



Please...
1-Return ABB to the home position
2-Pick up bottom from the fixture
3-Pick up roof from the fixture
4-Turn off KUKA-handshaking by running the Shilan_init
5-Run Shilan_r1hometoagv for KUKA
☐ Roof in the correct position
☐ Floor in the correct position

6-Run ABB-initialization
7-Put KUKA in Auto mode
8-Put ABB in Auto mode

Figure A.25: The dynamic menu and the static menu when **LiftUpBottom** was selected as a restart state.

the same for all selected restart states. Therefore, the static menu includes general requirements like putting machines in Auto mode (the items 7 and 8 in Figure A.25). Since, the robot R_2 is always reset from the beginning by the recovery procedure, initialization of R_2 is also included in the static menu (item 6 in Figure A.25, R_2 is called by the same name as its vendor).

Resuming the nominal production from **R1_LiftUpBottom**, execution requires the operator to move R_1 to R1Wag and R_2 to R2Hemmalage, then the handshaking signals *R1IN _Hemmalage* and *R2IN_Hemmalage* should after these movements be False and True, respectively (the items 1 and 5 in Figure A.25). **Shilan_r1hometoagv** is a robot program which is run by the R_1 controller to move this robot from R1Hemmalage to R1Wag. Whenever the handshaking signals *R1Hemmalage* and *R2Hemmalage* have the intended values, these two messages disappear from the dynamic menu.

Additionally, as mentioned in Chapter A.3.1, once a robot starts a new operation, the robot controller clears the handshaking signal of the previously executed operation. Then, at any time, at most one handshaking signal has the value True. Since, R_1 is going to continue from the second operation of the SOP, the handshaking signal of the first operation must be set to True. In this case, only *R1_INHomeToWag* has the value True, otherwise, message 4 appears. The instruction number 4, in the instruction list, asks to turn off all of the handshaking signals except *R1_INHomeToWag*.

Since in this physical state, plates are supposed to be in the wagon, there is no plate on the fixture. Then, installed sensors on the fixture sense nothing and their output value must be False, otherwise, messages number 2 and 3 in Figure A.25 appears. Finally, there are two safety related check boxes. Since there is no sensor installed on the wagon to check the availability of the plates, we rely on the operator to check plate availability and to confirm by clicking those check boxes.

To restart the system, execution of these instructions are required. Also, since all instructions are connected to measurable values (sensors, handshaking signals, or check boxes), their completion is also measurable. **Corr_Pos** is calculated by means of these values (Figure A.26).

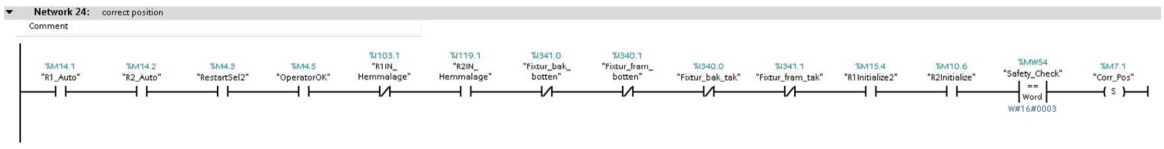


Figure A.26: Assesment of the correct position for restart.

Whenever all instructions are executed and their connected conditions are satisfied, **Corr_Pos** will be turn on and then the **Restart** button will appear. If the operator clicks on the **Restart** button, the current control state will be changed to the restart state by the restart block and the manufacturing will continue.