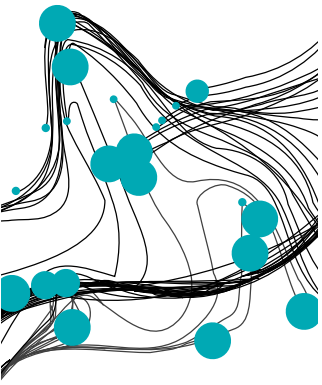
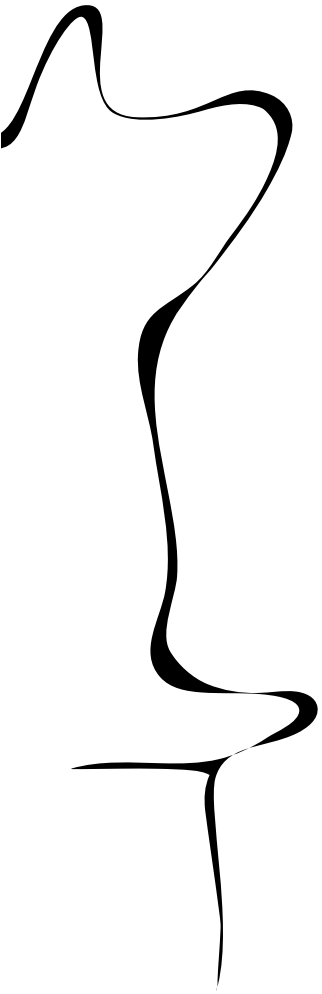


RAM ROBOTICS AND MECHATRONICS



Evaluation of the Structured Design Method on discrete-events in Cyber-Physical Systems

K.J.C. (Koen) den Hollander

MSC ASSIGNMENT

Committee:
dr. ir. J.F. Broenink
T.G. Broenink, MSc
dr. ir. A.B.J. Kokkeler

June, 2020

017RaM2020
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE
UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Summary

The design of reliable and robust Electronic Control Software for Cyber-Physical Systems becomes more challenging due to increasing complexity and the multi-disciplinary nature of these systems. A new design method has been developed, which is called the Structured Design Method, to aid in the design of these systems.

This method has been applied in earlier work in the design of Electronic Control Software for a continuous-time physical system, which is controlled by a discrete-time loop controller. In this report it is investigated how the applicability of the Structured Design Method is to design Electronic Control Software for a plant with discrete-events and if enhancements or adaptations of the method are necessary. In this assignment also the stakeholders of the design method are taken into account in the application.

The method is investigated by analyzing how it is applied in earlier work and what the requirements are of the method from the point of view of the stakeholders and for designing, integration and testing of discrete-event systems and signals in a Cyber-Physical System. The Structured Design Method is then used in a case study, a production-cell demo setup, to test the applicability.

Properties of the method that have been found during the analysis from the point of view of the stakeholders are the traceability of the method during application, design error detection mechanism and that the method must be easy-to-use. Properties of the method that have been found for the application to discrete-event systems and signals are on extensibility and interchangeability of the software architecture and integration of the discrete-event controller with the discrete-time controller.

It has been concluded that the Structured Design Method is suitable in the application to design Electronic Control Software for a plant that contains discrete-events. Adaptations in the application of the method in the case study, are the order of implementation of the functionality. There is iterated over the levels of detail instead of over the features, this differs from earlier work. An important property of the simulation framework is the interchangeability. Easy interchangeability of the framework supports testing of the Electronic Control Software.

The merging of the discrete-event control and the discrete-time control is important for the Structured Design Method because this is done in different iterations. A *dummy plant* is proposed as a temporary implementation of the discrete-time plant. This dummy plant introduces different levels of detail and helps to merge and test these systems in the Electronic Control Software. It has been shown in the application of the Structured Design Method to the case study that one level of detail can be implemented for multiple features in a single iteration, as long as the functionality in the level of detail is the same. Testing is an important part of the Structured Design Method, it helps to verify the correct behavior. Testing of the control logic of the Cyber-Physical System can be done using a proposed *event-based plant*. With this plant the control logic can be tested without implementing the discrete-time plants.

Contents

1 Introduction

1.1 Context

Due to the increasing complexity and multi-disciplinary nature of Cyber-Physical Systems, it becomes more challenging to develop reliable and robust Electronic Control Software for Cyber-Physical Systems.

Examples of complex Cyber-Physical Systems are autonomous driving cars and smart electricity grids. Challenges in these systems are in combining control, computation and communication (NWO-AES, 2018). Other challenges lie in dealing with unpredictable behavior of Electronic Control Software (Lee, 2008).

To aid in the design of Cyber-Physical Systems(CPS), a new design method has been developed which is called the Structured Design Method (Broenink and Broenink, 2019). This is a combination of a model-driven design approach and an iterative design method.

Developing reliable and robust Electronic Control Software(ECS) using the Structured Design Method(SDM) is done with models. The method starts with a simple model for which control software is designed. In every iteration more detail is added to the model and thus to the control software until the design is complete. This structured way of working helps to test the implemented functionality of the Electronic Control Software on a plant model, such that a reliable and robust implementation is achieved.

When designing Electronic Control Sofware for Cyber-Physical Systems, different systems are present in the models. Examples of these systems with their characteristics are:

1. **Networked Cyber-Physical Systems**
   A main parameter in this system is timing, which is an important parameter for real-time applications. Control loops rely on accurate timing for stable feedback loops and the delay and jitter must be within the maximum limitations.

2. **Control logic**
   The control logic that is implemented must contain the correct sequence to obtain the desired behavior of the system.

3. **Safety layer**
   The control loop and control logic rely on sensor input to make the right control decisions. When this input is not present anymore due to a sensor failure, appropriate actions must be taken to maintain the reliable and robust operation of the Electronic Control Software. Checks are performed on the calculated output signals to confirm that they are in bound and adjusted if necessary.

4. **Control law**
   The control laws only operates in a proper way when the timing constraints are met. The task of the control loop is more than only calculating a new control value. Reading of sensors and sending control signals to actuators are also part of this task. These tasks influence timing delays in the control loop.

5. **Sensor**
   The data from sensors can contain noise and can be delayed. These parameters can influence the stability of control loops.

6. **Actuator**
   Due to parameter variation in actuators the control signals are not converted to the desired input of the system, this can lead to instability of the control loops.

The abovementioned systems can be divided based on the type of systems, these are continuous-time systems, discrete-time systems and discrete-event systems. The abovementioned systems are categorized below based on the type of system:

- Networked Cyber-Physical Systems, control logic and safety layer are discrete-event systems. The signals in these systems change on arbitrary moments in time.
- Control laws for plants are an example of a discrete-time system. This tasks is executed on equidistant time intervals.
- Sensors convert the continuous-time signals of a plant to discrete-time signals for the digital controller to work with the signals. Actuators convert the discrete-time signals of a digital controller to continuous-time signals for the plant to work with the signals.

Prior work that has been done on the Structured Design Method is the development of the method and the application of the method to design Electronic Control Software for a Segway (Broenink and Broenink, 2019). Other research that has been done is applying the Structured Design Method to design the Electronic Control Software for a slider setup (Morsinkhof, 2019).

In these two research projects the Structured Design Method has been used to design discrete-time loop controllers to control continuous-time systems. The method has not been applied in the design of other controllers than combinations of continuous-time systems and discrete-time systems, such as the Segway.

Further investigation of the Structured Design Method is necessary to gain more insight on how the design method works when it is applied to other type of systems. This can lead to a better applicability of the Structured Design Method.

1.2 The Structured Design Method

In this section an overview is given on the Structured Design Method (SDM); this method has been developed in (Broenink and Broenink, 2019). This design method is used throughout this research assignment and is therefore discussed. The information in this section is copied from (Broenink and Broenink, 2019). A graphical overview of the method is given in Figure 1.1.

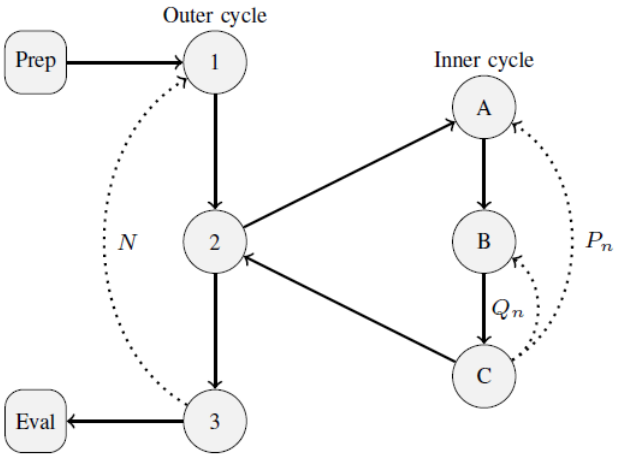


Figure 1.1: Graphical overview of the Structured Design Method (Broenink and Broenink, 2019)

The steps involved in the Structured Design Method are stated below (Broenink and Broenink, 2019).

→ Order and split the features and levels of detail as preparation
1. Design new feature, models and tests

2. Implement and test a new feature
   (a) Design a feature based on an ideal model
   (b) Combine the feature with a more detailled version of the rest of the system, adding more detail when needed.
   (c) Determine if the tests pass, if so add more detail(go to B). If it fails, redesign the feature(go to A)
3. Continue to next feature, untill all features are implemented

→ Evaluate and reflect on the cycle process

The method starts with the preparation phase, here the functionality of the system under design is determined. The functionality is split into features and the order is determined in which they are implemented. Then the features are split into different levels of detail. These levels determine the detail that is implemented in a model on which the feature can be tested after it has been designed.

The method consist of two cycles, an inner and an outer cycle. The design starts in the outer cycle where the feature is designed and tests for evaluation are developed, this is done for the different levels of detail. The next step is to enter the inner cycle to implement and test the feature.

First a simple model of the plant with the corresponding functionality of the controller is implemented and tested. When the desired behavior is obtained more detail is added to the model and more functionality is added to the controller. This process is repeated until all levels of detail are implemented and a detailed realization of the plant has been achieved. This are steps $A \rightarrow B \rightarrow C$.

If the complete feature is implemented with the desired level of detail, the next feature can be implemented until all features are added. This are steps $1 \rightarrow 2 \rightarrow 3$. When all features are implemented, the complete design is evaluated and the design process is reflected upon. The advantage of these small design steps and testing between levels of detail is that the process is traceable and increases the probability that design errors are detected in an early stage. This decreases the Cost of Change and the Chance of Failure.

1.3 Problem statement

To help in the design of Electronic Control Software for Cyber-Physical Systems, the method must be able to help in the design of different types of systems. Figure 1.2 shows a schematic overview of a Cyber-Physical System, together with the type of signals that are present between the different blocks. It can be observed that discrete-time and continuous-time signals are present in the control loops and the physical systems and that discrete-event signals are present in the embedded software.

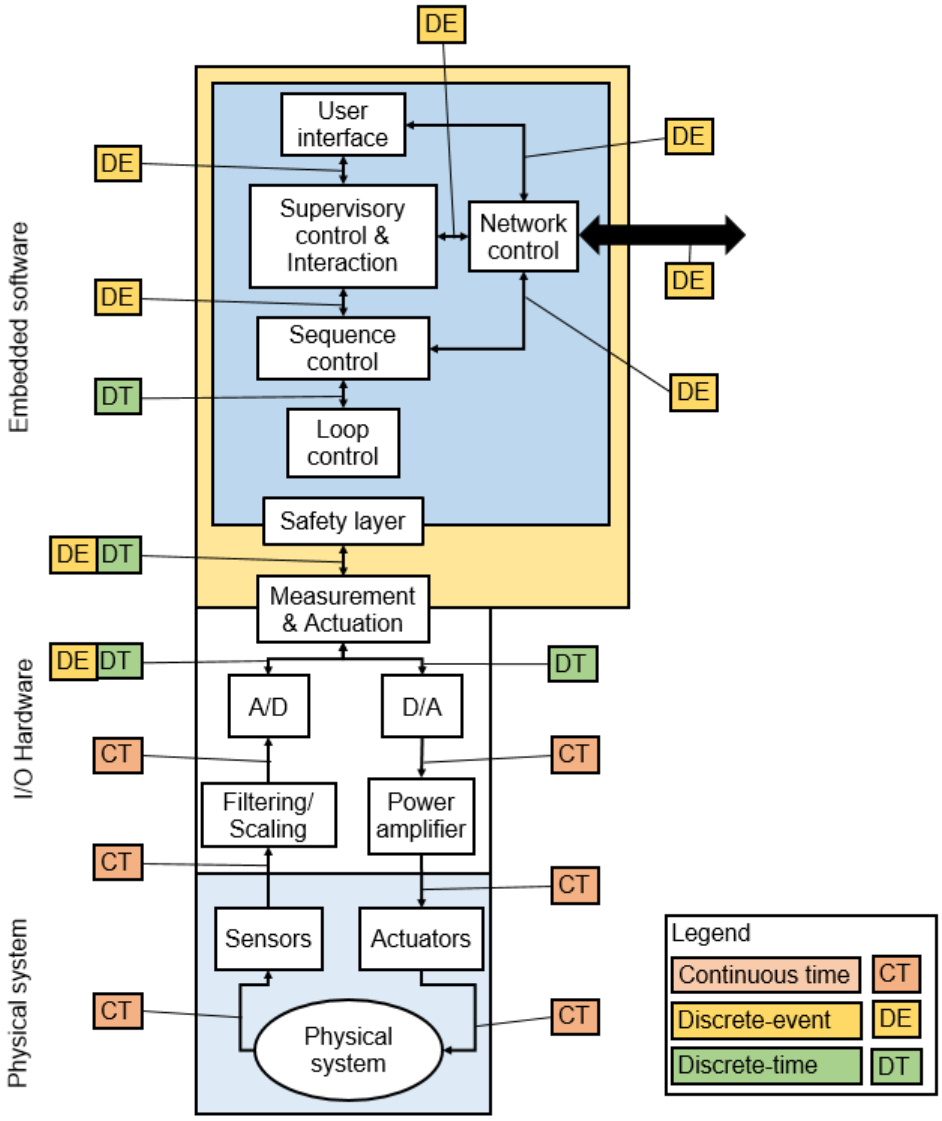


Figure 1.2: Adapted generic architecture of a Cyber-Physical System. Between the system blocks the type of signals that are present are stated. On top is the cyber part, on the bottom the physical part and in the middle the combining parts. This figure has been derived from (Broenink et al., 2016).

1.4 Research objective

The research objective of this Master Thesis is:

Investigate the applicability of the Structured Design Method to the design of Electronic Control Software to physical systems that contain discrete-events.

This objective is important for the Structured Design Method to be applicable in the design for different components of the Electronic Control Software. In this way discrepancies can be identified and enhancements or adaptations of the SDM can be determined. To gain more insight in the Structured Design Method, the method is being tested for the design of components that use discrete-event signals. To test the SDM for these type of signals, the Structured Design Method has been used to design Electronic Control Software for a plant that contains discrete-events. By using this plant, not only discrete-time signals are available but also asynchronous signals, or discrete-event signals; with these signals ECS can be developed for other components of the system.

In prior work only the application of the design method has been investigated. In this project other aspects of the method will also be highlighted. For the SDM to have an impact on the design community, the SDM will be analyzed from the point of view of the stakeholders and this will be taken into account in the assessment of the application of the SDM.

1.5 Approach

To test the applicability of the Structured Design Method, the following steps are taken.

In a first step, prior work of the application of the SDM to the design of ECS for the Segway setup has been analyzed. This step gives information on how the method is applied to discrete-time and continuous-time signals. Another research that will be discussed is how the functionality is structured for simulation.

As a second step an analysis is done from the point of view of the stakeholders on designing a CPS. In the design of a Cyber-Physical System different stakeholders are involved which all have challenges of their own and these challenges are identified in this analysis.

Another analysis has been done on the different components that are developed in a Cyber-Physical System. With this analysis, design challenges are identified that are present from an application point of view to discrete-event signals.

In a third step the SDM is applied to a case study. To test the SDM on discrete-event signals, a case study has been used to obtain results about the application of the method. This test case is designing of Electronic Control Software for a production-cell setup (van den Berg, 2006).

The production-cell setup constist of multiple motors, encoders and end switches. Also block detectors are present that change their output value at arbitrary time instances when blocks are detected and are thus discrete-event signals. This makes the production-cell setup a plant that contains discrete-events and is therefore an ideal case study to test the SDM in the design of ECS for discrete-event systems.

After an analysis from different point of views on the SDM and the application of the method to a test case, the method can be evaluated and conclusions can be drawn.

Possible results of this research are that the method needs adaptations or enhancements to work more optimal with discrete-events.

1.6 Report structure

In Chapter 2 background information is given on structuring of models for simulation and the application of the SDM to a Segway robot.

Chapter 3 gives an analysis on the design challenges from the point of view of the stakeholders and an analysis is done on the SDM for implementation of discrete-event signals.

Chapter 4 describes the application of the SDM to the production-cell setup.

In Chapter 5 the application of the SDM to the production-cell setup is evaluated and properties of the design method are highlighted which are observed during the application to the production-cell.

Chapter 6 gives conclusions on the application of the SDM to discrete-event signals.

2 Background

In this chapter the application of the Structured Design Method to the Segway is discussed (Broenink and Broenink, 2019), together with the research on structuring of the models into features and level of detail (Broenink and Broenink, 2018). In this way initial insight of the application of the Structured Design Method is given.

In Section 2.1 the structuring of the models is discussed and in Section 2.2 the application of the SDM to the Segway is discussed.

2.1 Model structuring

In (Broenink and Broenink, 2018) an approach is presented to structure models and divide them into features and levels of detail. This structure can be used to simulate a system in different levels of detail in a structured way. The discussed design process is also used in the SDM and is therefore of interest for the application of the SDM to the production-cell setup.

The following steps are performed to create a suitable structure for detail variation (Broenink and Broenink, 2018).

1. Identify detail invariant signals, including but not limited to:
   (a) Physical quantities
   (b) System states
   (c) Representations of physical quantities
2. Define interface and model structure based on detail invariant signals
3. Model submodels based on behavioral or port-based modelling principles
4. Identify region of validity of simplifications of model effects, classify these as
   (a) Essential
   (b) Hard limit
   (c) Soft limit
5. Create models based on required regions of validity

When these steps are performed, submodels are derived which can be used in simulations with different levels of detail.

It can be concluded from this research that the structuring process only deals with continuous-time plant models. In plants containing discrete-events, no sensor noise or motor limitations are present for example, so other abstractions of the plant are necessary to determine the levels of detail.

2.2 Application of SDM to Segway

In the research (Broenink and Broenink, 2019) the Structured Design Method has been developed and applied for the design of ECS for a Segway. The method has been used as is described in Section 1.2.

The first step in the application of the SDM is the preparation phase. Here the features and levels of detail are determined. The functionality, limitations and levels of detail are from (Broenink and Broenink, 2019). The functionality, or features, that must be implemented are:

- Balancing the mini-Segway
- Steering the mini-Segway
- Driving forward and backward

From these features limitations are identified, these are:

- Maximum power of the motors
- Sensor behavior and limits
- Sampling frequency of the analog-to-digital converter in the Raspberry Pi

The levels of detail that are identified from the limitations are:

- Non-linear model
- Motor limitations
- Sensor models including limits on readout
- Discrete-time models
- The prototype

In this stage the features and levels of detail are known and the design cycles can be started.

It is observed in the design cycles of the Segway that the features are implemented one after another and that there is not switched between features during implementation of one level of detail. A table has been made which shows the implementation order of the features with the levels of detail, see Table 2.1. In the case of the Segway, it is not possible to switch between features when one level of detail is implemented; the Segway cannot steer before it is balancing. There is a dependency of the steering feature on the balancing feature.

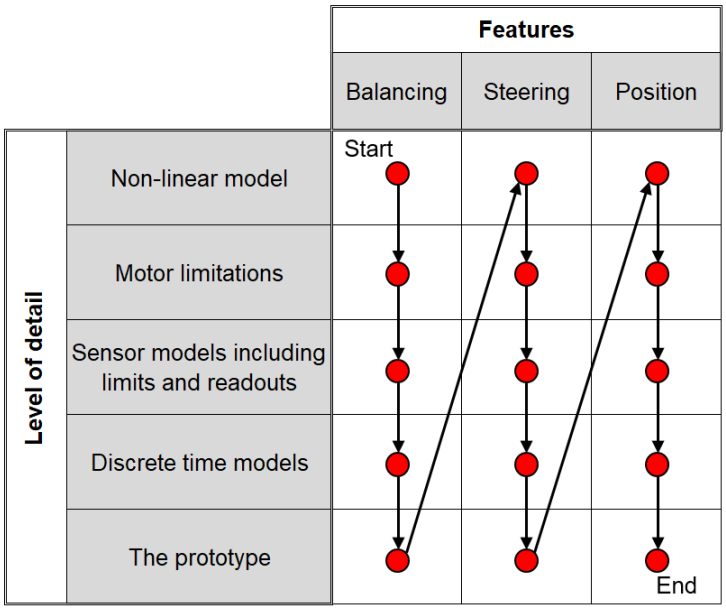


Table 2.1: The order that is used to implement features and levels of detail for the Segway

It can be concluded from this application of the SDM that the order in which the level of detail is implemented is the same for every feature. In a different application it could be usefull to deviate from this order to get a more optimal implementation order.

3 Analysis

In this chapter an analysis is performed on the design of a CPS from the point of view of the stakeholders. With this analysis challenges are obtained which are encountered. From these challenges requirements can be derived which can be used to assess the performance of the SDM.

To obtain challenges from an application point of view of the SDM to discrete-event signals, an analysis is performed on the different parts that are designed in a CPS.

3.1 Design method challenges for stakeholder

For the SDM to have impact, the stakeholders of the method need to see its added value. Many resources are involved in the design of complex CPS. These resources need to be managed in an efficient way and this comes with challenges, being identified here. The stakeholders that have been identified are shown in Figure 3.1. The stakeholders are identified as having the highest stakes in the development of a CPS and are of imporance for the usage of the SDM.

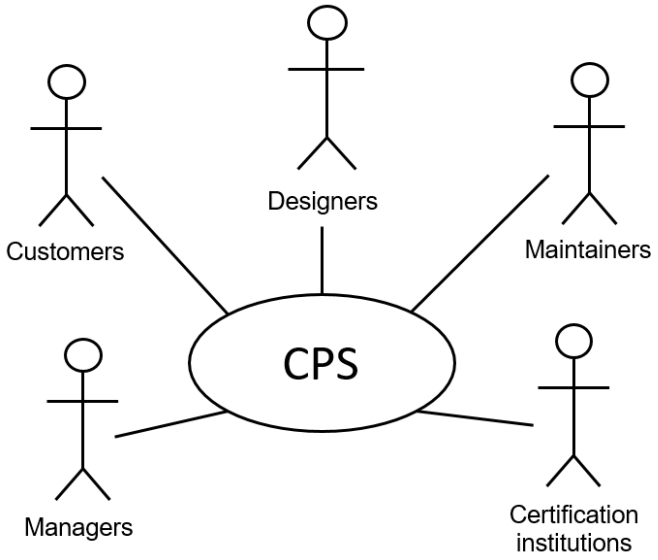


Figure 3.1: The stakeholders that are identified as having the highest stakes in the development of a CPS and are of importance for the usage in the SDM.

The stakeholders that are present in the design of a CPS are stated below, together with aspects that are of interest.

1. **Designers**
   These people apply the SDM to design ECS for CPS. For them the method must be practical and easy-to-use. The method must be applicable to different systems such that different design teams can collaborate in a efficient way. When a CPS is designed in a multidisciplinary team, all these teams have their own formalism and culture (Fitzgerald et al., 2015). A unified way of working can help to overcome this problem.

   When the CPS is overhauled, the design decisions that have been made need to be present. When the initial design process has been logged, design decisions can easily be retraced which improves the time efficiency and thus design cost. This holds also for product improvement, this is easier when the design decisions are easily available. This prevents that design errors are repeated.

2. **Managers**
   They are the supervisors of the designers. The goal for managers is to manage resources in an efficient way, such that a profit can be made. For managers, cost and time efficiency are important. Cost go up if design errors are made which are detected in a later stage and as a result a lot of redesign is necessary. This redesign also takes more time for the design to be finished which suppresses profit. Due to a better time efficiency the system design is finished in less time.

3. **Customers**
   These are the end-users of the designed CPS. The method must help to fulfill the requirements which are stated by the customer. They can help to give more insight in the translation from requirements to a design that fit the needs. Another aspect is the time efficiency, which could surpress the cost of the CPS. A high quality design of a CPS will deliver the customer a robust system which can last for the complete life-cycle.

4. **Maintainers**
   When a CPS is maintained or repaired, sometimes parts are not available anymore and a replacement has to be found. For these decisions to be correct, design decisions must be available which were made during the design of the system. For maintainers it is therefore important that design decisions can be retraced and thus traceability is important of the design process.

5. **Certification institution**
   This stakeholder has the task to formally review the design of the CPS (Fitzgerald et al., 2015). To review the design, the design steps and design decisions that are taken and implemented must be available. This means that a mechanism must be embedded in the design method which makes these steps and decisions easily available.

The elements that are of interest for the SDM are stated below.

1. **Practicability**
   To make a design method user-friendly, it must be easy to learn and apply. Otherwise the method will not be appreciated by the designers and the impact of the design method will be low.

2. **Internal traceability**
   This type of traceability makes it possible for designers and managers to see where they are in the design process and which decisions have been made.

3. **External traceability**
   When a certifying institution wants to review the design, all the necessary information is available in an easy way.

4. **Time efficiency**
   When designers and managers work in a routine way they get more acquainted with the design method and this will accelerate subsequent steps in the design process.

5. **Cost**
   If due to routine way of working design time is reduced, then also the design cost are reduced.

6. **Design error detection**
   When a design error is detected in an early stage this reduces design time and design cost.

7. **Applicability**
   A design method must be available for all different domains. This helps the different design teams within an organization because they operate using the same method which lead to a uniform culture and formalisms.

Now requirements can be derived from the identified challenges. To make well defined requirements the EARS method (Mavin and Wilkinson, 2010), which stands for "Easy Approach to Requirement Syntax", is used.

1. The design method shall have traceability properties, internal and external.
2. The design method shall be generally applicable, such that different design teams can use the same method.
3. The design method shall have an design error detection mechanism, such that design errors are averted and design time and design cost are supressed. These three elements are strongly related.
4. The design method shall be easy-to-use, this will improve the practicability of the method and as a result will improve the time efficiency and reduce cost.

When these requirements are fulfilled, it is believed that working with the design method achieves a process which has impact on the design process during the design of a CPS.

3.2 Challenges during application of SDM

An analysis is done on the cyber related components and aspects that are present in a CPS. The point of view that is used is that from designing, integration and testing discrete-event systems and signals in a CPS. The analysis is performed according to the components (blocks) that are present in Figure 1.2.

Electronic Control Software architecture

The challenge of the Electronic Control Software is the ability to cope with change. When new modules are added they must be integrated in an easy way. This is important for the SDM because in every design cycle new functionality is added, the architecture must support extensibility.

For the integration of the ECS, which runs during the design process on different devices with the usage of a virtual and real plant, the interchangeability is important. When it is easy to switch between the different simulation scenarios it is possible to adapt fast when errors are detected and changes have to be tested. These simulation scenarios can be divided in three cases, which are stated below.

1. **Development scenario**
   In this scenario the development device is connected to the virtual plant. This connection does not require any adapter because the signals which are send between the entities are real physical units. In this stage the cyber part of the CPS is under development and the simulation scheme can be used to test the implementated functionality of the ECS on the virtual plant.
2. **Hardware-In-the-Loop scenario**
   In this Hardware-In-the-Loop(Isermann et al., 1998) scenario the target device is connected to the virtual plant via a adapter. This adapter converts the control signals, which come from the controller, to physical units that can be used to control the virtual plant. In this stage the cyber part is tested to verify that the target device executional aspects are correct.

3. **Real scenario**
   In this scenario the target device is connected to the real plant. In this scenario no adapter is present because the control signals from the ECS can interface to the real plant directly. In this stage the cyber part is tested on the real plant to verify that the complete implementation is correct.

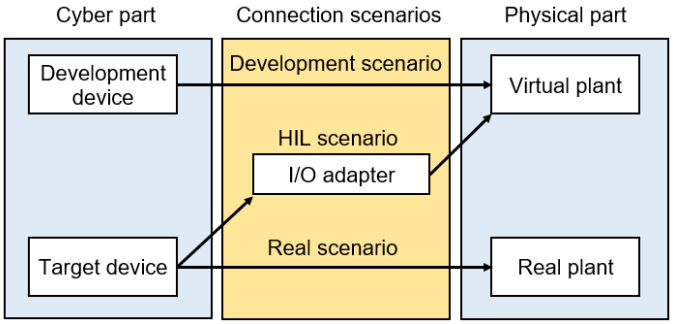


Figure 3.2: Overview of different connection scenarios for simulation which are present throughout the design process.

It can be seen in Figure 3.2 that the ECS runs on different devices and these devices must be able to connect to different plants. Depending on the interfaces, an adapter is present for the correct conversion.

User Interface

The User Interface (UI) usually runs in a soft real-time manner. An important factor is the time that it takes of for the communication of the UI with the controller. If the UI runs on a separate computer the communication can takes more time and this have to be taken into account to ensure the main computer runs in-time and the hard real-time constraints are still met.

Supervisory Control & Interaction

For the discrete-event systems, such as the supervisory control and control logic, it is important that the desired behavior of the plant is implemented for testing. The implemented functionality on the ECS is tested on the virtual plant to verify the correct behavior before it is used on the real plant. Here it can be seen that the interface is again of importance for interchangeability. The plant that is used for testing must be able to give the controller the necessary information to make discrete-event control decisions.

A plant is proposed that is used for testing the control logic, this is the *event-based plant*. This plant contains simplified behavior of a discrete-event system. As an explanation of this concept an example is given of a transportation belt that transport blocks.

This example contains a transport belt that is driven by an electric motor, at the end of the belt is a block detection sensor. When this block detection sensor detects a block the belt is stopped.

The electric motor is controlled by the signals *run* and *stop* instead of a real valued control signal. When the received signal is *stop*, the blocks are not moved by the transportation belt. When the received signal is *run* the blocks are moved forward. The belt does not consist of a continuous belt but of discrete positions. A block detection sensor is connected to the last position. If a block is present on this position, the sensor value changes and this is received by the control logic. This changes the signal to the electric motor from *run* to *stop*. A graphical overview of this event-based plant is shown in Figure 3.3.

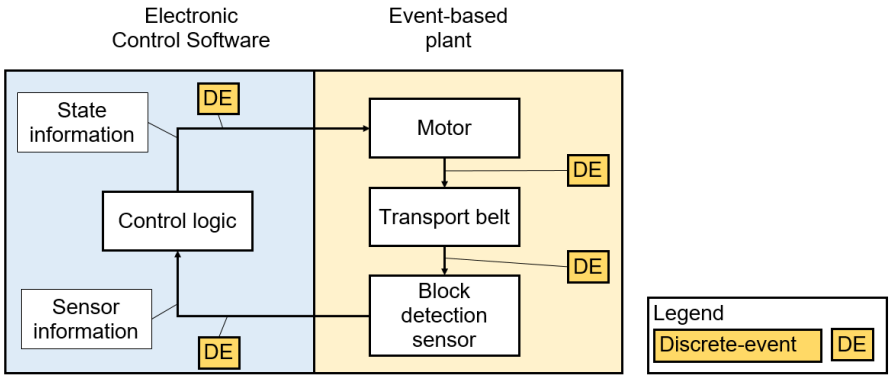


Figure 3.3: Graphical overview of an event-based plant. The output signals from the control logic are discrete control signals instead of real valued control signals. The input signals from the plant are the discrete signals from the block detection sensor.

By using this scheme, the discrete-time plants do not have to be present to test the control logic in a simulation. In a later stage this event-based plant can be changed to a discrete-time plant which receives real valued control signals as an input and gives real positions as an output. When the transportation belt is velocity controlled, the velocity can then be mathematically integrated for a block that is on the transportation belt to change the position of the block.

With this event-based plant, the plant implementation can have different levels of complexity. First finite states and later real valued signals. These levels of complexity can be used as different levels of detail in the application of the SDM.

Loop Control

The loop control interface is important for easy integration with the control logic. The control logic can for example provide the setpoint to the loop controller and requires the current value as feedback for discrete-event control decisions. This architecture has to be established first, such that the implementation does not has to be changed when the control logic and the loop control are integrated. In Figure 3.4 a proposition is made for the architecture of how the different components can provide the necessary information to where it is needed.

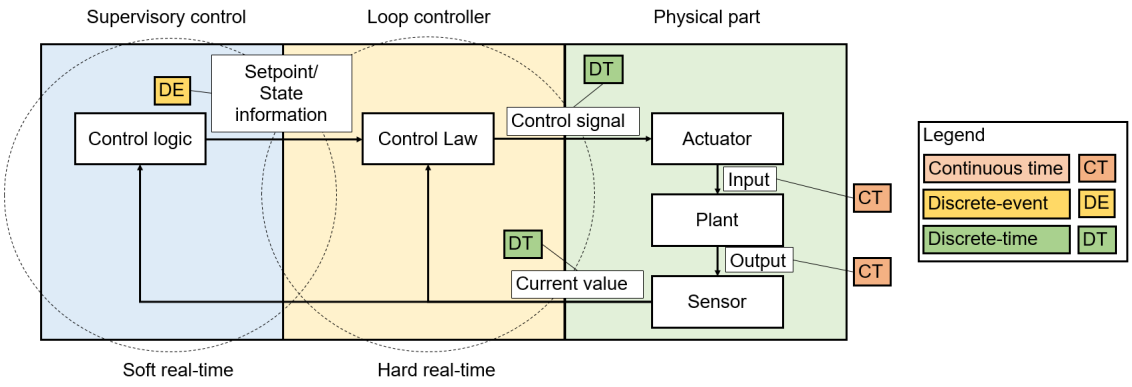


Figure 3.4: Proposal for the architecture for the control logic, loop controller and plant. This proposition supports easy integration

The loop control also needs states for initialization, on and off for example. With these states the supervisory control can control the loop control according to the state of the supervisory controller.

The order in which the control logic, loop control and the control laws are designed is not always done in a logical order. This is due to the desired information that is not present at the right moment. To test the control logic, input from the event-based plant must be available but this is not always the case. A mechanism must be available to test the control logic while the discrete-time signals from the plant are not available yet.

A mechanism that has been proposed and implemented in this research is the *dummy plant*. This plant serves as an initial implementation to receive control signals and feedback signals to the control logic. These plants also come with an initial loop controller that steers the plant between -1 and 1. This setpoint range is chosen because of convenience. When in a later stage more information is available on the continuous-time plant, the parameters in the dummy plant can be changed to the real values. The control law parameters can at that moment also be changed to the tuned parameters. The idea of the dummy plant is that the plant is as simple as possible with minimal amount of dynamics. It serves as an temporary plant implementation. In Figure 3.5 a proposition is made for the dummy plant that is used in the case study. There is chosen for a simple first-order model of the plant that is controlled by a simple proportional controller. The system model is chosen with low bandwith such that the movement can easily be verified in simulation results.

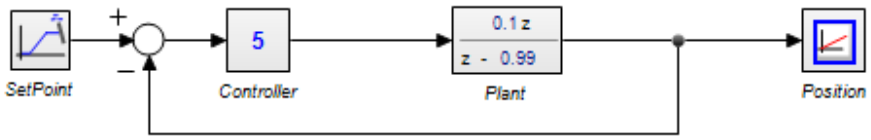


Figure 3.5: The proposed dummy plant, the controller is a simple proportional controller and the plant is a simple first order system. The setpoint range that is used is between -1 and 1. This dummy plant is used in the case study.

With this dummy plant proposal, the discrete-time plants are present in different complexities throughout the design. It is first present as an temporary implementation and is later adapted to the real plant. These levels of complexities can be used as different levels of detail in the application of the SDM.

Network control

When the CPS has distributed computing devices the time for communication has to be taken into account. The processing time has influence on the time constraints that are present. For proper operation of the control loops the time constraints must be met.

The interfaces between the components in the ECS and the network control must also be extensible such that new functionality can be integrated easily. The output of the network control to the outside world can also be used for communication with the I/O devices and the plant. This makes the ECS better interchangeable between the virtual plant and the real plant.

Requirement derivation

The requirements that have been derived from these challenges are stated below.

1. The ECS architecture shall support extensibility, this supports easy implementation of new modules as is done in the SDM.
2. The ECS architecture shall support interchangeability, this supports easy switching between different type of simulations.

3. The ECS architecture shall support a mechanism for easy integration of discrete-event signals and discrete-time signals. This supports the integration of discrete-event control logic and discrete-time loop control.

3.3 Application of requirements on case study

The requirements which have been identified from the stakeholders can be used to assess the SDM method when it has been applied to the design of ECS for the production-cell setup.

The traceability and applicability of the SDM are assesed during evaluation of the method. The design error detection mechanism of the SDM and the ability to use the method in an easy way are also assessed during the evaluation of the method.

Challenges that are identified during application of the SDM are taken into account during the execution of the SDM on the production-cell setup. In this way it is tried to overcome issues and to have a smooth implementation and integration of the functionality.

In the evaluation of the SDM, the requirements that have been identified on the application of the SDM are used to assess the performance of the method.

4 Case study: Production-cell demo setup

In this chapter the case study is described that is performed on the production-cell setup. It is used to test the Structured Design Method on a plant containing discrete-event signals.

In Section 4.1 an overview is given on the production-cell setup. In Section 4.2 the Structured Design Method is started with the preparation phase on the design of the Electronic Control Software. In Section 4.3 the design cycles are started and the design is implemented and tested. In Section 4.4 an evaluation and reflection is done on the design process. During the execution of the Structured Design Method the steps in Figure 1.1 are used.

4.1 Overview of the production-cell

The production-cell setup is a simplified representation of an injection molding machine and is shown in Figure 4.1. Blocks are used to represent molded parts which are first transported by a Feeder Belt to the Feeder. Then the blocks are fed into the Molder by the Feeder. The next step is to extract the blocks from the Molder by the Extractor onto the Extractor Belt, which transport the blocks away from the Extractor. To make the setup run continuously, the blocks are picked up from the Extractor Belt and put onto the Feeder Belt by a rotator mechanism. In Figure 4.2 a model of the production-cell setup is shown where the black arrows show the direction of movements of the blocks and the different mechanisms are indicated.

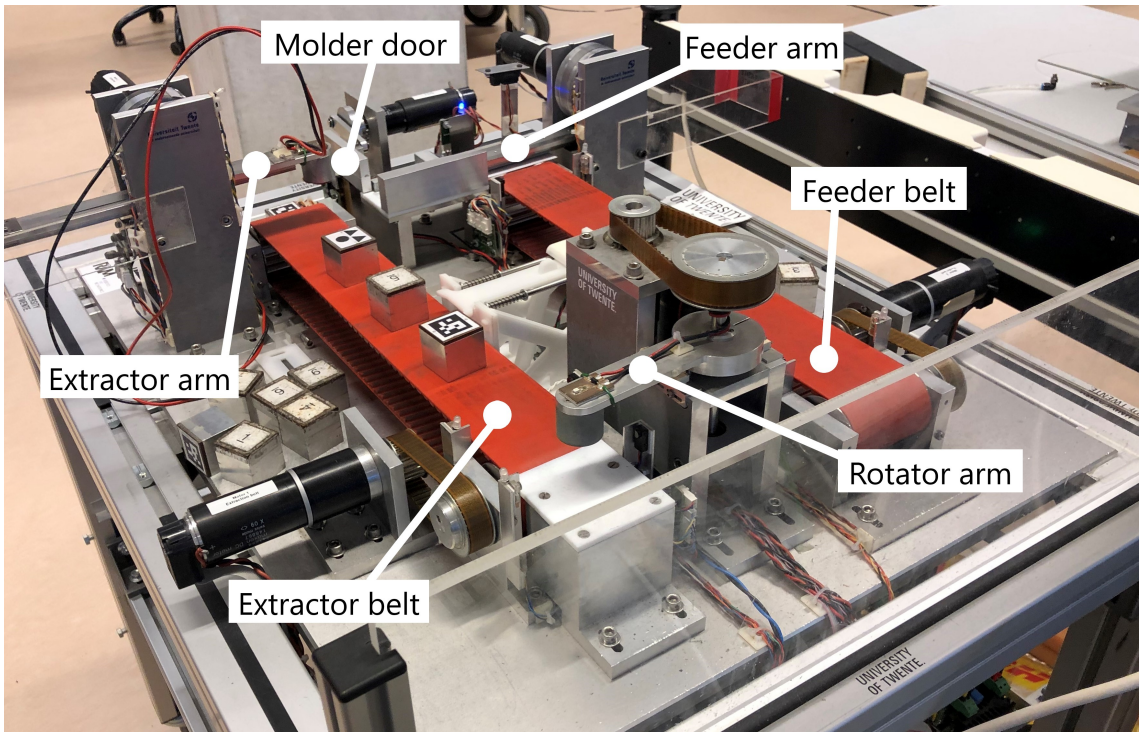


Figure 4.1: Production-cell setup that is used as a case study for the Structured Design Method, the different subsystems are indicated

In Figure 4.2 sensors can be seen (denoted Sx) which are used to detect blocks on certain positions (denoted Px). There are six electric motors present which are used to operate the mechanical systems. Each of these motors have an encoder for feedback purposes and some subsystems also have end switches which can be used for homing purposes (denoted ESx).

The embedded computer boards are two RaMstix boards. This RaMstix board is an in-house developed ECS board which contains a Gumstix Overo computer (Gumstix, 2020). The Em-

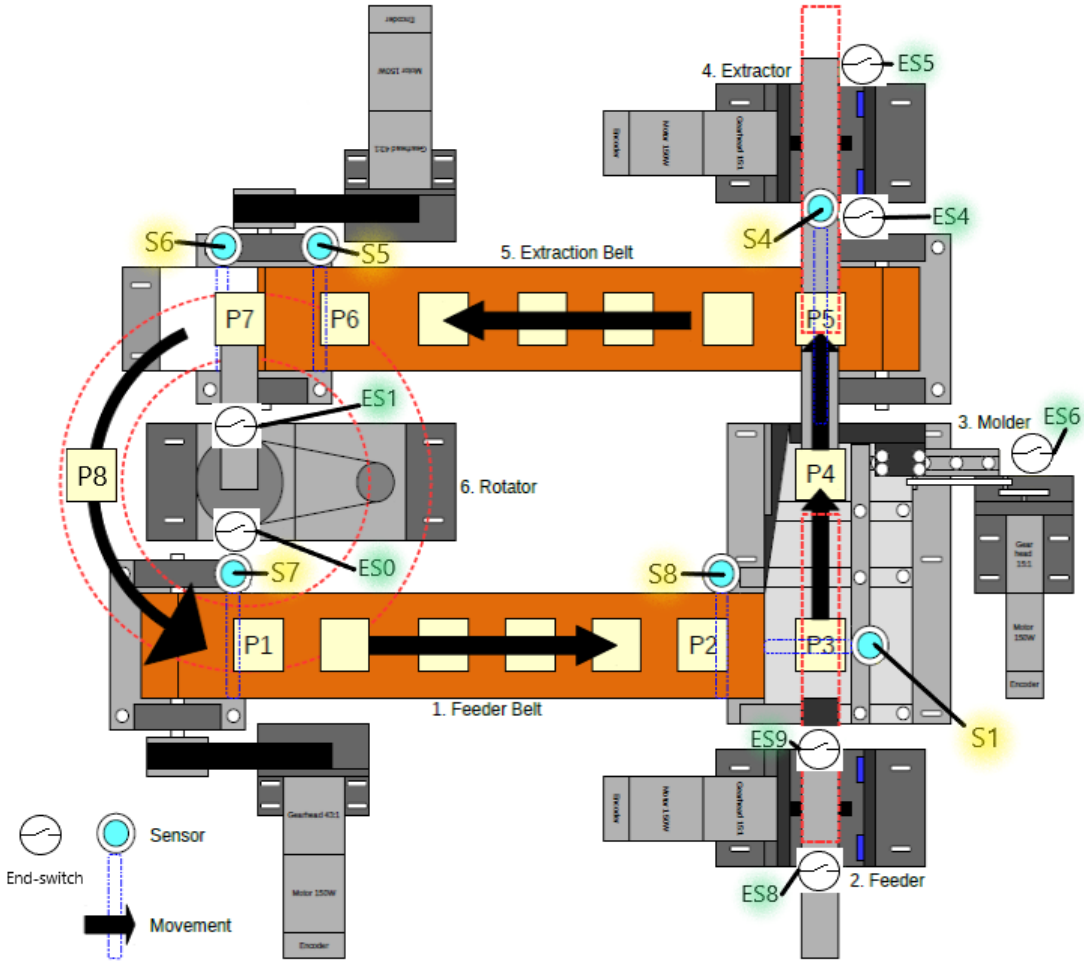


Figure 4.2: Model of the production-cell, this figure has been adapted from (van den Berg, 2006). The black arrows indicate the directions of movement of the blocks, the circles indicated with Sx are the block detection sensors and the circles indicated with ESx are the end switches that are used for homing. The squares inidicated with Px are positions that are monitored by block detection sensors or have another function in the ECS. The six subsystems of the setup are indicated by their name.

bedded Computer runs Linux and has an FPGA that is connected to a subset of the pheriph-erals of the production-cell. The Embedded Computer and FPGA communicate via a General Purpose Memory Controller (GPMC). The RaMstix boards communicate via Ethernet to each other. A Linux operated PC is present to control the RaMstix boards and runs a GUI for control of the complete setup. In Figure 4.3 a simplified overview is shown of the embedded computer boards.

The production-cell setup has been developed in (van den Berg, 2006), the setup has been build to test and demonstrate implementations of new communication and control algorithms. The setup has been used to test a model-driven design approach with new tool-chains (van de Rid-der, 2018). Also a Finite State Machine based controller has been proposed, implemented and tested on the production-cell (Meijer, 2013). Some of the ideas and implementations have been used as reference for the design of the ECS during this case study.

Across the execution of the SDM, references are made to positions, sensors and end-switches. When these are mentioned, references are ment to parts in Figure 4.2.

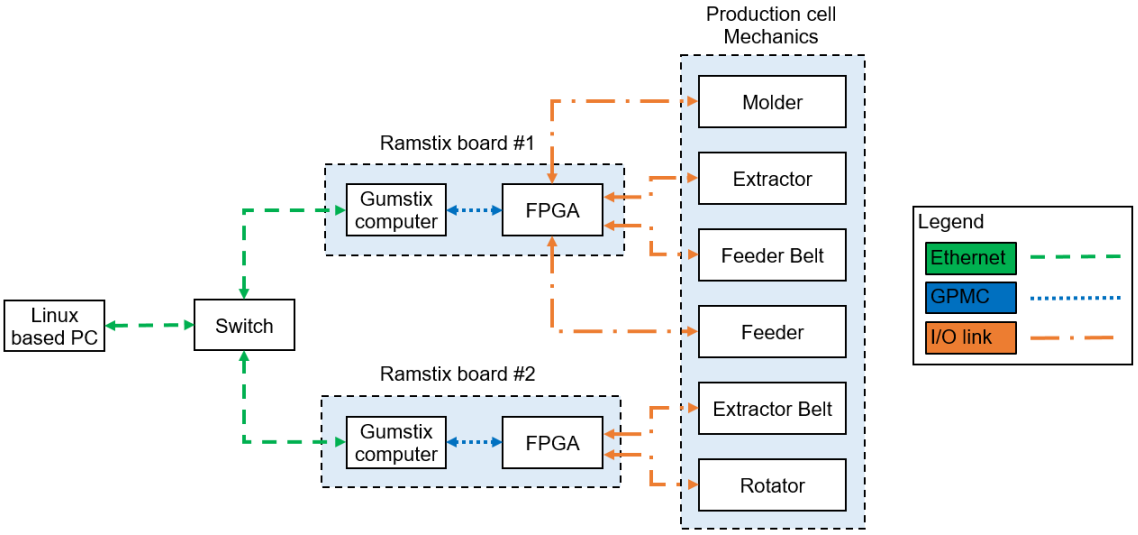


Figure 4.3: Simplified overview of the electrical installation of the production-cell. The production-cell can be controlled using the GUI on the Linux based PC. The PC communicates with the two RaMstix boards. Both of the RaMstix boards are connected to a subset of the pheripherals of the production-cell.

4.2 Preparation phase

During the preparation phase of the Structured Design Method, a functional analysis is performed to identify the functionalities and features that need to be implemented into the Electronic Control Software. This is done in Section 4.2.1. In Section 4.2.2 the levels of detail are determined that are used to implement the features.

When the functionality and features are known, requirements can be derived that describe these aspects in a formal way. These requirements are used during testing to verify that the desired functionality has been implemented. The requirements of the different subsystems are derived in Section 4.2.3.

From the functionality in the production-cell, Finite State Machines can be derived that can be implemented in the ECS. The Finite State Machines are derived in Section 4.2.4. In Section 4.2.5 the initialization algorithms for the features are described.

To test the implemented control logic in the ECS, a plant must be available. The functionalities that the different parts of the event-based plant need are derived in Section 4.2.6.

In Section 4.2.8 the simulation framework that has been used is described and in Section 4.2.9 the architecture of the ECS is described. In Section 4.2.10 a conclusion is given on the preparation phase.

The focus of this case study is on the discrete-event signals and the combination of discrete-time and discrete-event signals. The continuous-time plants are therefore of less interest. The SDM will therefore not be used to design and implement competent models of the the continuous-time plants and control laws.

4.2.1 Functional analysis

The functionality of the overall production-cell is divided into six subsystems. These subsystems are considered as the features that are implemented during the execution of the SDM. With these features it is possible to divide the features into levels of detail. The identified functionalities and features are stated in Figure 4.4.

The functionality per subsystem, or feature, is stated below:

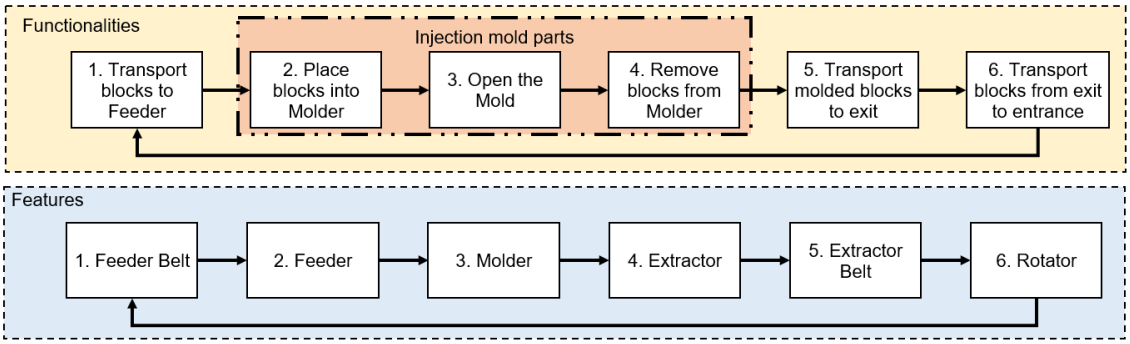


Figure 4.4: Identified functionalities of the production-cell, together with the derived features that are implemented during the execution of the SDM. Using these features it is posible to divide the features into levels of detail.

1. **Feeder Belt**
   New blocks are put into the system on the Feeder Belt. The Feeder Belt transports the blocks to the Feeder. At the end of the Feeder Belt is a stationary plateau where the blocks are pushed on by the Feeder Belt. On the Feeder Belt are two sensors present that detect blocks and make sure that the block on the plateau and the block on the belt do not collide with each other.

2. **Feeder**
   When a block is ready on the plateau at the end of the Feeder Belt, the Feeder can push this block into the Molder. The Feeder consist of a translational pusher arm that translates along one axis and connects the Feeder Belt to the Molder.

3. **Molder**
   The Molder consist of a door that resembles the mold of the injection molding machine. When a block is pushed in the Molder by the Feeder the door opens to resemble that the molding is finished.

4. **Extractor**
   When the molding is performed, the Molder door opens and the block can be removed. This action is performed by the Extractor. This mechanism picks up the block in the Molder and puts the block on the Extractor Belt. The Extractor consist of a translating arm to which a magnet is attached to pick up the block. The Extractor connects the Molder and the Extractor Belt. Block detection sensors are present to detect blocks at the Extractor Belt and make sure that the blocks do not collide.

5. **Extractor Belt**
   The Extractor Belt transports the blocks from the Extractor to the Rotator. At the end of the belt is a stationary plateau where the Rotator picks up the blocks. At the Extractor Belt are two sensors present to detect the block on the plateau and the blocks on the belt and make sure that the blocks do not collide.

6. **Rotator**
   When a block is present at the plateau at the end of the Extractor Belt, the Rotator can pick up this block. The Rotator moves the block to the start of the Feeder Belt. A block detection sensor is present at the start of the Feeder Belt to make sure that the block on the Rotator and the block at the start of the Feeder Belt do not collide. The Rotator is a mechanism with a rotating arm with a magnet attached to it that can pick up and move the block. The Rotator connects the end of the Extractor Belt with the start of the Feeder Belt to make the setup run continuously.

The order of implementation of the different features will be the same as the order in which blocks are processed by the production-cell, so starting from the Feeder Belt and ending at the Rotator. In this way the ECS is build up step by step and the new functionality can be tested together with already implemented parts of the ECS from earlier design cycles in the SDM.

Now that the features are determined, namely Feeder Belt, Feeder, Molder, Extractor, Extractor Belt and Rotator, the level of detail that has to be implemented per feature can be determined.

4.2.2 Levels of detail

In this section the levels of detail that are used in the features are described. These levels of detail have been determined because it gives the oppertunity to test the implemented behavior in small steps. The paradigm that is used is from (Broenink et al., 2016). The levels of detail that are defined are stated below.

1. **Control logic implementation**
   Parts of the control logic for the feature will be implemented and tested. At this stage the continuous-time plants, which provide the ECS with encoder values, are not implemented yet and certain control decisions that use these encoder values cannot be implemented and tested at this level of detail. To handle this abscense of discrete-time plants, the event-based plant is used. The event-based plant is implemented in a way that movement of blocks is atomic and that block positions are discrete, time is considered to be a unitless step. This is done because there are no dynamic systems involved in this level of detail which does depend on a fixed time step. This implementation provides enough level of detail to test the discrete-event control logic.

2. **Discrete-time implementation**
   When the control logic has been implemented, the discrete-event plants can be converted to discrete-time. This prepares the plant and controller for the implementation of the discrete-time plants and loop controllers. The unitless time step that is used in the first level of detail is converted to a real step in time. The discrete-event control logic can be tested to see if the behavior is still correct with this real timestep.

3. **Implementation of plant**
   In this design cycle the dummy plants, that have been propsed in Section 3.2, are implemented and the encoder values for the control decisions become available. This means that the remaining functionality, that depend on signals from the discrete-time plant, of the control logic for all the features can be implemented and tested. At this point the complete controller has been implemented, only with dummy plants instead of the real plant and control loop implementation. This dummy plant serves as an temporary implementation of the discrete-time plant because the systems of the plants and the tuned loop controllers are assumed to be not known at this level of detail.

4. **Initialization algorithms**
   Now that the discrete-event control logic and the loop controllers are implemented, the initialization algorithms can be added. In this step also the end switches, which are used for initialization are implemented.

5. **Real plant implementation**
   In this design cycle the parameters of the dummy plants and their controllers are changed to the models of the real plants and the controllers. Dimensions of the transportation belts and the positions where the sensors detect a block are adapted to the real plant.

6. **Production-cell implementation**
   The last step is to implement the ECS on the target device to control the real plant instead of the virtual plant. To do this, extra software has been added to retrieve the plant values from the production-cell and to write values to the production-cell. Because the production-cell has multiple controllers which communicate via a Ethernet, this communication is implemented and tested first on a test platform before it is implemented in the ECS of the production-cell.

The order of implementation for the levels of detail that will be used will deviate from the order that is discussed in Figure 1.1. In the SDM the levels of detail are implemented one after another before moving to the next feature. In the application to the production-cell one level of detail will first be implemented for multiple features before moving to the next level of detail. This is done because there are dependencies between features instead of level of detail. So there is iterated horizontally instead of vertically. The proposed implementation order is shown in Table 4.1.

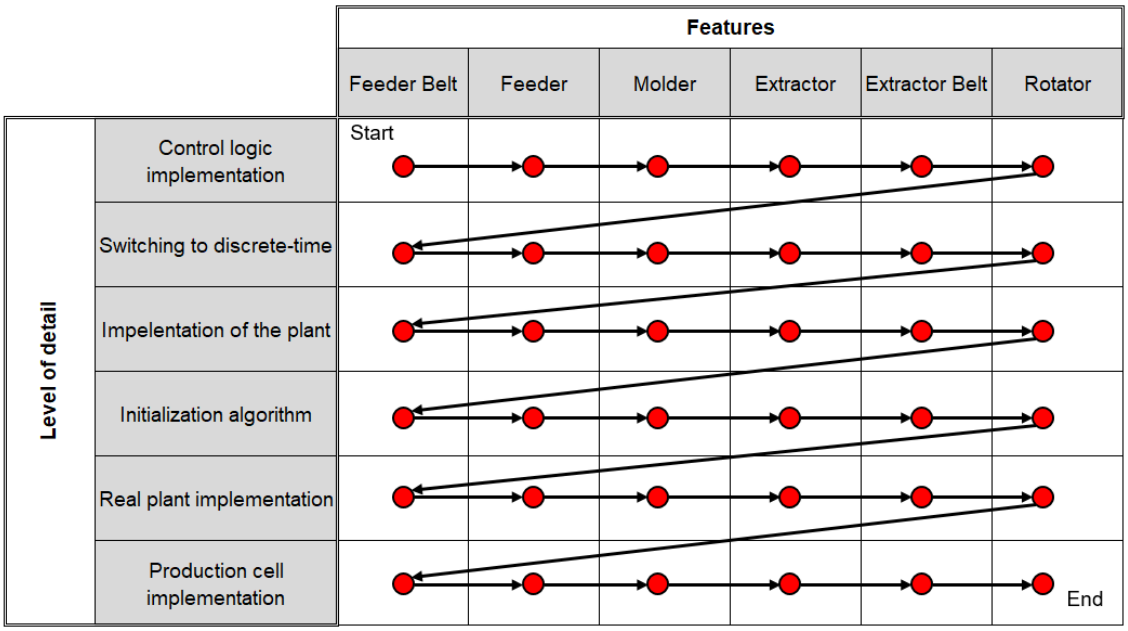


Table 4.1: Propsed implementation order of features and levels of detail, where the order deviates from the standard order. One level of detail is first implemented for all features before there is moved to the next level of detail.

4.2.3 Requirement derivation

From the functional analysis the requirements are derived. The requirements are stated according to the EARS method (Mavin and Wilkinson, 2010) and are stated per feature. The focus of this report is on the discrete-event part, therefore only the requirements for the discrete-events are stated.

The derived requirements and designed Finite State Machines are the initial design and are implemented and tested during the execution of the design cycles. The design can still change if the developed design turns out to be incorrect.

1. **Feeder belt**
   (a) While the Feeder Belt is in initialization, the Feeder Belt shall be at a halt
   (b) While the Feeder Belt is running, the Feeder Belt shall run at a constant velocity

(c) While the Feeder Belt is stopped, the Feeder Belt shall be at a halt
(d) When a block is present at position P2 and position P3, the Feeder Belt shall stop, otherwise the Feeder Belt shall run

2. **Feeder**
   (a) While the Feeder is in initialization, the Feeder shall follow an initialization algorithm
   (b) When a block is present at position P3, there is no block at position P4 and the Molder door is closed, the Feeder shall move the block at position P3 to position P4
3. **Molder**
   (a) While the Molder is in initialization, the Molder door shall follow an initialization algorithm
   (b) When a block is present at position P4, the Feeder is not feeding in a block and the Extractor is not extracting a block, the Molder door shall open
   (c) When a block has been removed by the Extractor and the Molder door is open, the Molder door shall close
4. **Extractor**
   (a) While the Extractor is in initialization, the Extractor shall follow an initialization algorithm
   (b) When a block is present at position P4, the Molder door is open and there is no block at position P5, the Extractor shall extract the block from position P4 to position P5
5. **Extractor Belt**
   (a) While the Extractor Belt is in initialization, the Extractor Belt shall be at a halt
   (b) While the Extractor Belt is running, the Extractor Belt shall run at a constant velocity
   (c) While the Extractor Belt is stopped, the Extractor Belt shall be at a halt
   (d) When a block is present at position P6 and at position P7, the Extractor Belt shall stop, otherwise the Extractor Belt shall run
6. **Rotator**
   (a) While the Rotator is in initialization, the Rotator shall follow an initialization algorithm
   (b) When a block is at position P7 the Rotator shall pickup the block
   (c) When the Rotator has picked up a block and there is no block present at position P1, the Rotator shall move the block from position P7 to position P1
   (d) When the Rotator has picked up a block and there is a block present at position P1, the Rotator shall move the block from position P7 to position P8
   (e) When the Rotator is at position P8, the Rotator shall wait untill the block at position P1 is removed by the Feeder belt and then move to position P1

4.2.4 Finite State Machines

From the requirements which have been derived in Section 4.2.3, Finite State Machine are developed. These FSM's are implemented in the ECS.

The initial state is indicated by a black circle. The arguments in the blue squares are actions that are taken when that state is reached. The arguments in the white squares are the conditions that must be true to proceed to the next state. The bold case arguments in the white squares are variables that come from another features and are external variables, the non-bold case arguments in the white squares are internal variables.

Feeder Belt

The functionality of the Feeder Belt has been divided into two states, these are RUNNING and STOPPED. When blocks are detected at position P2 and position P3 the transport belt has to stop. When only one or none of the sensors at these positions detect a block the belt can start transporting blocks again. The Finite State Machine of the Feeder Belt is shown in Figure 4.5.

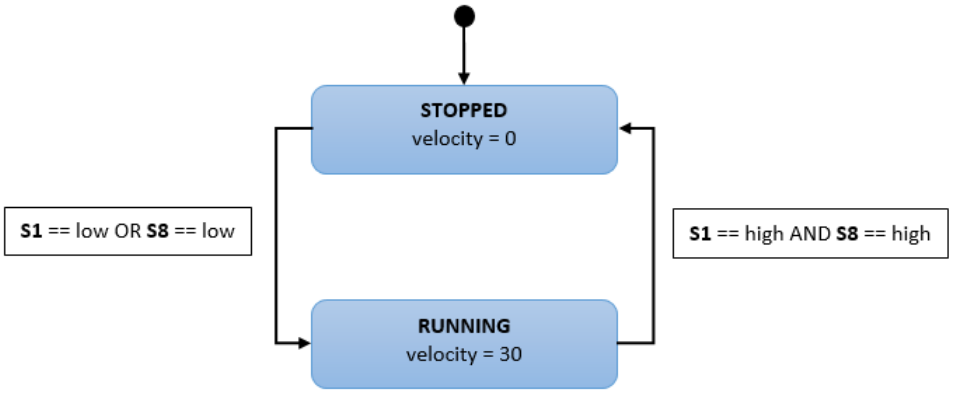


Figure 4.5: Finite State Machine for the Feeder belt. The belt runs at a constant velocity and when blocks are detected at position P2 and P3 the belt stops rotating. When the block at position P3 is removed the belt starts moving again.

Feeder

Functionality of the Feeder is divided into three states. The initial state is WAITING_FOR_BLOCK which checks position P3 if a block is present to be pushed to the Molder. If a block is present and the Molder door is closed then the state changes to FEEDING and the Feeder moves forward. When that position has been reached the Feeder state changes to FEEDER_RETRACTING and moves the arm back and then the state changes to WAITING_FOR_BLOCK. The Finite State Machine of the Feeder is shown in Figure 4.6.

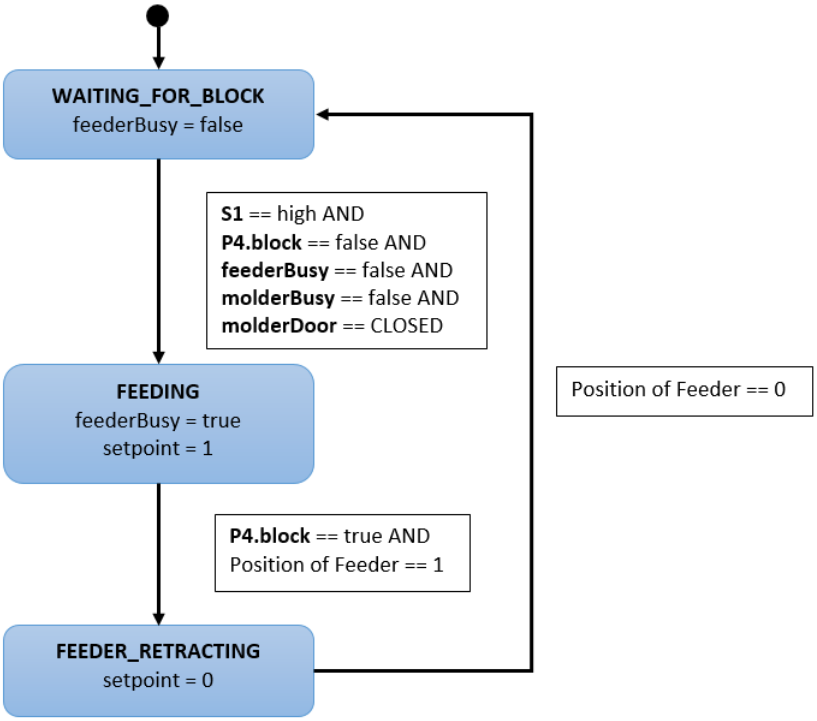


Figure 4.6: Finite State Machine for the Feeder. When a block is detected at position P3 and all the conditions are met, the Feeder arm pushes a block from position P3 to position P4. After that the arm is retracted to the initial position.

Molder

The Molder door functionality has been divided into four states. These are OPEN, CLOSED, OPENING and CLOSING. The initial state is OPEN, when the conditions are met the state is changed to CLOSING and the door moves to the closed position. When the door reaches the setpoint which corresponds to the closed position the state changes to CLOSED. The same holds for opening the door, if the conditions are met the state is changed to OPENING. When the open position has been reached the state changes to OPEN. The Finite State Machine of the Molder door is shown in Figure 4.7.

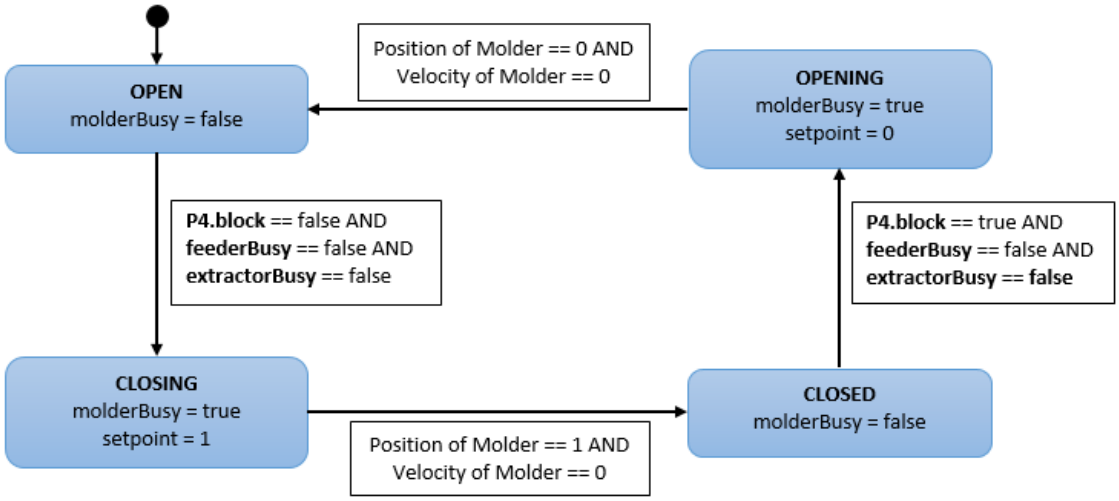


Figure 4.7: Finite State Machine for the Molder. The initial position of the Molder door is open. When the conditions are met the door starts moving to the closed position. When that position is reached the state changes to closed. The same holds for the reverse action, opening the door.

Extractor

For the Extractor, the functionality is divided into five states. The initial state is WAITING_FOR_BLOCK, here the extractor arm waits untill a block is pushed onto position P4 by the Feeder and the Molder door has been opened. Then the state is changed to MOVE_TO_BLOCK and when that position has been reached the magnet is turned on and a fixed time is waited to let the block stick well to the magnet. When the timer is finished the state is changed to RETRACT_BLOCK and the arm moves back. When that position has been reached the magnet is turned off and the block is dropped at position P5 and a fixed time is waited. When the timer has been finished the state is changed to WAITING_FOR_BLOCK to process the next block. The Finite State Machine of the Extractor is shown in Figure 4.8.

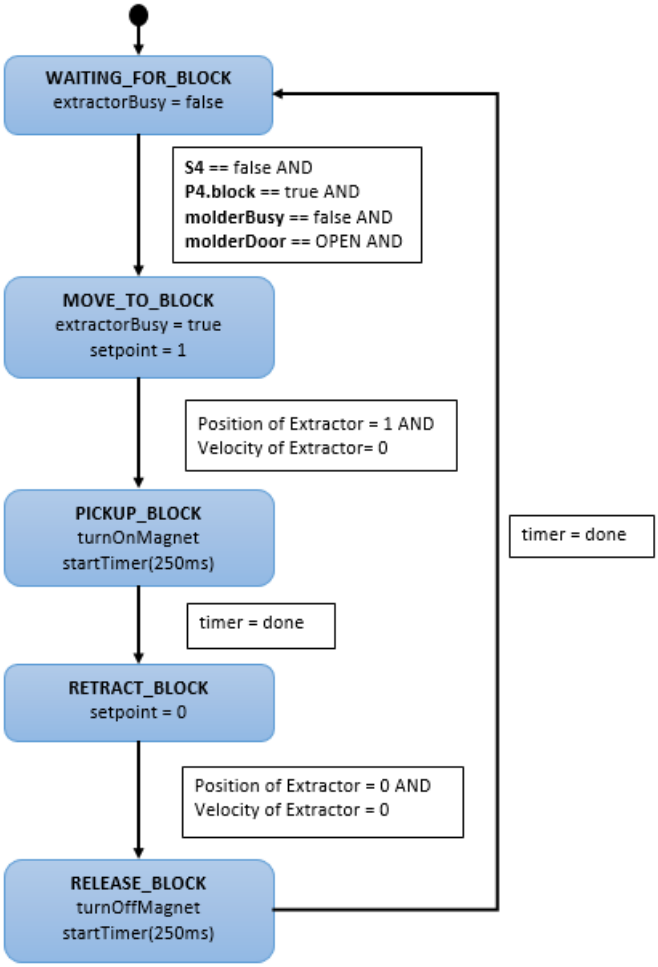


Figure 4.8: Finite State Machine for the Extractor. The complete motion of the Extractor has been divided into five states. When the conditions are met to remove the block from position P4, the arm moves forward and picks up the block with the magnet. Then the arm moves back to the drop location and the magnet is turned off. Then the motion is complete.

Extractor Belt

The Extractor Belt functionality has also been divided into states RUNNING and STOPPED. The belt runs always untill sensor S5 and sensor S6 detect a block. Then the state is changed to STOPPED, when one of the sensors does not detect a block anymore the state is changed to RUNNING again. The Finite State Machine of the Extractor Belt is shown in Figure 4.9.

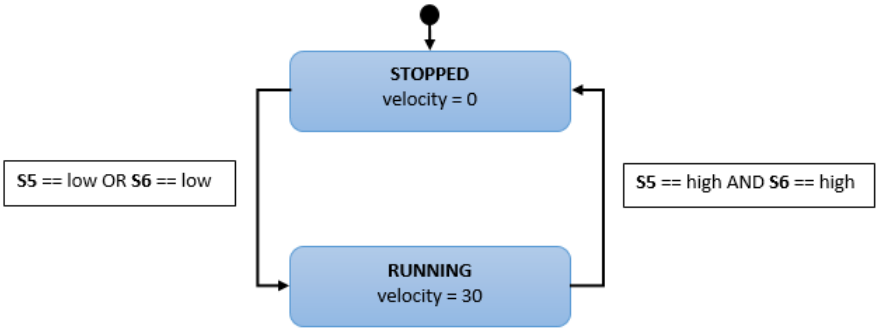


Figure 4.9: Finite State machine for the Extractor belt. The belt runs at a constant velocity and when blocks are detected at position P7 and P6 the belt stops rotating. When the block at position P7 is removed the belt starts moving again.

Rotator

The functionality of the Rotator has been divided into six states. The initial state is WAITING_FOR_BLOCK, when a block is detected by sensor S6 the state is changed to PICKUP_BLOCK. Then the magnet of the Rotator is turned on and there is waited a fixed time using a timer to make the block stick well to the magnet before the block is moved. After that sensor S7 is checked if a block is present at position P1. If a block is present at position P1, the next state is ROTATE_90_DEGREE and the Rotator arm rotates to position P8. This state checks sensor S7 untill no block is present anymore at position P1 and then goes to the state ROTATE_180_DEGREE. If no block is detected by sensor S7 after the block has been picked up from position P7, the next state is ROTATE_180_DEGREE. When the rotator arm arrives at position P1 the magnet is turned off and the block is released. When the block is released the state changes to ROTATE_BACK and the arm moves to the initial position, above position P7. Then the state changes to the initial state, WAITING_FOR_BLOCK. The Finite State Machine of the Rotator is shown in Figure 4.10.

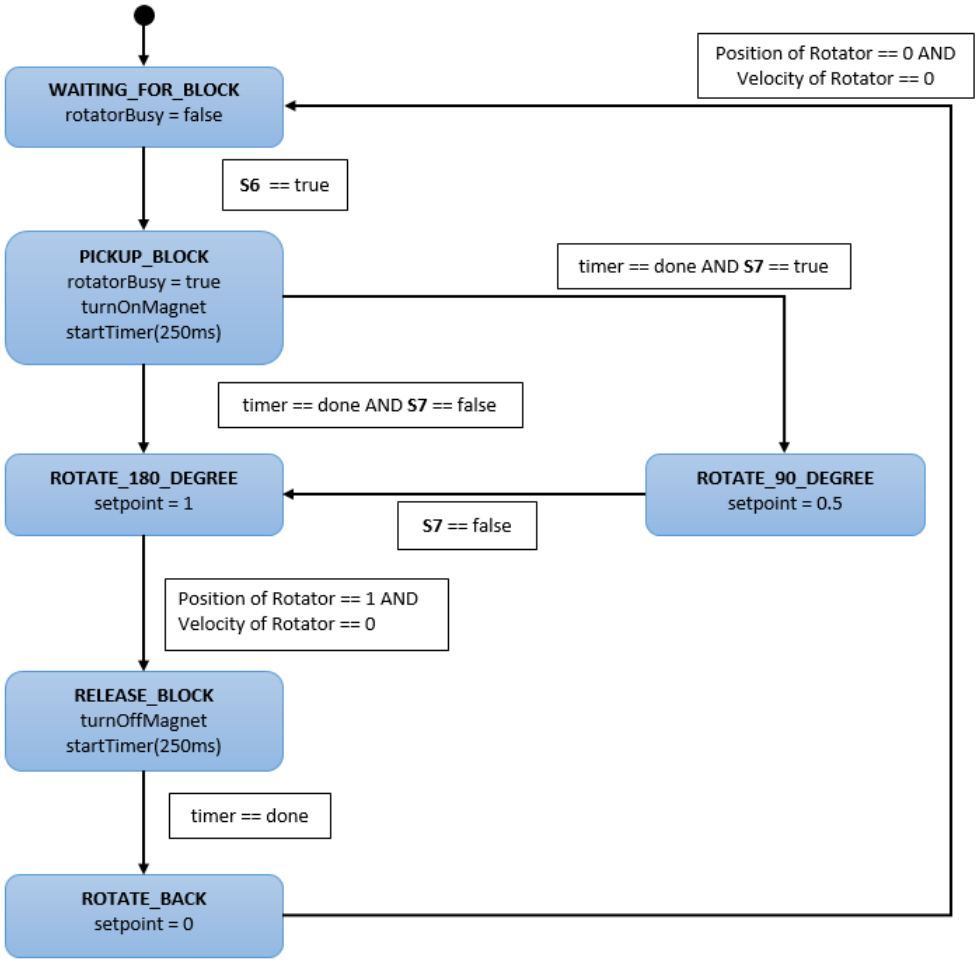


Figure 4.10: Finite State Machine for the Rotator. When a block is detected by sensor S6 the block is picked up by the Rotator magnet. To make the block stick well to the mangnet, a fixed time is waited using a timer. Then the drop location is checked using sensor S7. When a block is present the block is moved to position P8 to wait, otherwise the block is moved to position P1. When position P1 is reached the block is dropped and the Rotator arm moves back to the initial position.

4.2.5 Initialization algorithms

The initialization algorithm is used in the Feeder, the Molder door, the Extractor and the Rotator. When the production-cell is turned on the actual positions of the mechanisms is not known due to the usage of relative encoders. Therefore end switches are used which are present at fixed positions. Relative to these positions, encoder values have been determined of the positions that are used as setpoints, and these can be used for the different subsystems.

The initialization algorithm works by moving the systems with a constant control signal in the desired direction. When the position of the systems reaches the position of the end-switch a signal is given by the switch to the ECS. At that point the motion is stopped and the current position of the system is known. Then the encoder position is set to zero and new motions are performed relative to this new zero position. Then the initialization is complete.

The Feeder Belt and Extractor Belt are at a halt during initialization, because their position is not of interest and does not need any initialization. These systems are velocity controlled instead of position controlled.

4.2.6 Discrete-event plant models

In this section the functionalities that have to be present in the discrete-event plant models are described. These functionalities need to be present to be able to test the implemented functionalities in the ECS. For the implementation of the discrete-event plant models the event-based plant is used. The functionalities that are present in the different plant models are described below.

1. **Feeder Belt**
   (a) Move the blocks on the belt depending on the velocity of the belt.
   (b) Control the sensor ouputs if the blocks are in a certain range of the belts. Do this for sensors at Position P1, P2 and P3.
   (c) Put blocks on the belt if there is free space on the belt.
   (d) Remove the blocks from the belt if the Feeder is moving the blocks from the Feeder Belt.
   (e) A mechanism is put into place that checks if the blocks collide.
2. **Feeder**
   (a) When a block is moved by the Feeder, the block is moved from position P3 to a temporary position to mimic that the sensor at Position P3 does not detect the block anymore.
   (b) When the Feeder reaches the Encoder position that corresponds to Position P4, the block is moved from the temporary position to Position P4.
3. **Extractor**
   (a) When a block is moved by the Extractor, the block is moved from positoin P4 to a temporary position.
   (b) When the Extractor reaches the encoder position that corresponds to Position P5, the block is put from the temporary position onto the Extractor Belt.
4. **Extractor Belt**
   (a) Move the blocks on the belt, depending on the velocity of the belt.
   (b) A mechanism is put into place that checks if the blocks collide on the belt.
   (c) Control the sensor ouputs if the blocks are in a certain range of the belt, do this for sensors at Positions P5, P6 and P7.
   (d) Remove a block from the belt if the Rotator has picked up a block.
5. **Rotator**
   (a) When a block is moved by the Rotator, the block is moved from the Extractor Belt to a temporary position.
   (b) When the Rotator reaches the encoder position that corresponds to Position P1, the block is moved from the temporary position to the begin of the Feeder Belt.

The event-based plant are functions that act on blocks in the production-cell. An example is given for the Feeder Belt on how the architecture of this plant operates. The functionalities are parallel systems that perform actions on the blocks or extract information from the blocks, such as the position on the belt. This overview can be seen in Figure 4.11. The *move block* component for example increments the position based on the velocity of the plant. In a low level of detail this block receives a *run* or *stop* command and in a higher level of detail this block receives a real valued number from the dynamic system of the motor.

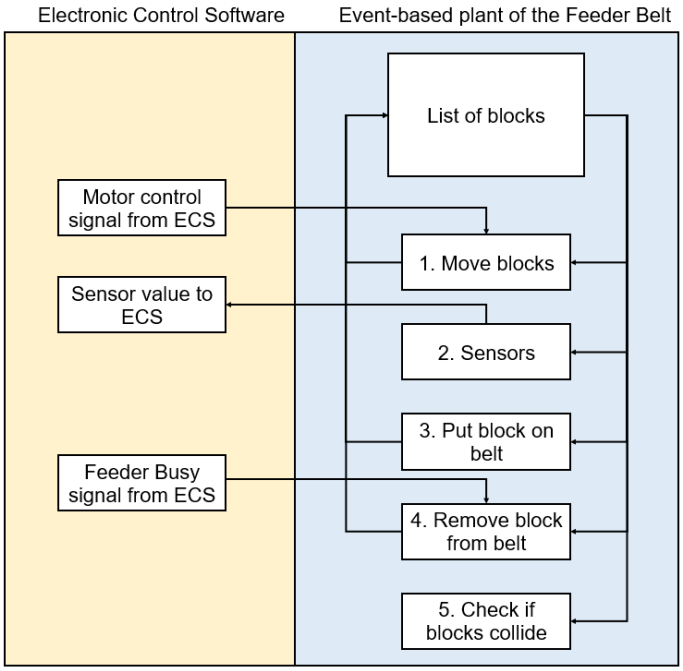


Figure 4.11: Overview of the event-based plant implementation of the Feeder Belt. The list of blocks holds all the blocks that are present on the belt. The different function blocks check all the blocks in the list and perform an action on them based on their function.

4.2.7 Discrete-time plant models

For all the subsystems, the continuous-time plants are converted using the Z-transform to discrete-time plants. The plants run at the same rate as the controller for convenience. The input, or control signal, comes from the ECS and steers the systems. The encoder values are send back tot he ECS for feedback purposes.

The output of the end switches that are present on the Feeder, Molder, Extractor and Rotator are controlled to a high state when the subsystems are inside a certain position range. In this way they can be used for initialization purposes.

4.2.8 Simulation framework

For the implementation of the plant and the controller the C++ programing language is used. These C++ programs can run on the target device and on the development computer. In this way the code that is designed and tested is also used on the target device and no conversion is necessary but only cross-compilation. Other advantages are that there is more freedom in how the implementation of the plant and the controller can be done. Disadvantages are that all components of the plant, controller and the communication between the boards and the PC have to be implemented.

A framework has been designed which hold a simulator entity, the simulator holds the controller and the plant that can communicate with each other via pointers inside the simulator. During the simulation the obtained data is saved into a CSV-file and can, after the simulation be visualized using a Python script. In Figure 4.12 a single run is shown of the simulator as it is used in the first design iteration. The simulator first executes the Finite State Machines of the different components that are present and then executes the plant and eventually saves all the obtained data in that iteration.

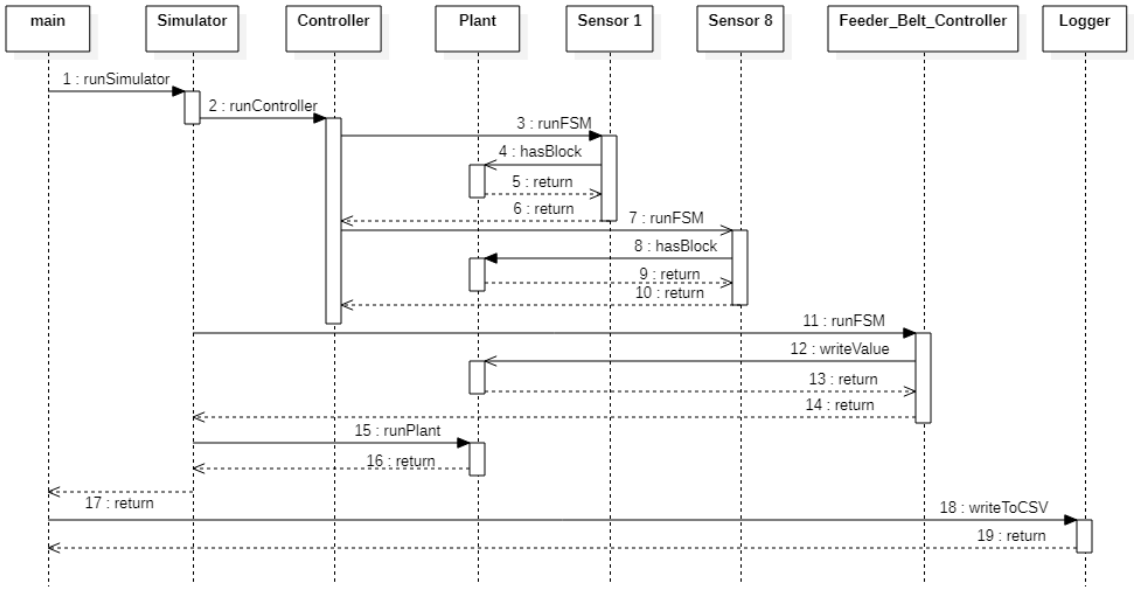


Figure 4.12: Single time step of the simulator, this run comes from the implementation of the Feeder Belt in the first iteration. The simulator runs the controller and the plant. In the controller all the Finite State Machines of the present components are run and the output of these FSM's is used in the plant. As a last step all the obtained data is stored into a CSV-file.

To support the traceability of the method, a form is designed where the design cycles are logged. This form is shown in Appendix C. The columns that are present describe the design cycles and the applicable documents and models in a structured way. This information is of interest when certain implementations must be analyzed at a different moment in time.

4.2.9 ECS architecture

The ECS framework has been setup such that it is easily extensible with new modules that are added in every design cycle.

Communication between the plant and the controller is done in the simulator entity, where information is exchanged. In the controller the block detection sensors have pointers to the different positions in the plant and detect the presence of a block via these pointers.

In Figure 4.13 the a UML diagram is shown that is developed and has been used up to the implementation on the real production-cell (design cycle 5). It is shown that for every feature a seperate Class is made, in this Class the Finite State Machine of that feature is implemented.

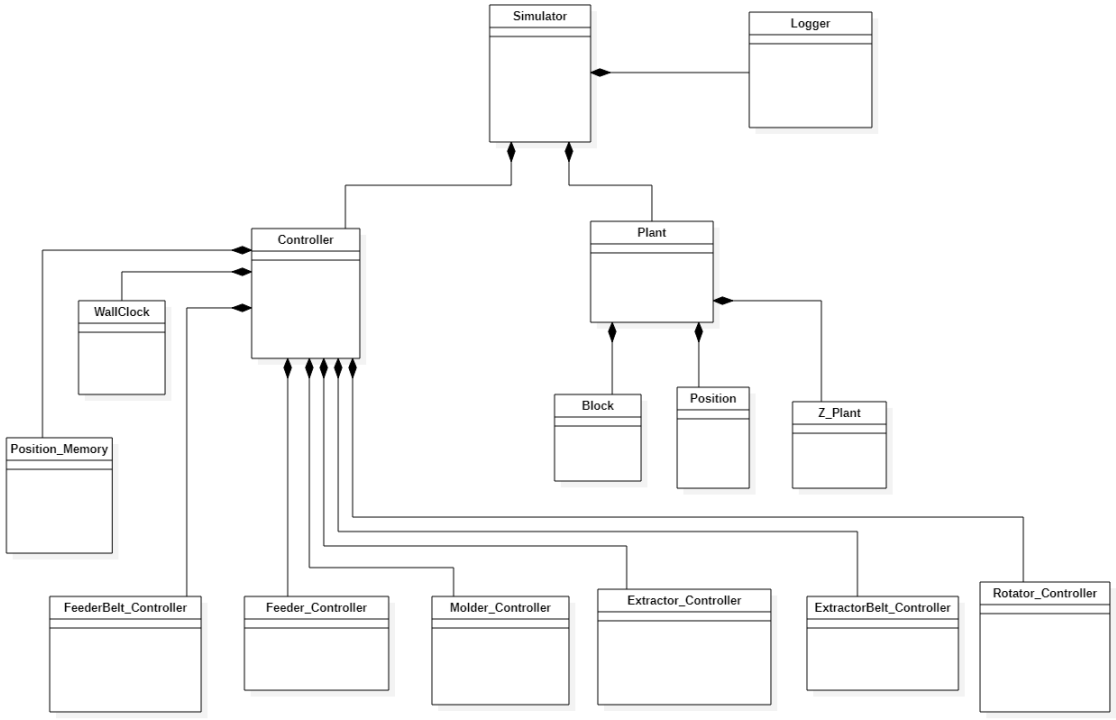


Figure 4.13: UML diagram of the ECS which is implemented and used up to design cycle 5.

4.2.10 Conclusion on preparation phase

In the preparation phase the features and levels of detail for the execution of the design cycles have been determined. Also necessary functionality for the event-based plant and the control logic has been determined. Now all information is available to start the design cycles.

During the preparation phase it has not been assumed that models were available. This is different in the application of the SDM to the Segway robot where this was assumed. Therefore the preparation phase was more extensive because the models and functionality are designed in this case.

4.3 Design cycles

4.3.1 Level of detail 1: Control logic implementation

Step 1:

In the first design cycle the simulation framework is implemented, together with the control logic, the unitless time steps of the event-based plant and atomic block movement to test the ECS. The plant operate with blocks that are on positions, for example the transport belts consist of multiple positions and the blocks are moved to the next position if the belt is moving. In this way the logic can be tested. In Figure 4.14 a overview is given of the discrete positions that have been implemented and are used in this level of detail.

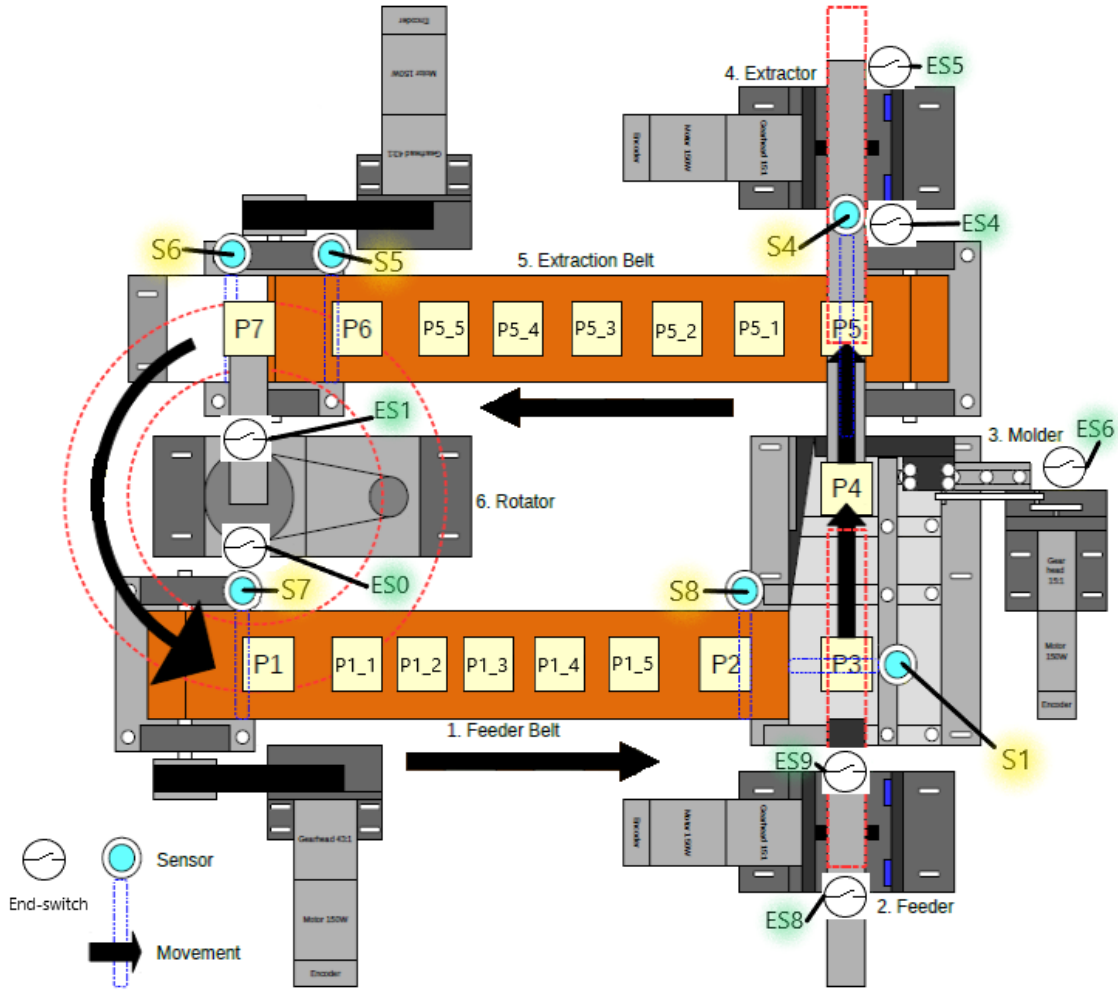


Figure 4.14: This model shows the positions that have been implemented in this design cycle and that are used in the simulations of the different systems.

The requirements and Finite State Machines that have been developed will be used to check if the desired behavior has been implemented.

Step 2: Feeder Belt Feature

A_1 : In the first design cycle the functionality of the Feeder Belt is implemented. This functionality is that the Feeder Belt stops when sensor S8 and sensor S1 detect a block. The belt starts running again when one of the two sensors do not detect a block anymore.

$B_{1,1}$: The RUNNING and STOPPED states are implemented in the Feeder Belt controller. The control logic has been implemented in the main controller.

$C_{1,1}$: It can be seen in Figure 4.15 that when both sensor S1 and sensor S8 detect a block, the Feeder Belt changes to the STOPPED state. When a block is removed from position P3, the belt starts moving again. It can therefore be concluded that the desired behavior has been implemented.

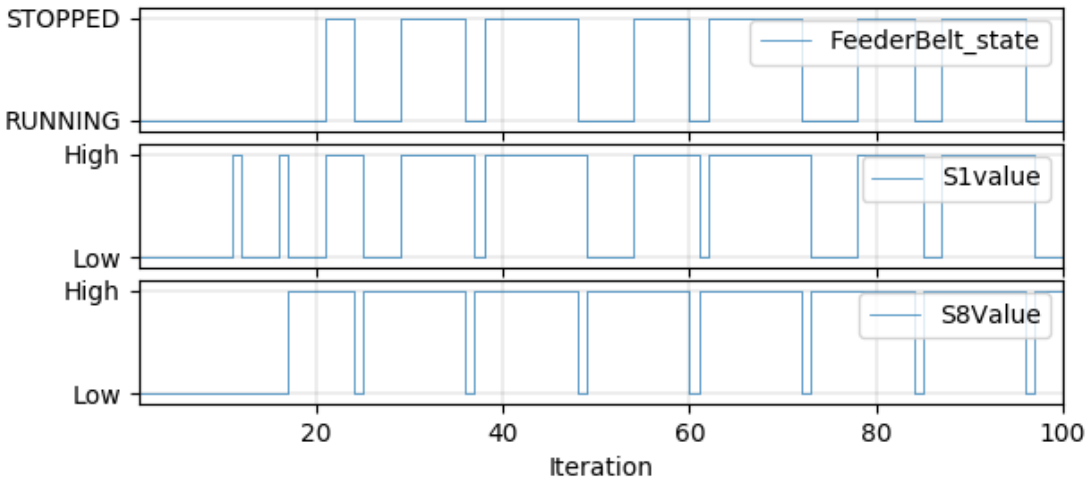


Figure 4.15: Simulation results of the ECS for the Feeder Belt with the control logic level of detail. When both sensor S1 and sensor S8 detect a block, the Feeder Belt state changes from RUNNING to STOPPED, when one of the sensors does not detect a block anymore the state changes to RUNNING again.

Step 2: Feeder Feature

A_1 : In the second design cycle the functionality of the Feeder is implemented. The Feeder starts feeding when a block is detected on position P3 by sensor S1, the block is then moved to position P4 and makes place for a new block on position P3. On position P4 no sensor is available to detect a block there. Therefore a position memory block has been designed that detects a rising-edge of the busy signal of the Feeder. When this transition is detected, the memory block remeberes that a block has been entered at position P4.

$B_{1,1}$: The WAITING_FOR_BLOCK state and FEEDING state have been implemented in the Feeder controller. The control logic of the Feeder has been added to the main controller. The position memory block has been implemented with a rising-edge detector on the busy signal of the Feeder.

$C_{1,1}$: Figure 4.16 shows that the Feeder state changes to FEEDING if a block is detected at position P3 and the block is moved to position P4. After that the Feeder state changes back to WAITING_FOR_BLOCK again. The position memory block detects a block and this can be seen in the change of the state of the variable P4_block.
It is observed that that the signal of P4_block does not go low again. This is done after a block is removed by the Extractor, but this functionality has not yet been implemented and is tested again after the Extractor is implemented.
The implemented functionality is as desired and the functionality of the Feeder Belt has not been altered.

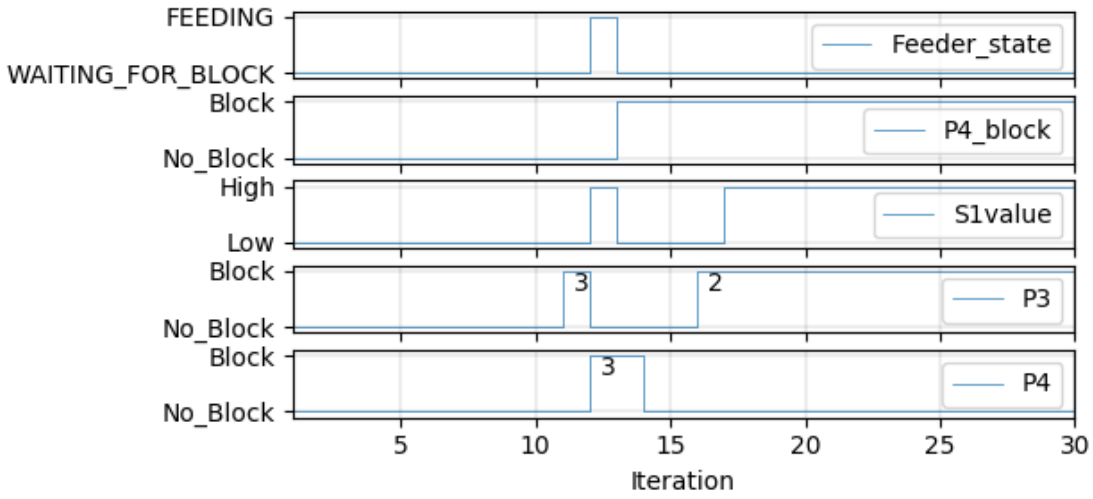


Figure 4.16: Simulation results of the ECS for the Feeder with the control logic level of detail. When sensor S1 detects a block the Feeder state changes to FEEDING and pushes the block from position P3 to position P4. The position memory block P4_block state changes when the Feeder has moved a block to position P4. The numbers 3 and 2 represent the identification numbers of the blocks that are present on the positions

Step 2: Molder Feature

A_1 : In the third design cycle the functionality of the Molder door is implemented. The Molder door opens when the variable P4_block detects a block. Then the door opens and the Extractor can remove the block.

$B_{1,1}$: The OPEN and CLOSED state for the Molder have been implemented, together with the control logic in the main controller.

$C_{1,1}$: It is shown in Figure 4.17 that when a block is pushed in, the position memory P4_block detects a block and the Molder door is opened. It can be concluded that the desired behavior has been implemented and the behavior of the earlier implemented parts has not been altered.

The Molder door does not close because the position memory variable P4_block does not go low because the block is not removed from position P4. This is done by the Extractor which is not present at this point in the implementation. This functionality is implememted and tested in the design cycle of the Extractor.

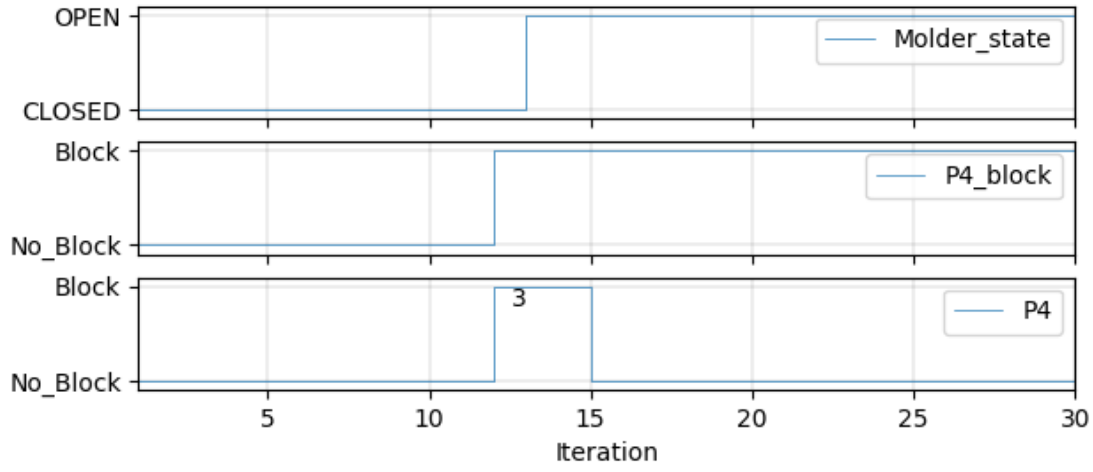


Figure 4.17: Simulation results of the ECS for the Molder with the control logic level of detail. When the block detection entity P4_block detects a block, the Molder door opens.

Step 2: Extractor Feature

A_1 : In the fourth design cycle the functionality of the Extractor is implemented. The Extractor moves a block from position P4 to position P5. The Extractor starts moving after the Molder door is opened. It also checks sensor S4 if a block is present at position P5 to detect if the block drop location of the Extractor is free to make the blocks not collide. The position memory, P4_block, functionality has been extended to detect the rising-edge of the busy signal of the Extractor. This signal is used to close the Molder door again. The magnet of the Extractor is implemented, which is turned on and off in the right states.
A small feature has been implemented to remove the block from position P5. In this way the integrated behavior of the Feeder, Molder and Extractor can be simulated.

$B_{1,1}$: All the states of the Extractor have been implemented, they are made such that they go to the next state automatically and that the magnet is turned on and off in the right states. The functionality of the position memory is extended to detect a rising-edge of the Extractor busy signal.

$C_{1,1}$: It can be seen in Figure 4.18 that when the Molder door opens the state of the Extractor changes and that the block is moved from position P4 to position P5. After the Extractor has been finished, the Molder door is closed again and a new block is pushed onto position P4 by the Feeder. The position memory detects now that the block is removed by the Extractor. It can be concluded that the desired functionality has been implemented.

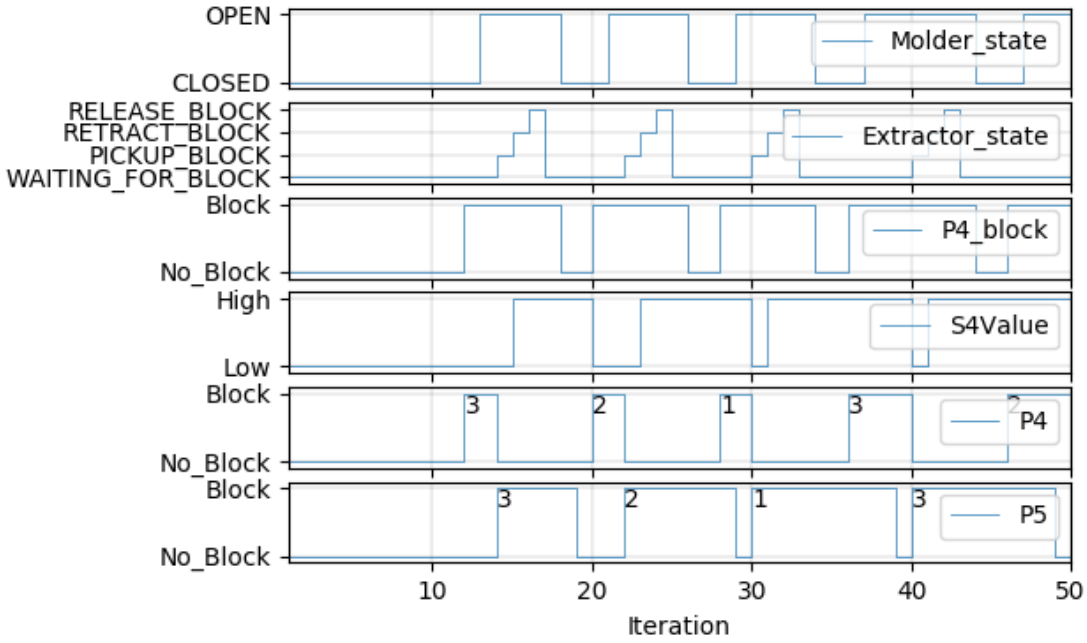


Figure 4.18: Simulation results of the ECS for the Extractor with the control logic level of detail. When a block is detected by the P4_block variable, the Molder door opens and the Extractor moves the block from position P4 to position P5. The P4_block variable detects that the block has been removed from position P4 and the signal goes low. Then the Molder door closes again and is ready to receive the next block from the Feeder.

Step 2: Extractor Belt Feature

A_1 : In the fifth design cycle the functionality of the Extractor Belt is implemented. The Extractor Belt stops when sensor S5 and sensor S6 detect a block. The belt starts running again when one of the block detection sensors does not detect a block anymore.

$B_{1,1}$: The RUNNING and STOPPED state have been implemented in the Extractor Belt controller. The control logic has been implemented in the main controller.

$C_{1,1}$: It can be seen in Figure 4.19 that when both sensor S5 and sensor S6 detect a block the belt stops. When a block is removed from position P7 and sensor S6 does not detect a block anymore the belt starts running again. The functionality of the earlier implemented systems has not been altered and it can be concluded that the desired functionality has been implemented.

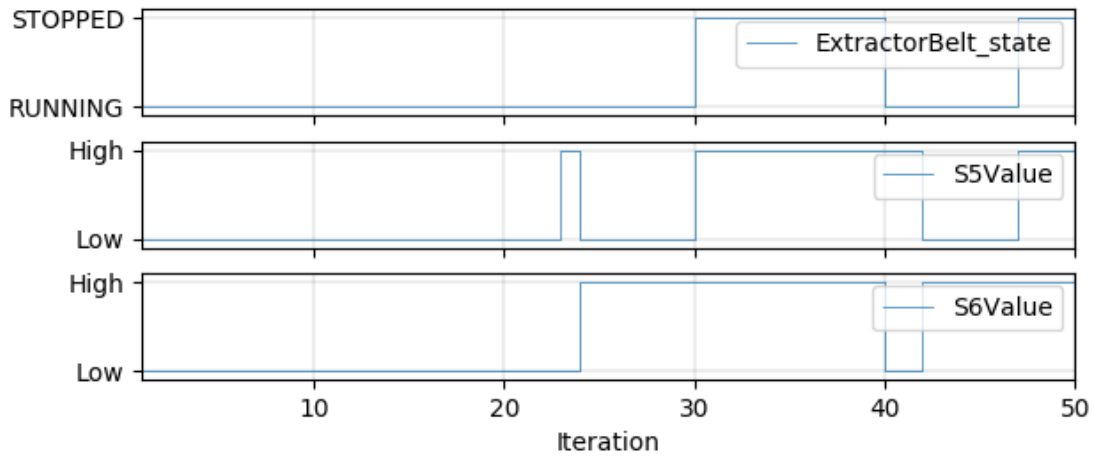


Figure 4.19: Simulation results of the ECS for the Extractor Belt with the control logic level of detail. When sensor S5 and sensor S6 detect a block the state of the Extractor Belt changes to STOPPED, when one of the sensors does not detect a block anymore the Extractor Belt starts running again.

Step 2: Rotator Feature

A_1 : In the sixth design cycle the functionality of the Rotator has been implemented. The Rotator waits until a block is detected on position P7 by sensor S6. When a block is present, the Rotator picks up the block and moves the block from position P7 to position P1. The magnet that is present on the Rotator is also implemented and controlled.

$B_{1,1}$: All the states of the Rotator have been implemented and the control logic has been implemented in the main controller.

$C_{1,1}$: It can be seen in Figure 4.20 that when a block is detected by sensor S6 at position P7 the block is moved to position P1 by the Rotator. The blocks are fed into the system on position P1 on the Feeder Belt, which can be seen in the simulation results. It can be concluded that the desired behavior has been implemented.

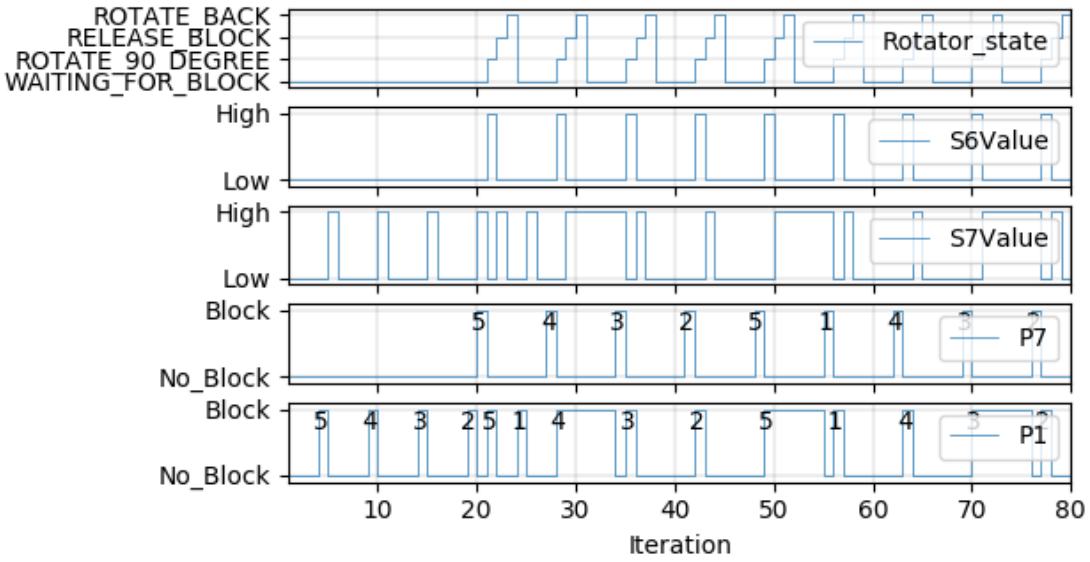


Figure 4.20: Simulation results of the ECS for the Rotator with the control logic level of detail. When a block is detected by sensor S6 the state of the Rotator changes and the block is moved from position P7 to position P1.

Step 2: Complete test for all features

Now all control logic has been implemented and that means that blocks can move constantly in the production-cell. The behavior of the complete production-cell is tested in three different tests. The first test puts one single block in the production-cell. The second test puts three blocks in the production-cell and the third test keeps pushing blocks in the production-cell until it deadlocks.

In the simulation results of one block it is can be seen that the block constantly steps over all the positions and that the desired behavior of the different features is achieved. In the simulation results for three blocks it is also shown that the blocks constantly steps over the positions and that the behavior of the featurs is still as desired. In the simulation results where as much blocks as possible are put into the systems untill it deadlocks, it is shown that the system does not deadlock. The reason for this is that blocks are moved atomic between positions and that it takes more unitless steps to make the control decisions, therefore there is always free space on for example position P5 to move a block to by the Extractor. Or said differently the capacity of the features is higher than the supply of blocks. There is not enough level of detail present to simulate a deadlock of the system.

Conclusion on Iteration 1

In some of the simulations of features not all necessary input signals are available to simulate the complete behavior. In these cases only the available signals are implemented and tested and then moved to the next feature. This can be seen when the Molder door only opens, but because there are not yet signals available from the Extractor to close the Molder door, the next feature is implemented. The closing of the Molder door is then tested later in the design cycle of the Extractor. In this case it is possible to implement functionality to test the closing of the door by removing the block. But this is extra work and it can be easier tested in the next design cycle.

Only a portion of all the control logic has been implemented because not all the signals, from for example the encoders, are available. Some of the signals are essential to test the functionality. For example in the design cycle of the Extractor, extra functionality is added which re-

moves the block from position P5 after the Extractor has put a block there. This functionality is put into place as an replacement for the implementation of the Extractor Belt which normally would remove this block. These small functionalities must be added to test the functionality for multiple blocks in a single simulation and to get information about the implemented behavior.

It can be concluded that when applying the SDM to implement control logic, sometimes implementing small functionality can help to test the features better. These small functionalities are a temporarly implementation for the next feature and contain some of the essential behavior. In other situations this can be skipped and functionality is added in a later stage, a balance have to be found in this. The event-based plant provided a mechanism to test the control logic without the discrete-time plants.

In this design cycle it had been chosen to implement one level of detail for multiple features. This has been done because the features depend on one another and testing is usefull if the complete setup can be tested together, instead of a single feature.

4.3.2 Level of detail 2: Discrete-time implementation

Step 1:

In this second level of detail, the unitless steps in the simulation are replaced by discrete-time steps. Discrete-time is necessary for the implementation of discrete-time plants. A separate design cycle is used to change the unitless steps to discrete-time because a significant part of the present functionality has to be changed or added. In this seperate design cycle, the implemented functionality of the plant can also be tested to validate that it behaves properly. The output of the simulation of this second design cycle can be compared to the output of the first design cycle to check if the desired behavior is still present. To mimic the block movement by the different features, temporary positions and timers will be added to simulate that moving a block takes time and to check if the control logic still has the desired behavior.

Step 2: Feeder Belt Feature

A_2 : In this first design cycle the discrete-time is implemented in the Feeder Belt. The file of the first design cycle is extended and adjusted with the new functionality.

$B_{2,1}$: The Feeder Belt plant has been changed and the positions are removed and replaced by a belt where the velocity of the belt is mathematically integrated into real positions instead of the discrete positions. Sensors detect a block if the blocks are in a certain range of the belt.

$C_{2,1}$: Discrete-time has been implemented successfully and the control logic still behaves the same as the first iteration.

Step 2: Feeder Feature

A_2 : The file of the Feeder of the first design cycle will be adapted here, the new designed functionality of design cycle A_2, of the Feeder Belt, will also be implemented and the steps are replaced by discrete-time. An extra position will be added between position P3 and position P4 were the block will be stored in temporarily while the Feeder is feeding. For now a timer will be used to simulate that feeding in a block takes time. This is done because the encoder of the Feeder that serves as a control decision signal for the ECS is not yet implemented.

$B_{2,1}$: The steps have been replaced by discrete-time and the temporary position and timer has been added.

$C_{2,1}$: The steps have been changed to discrete-time with success and the desired behavior has been obtained.

Step 2: Molder Feature

A_2 : The file of the Molder of the first design cycle will be adapted here, the new designed functionality of design cycle A_2, of the Feeder, will also be implemented and the unitless step is replaced by discrete-time. In the Molder a timer will be added to represent that opening and closing of the Molder door takes time.

$B_{2,1}$: The implementation of the Molder of iteration 1 has been replaced by the new designed implementation of the plant and the timer has been added.

$C_{2,1}$: The new features have been implemented with success and the discrete-event behavior of the plant is still correct.

Step 2: Extractor Feature

A_2 : The file of the Extractor of the first design cycle will be adapted here, the new designed functionality of design cycle A_2, of the Molder, will also be implemented and the steps are replaced by discrete-time. An extra position will be added between position P4 and position P5. This is a temporary position where the blocks are stored when the Extractor is moving the block. This mimics that the block is not present on position P4 or position P5 when the block is moved by the Extractor.

$B_{2,1}$**:** The new position has been placed between position P4 and position P5 in the plant and the timer is implemented.

$C_{2,1}$: The new functionality has been implemented with success.

Step 2: Extractor Belt Feature

A_2 : The file of the Extractor Belt of the first design cycle will be adapted here, the new designed functionality of design cycle A_2, of the Extractor, will also be implemented and the steps are replaced by discrete-time. The positions from which the belt consist are replaced by a continuous belt where the velocity is mathematical integrated to real positions instead of discrete positions.

$B_{2,1}$: The steps have been replaced by discrete-time and the Extractor Belt functionality has been replaced by a continuous belt.

$C_{2,1}$: The new functionality has been tested and implemented with success.

Step 2: Rotator Feature

A_2 : The file of the Rotator of the first design cycle will be adapted here, the new designed functionality of design cycle A_2, of the Extractor Belt, will also be implemented and the steps are replaced by discrete-time. An extra position will be added between position P7 and position P1. At this position a block will be stored temporarily if the block is moved by the Rotator. Also a timer will be added which represent that it takes time to move the block.

$B_{2,1}$: The position has been added and the timer has been implemented.

$C_{2,1}$: The new functionality has been tested and implemented with success.

Complete test for all features

The discrete-time level of detail has been implemented for all the features. Now the complete implementation can be tested, this is again done with three tests. For one block, three blocks and on deadlocking.

In the simulation results for the test with one block and with three blocks the desired results are observed, it only has been observed that in some instances the blocks dissapear from the simulation but other blocks do go well through the complete setup. The simulation to test for deadlock did not have an conclusive result. This was due to the dissapearance of the blocks.

The functionality of the discrete-event plants was not in the correct order and blocks were put "on top" of each other with which pointers were overwritten and thus blocks were lost.

Conclusion on Iteration 2

In an initial second level of detail implementation, it is tried to implement the discrete-time steps on the implementation of the model of the Rotator of design cycle 1 in a single design cycle. This gave a lot of errors which were not traceable, therefore it was decided to split the integration up into six design cycles.

In this design cycle the plants are not present yet, but to simulate that the blocks are moved by the different features, timers and temporary positions have been used to simulate that movement of blocks takes time. The encoder signals that would normally be used to change the state of the different controllers are not present yet and are implemented in a later stage. These timers and temporary positions that have been added, serve as an additional test of the control logic.

It can be concluded from this second design cycle that a single design cycle is necessary to go from unitless time steps to discrete-time. The unitless time steps serve as a testbed for the control logic, but to integrate the control logic with discrete-time loop controllers and plants the steps must be converted into discrete-time. This can be seen as a preparation step for integration of the discrete-event logic and the control loops.

Some flaws were encountered in this iterations. In some design cycles bugs were encountered in the design of the event-based plant, because not all signals were available to make discrete-event control decisions. One of the bugs was the dissapearing of blocks. It has been decided that this is not of great concern because in the next level of detail the encoders of the plants and the plants itself will be integrated and that this problem is therefore solved in the next design cycle. The overall behavior was as expected and some blocks moved through the complete setup and this is an indication that the overall behavior is correct.

4.3.3 Level of detail 3: Plant implementation

Step 1:

In this third design cycle, the discrete-time plants are implemented as dummy plants as was proposed in Section 3.2. This is done together with the loop controllers which are PID controllers. In the real plant the belts are velocity controlled and the other systems are position controlled. In this design cycle the belts will also be position controlled for convenience. Due to the implementation of the encoders, all the remaining states of the FSM's which depend on signals of the discrete-time plants to make control decisions, can be implemented in the different controllers. The continuous-time plants are converted using the Z-transform into discrete-time plants which can be implemented.

Before the design cycle was started, the modules of the PID controller, encoder and discrete-time plant have been developed and a complete control loop has been tested to verify that the operation is correct.

Due to the testing and validation of the modules in advance, it is known up front that the modules work correct and the implementation of this level of detail is the same for all the features. Therefore this level of detail is integrated for all the features in one single iteration.

Step 2: All Features

A_3 : In this design cycle the discrete-time plant and PID controller is implemented for every feature. The encoder and the PID controller. The remainder of the states of the FSM's are also implemented.

$B_{3,1}$: The new modules have been integrated in the subsystems and the control logic has been implemented.

$C_{3,1}$: It has been observed in the simulation results that the Feeder Belt starts rotating to early. When the Feeder starts to move a block from position P3 to position P4 and position P3 comes free, sensor S1 does not detect a block anymore. At that point the Feeder Belt starts running again. But this means that the block on the Feeder Belt would collide onto the Feeder arm.

A_4 : The FSM of the Feeder Belt is changed and the new implementation is tested. The adjusted FSM, with the new conditions, is shown in Figure 4.21.

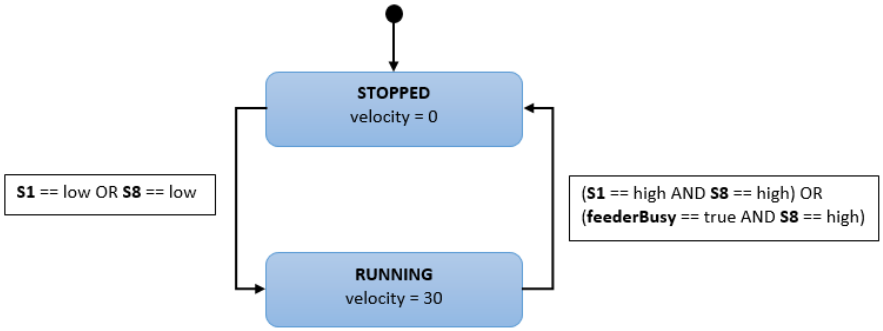


Figure 4.21: Finite State Machine for the Feeder belt with the new conditions after the redesign in iteration 3. The addition is when the Feeder arm is feeding and a block is present at position P2 that the Feeder Belt does not start rotating to prevent a collision of a block and the Feeder arm.

$B_{4,1}$: The control logic of the Feeder Belt has been changed to the new FSM in Figure 4.21.

$C_{4,1}$: With the changed functionality of the Feeder Belt, all the behavior is now as desired. It can be observed in the simulation result that all the dummy plants move between position '1' and '0' depending on the current state of the FSM of the feature. All the blocks move through the complete setup in an correct way and blocks do not dissapear anymore as was observed in the second design cycle. In Figure 4.22, the dummy plant is shown for only the Rotator feature.

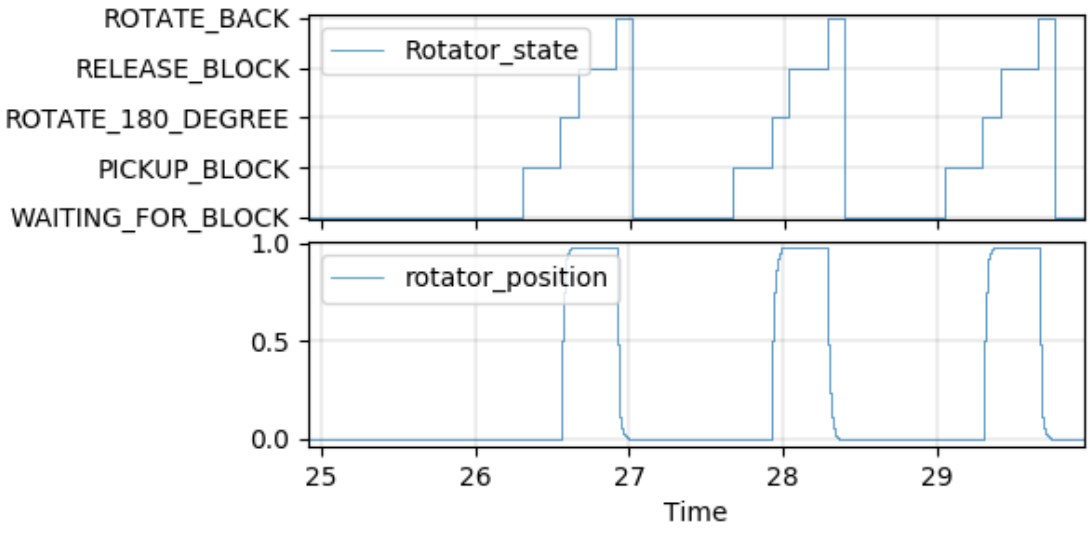


Figure 4.22: Simulation results of the ECS for the Rotator with the implementation of a plant level of detail. It can be observed that the Rotator moves between '1' and '0' according to the state of the Rotator FSM.

The ECS is tested using the three tests with one block, three blocks and for deadlocking. The test with one and three blocks has the expected results. In this design cycle the test for deadlocking also give the desired results. At a certain point in in time the blocks do not move anymore and the complete setup comes at a halt. The configuration in which the deadlock occurs is shown in Figure 4.23.

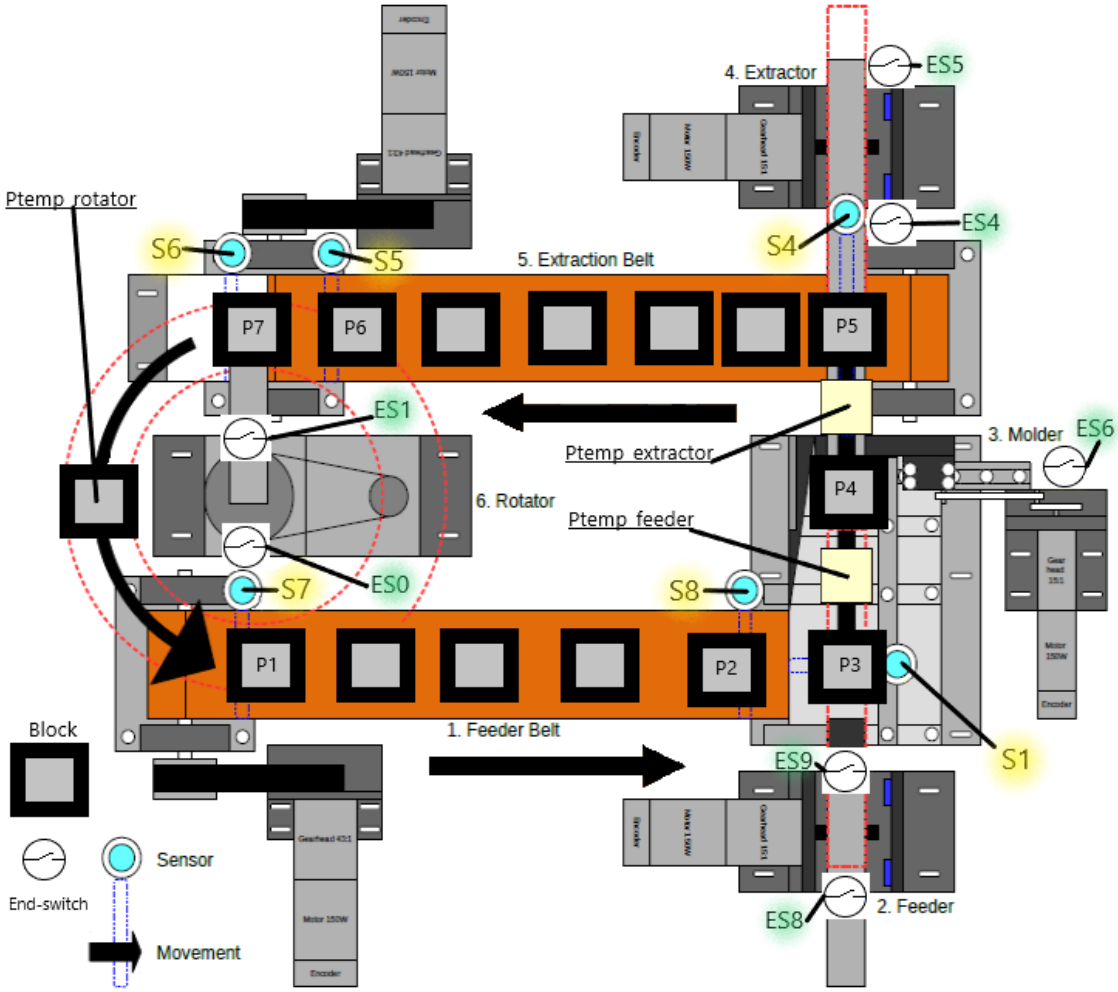


Figure 4.23: This model shows the configuration of the blocks in which the production-cell deadlocks. All the blocks are not able to move to the next position because these are occupied by another block.

Conclusion on Iteration 3

It can be concluded from this design cycle that a well developed Finite State Machine is of importance for a smooth integration of the remaining states in the control logic. The tests give feedback on the design and the designer can validate the simulation, this acts as an error detection mechanism.

Before this design cycle is started, modules were designed and tested before they were integrated in the ECS. In this way the correct behavior of the modules was tested and only the integration can give problems. Due to the well-tested modules and because the level of detail is the same for all the features, it has been decided to implement this level of detail for all the features at once. In Table 4.2 the design steps are shown that have been taken so far into the design and testing of the ECS. The green dot illustrates the redesign that has been done in this iteration for the control logic of the Feeder Belt.

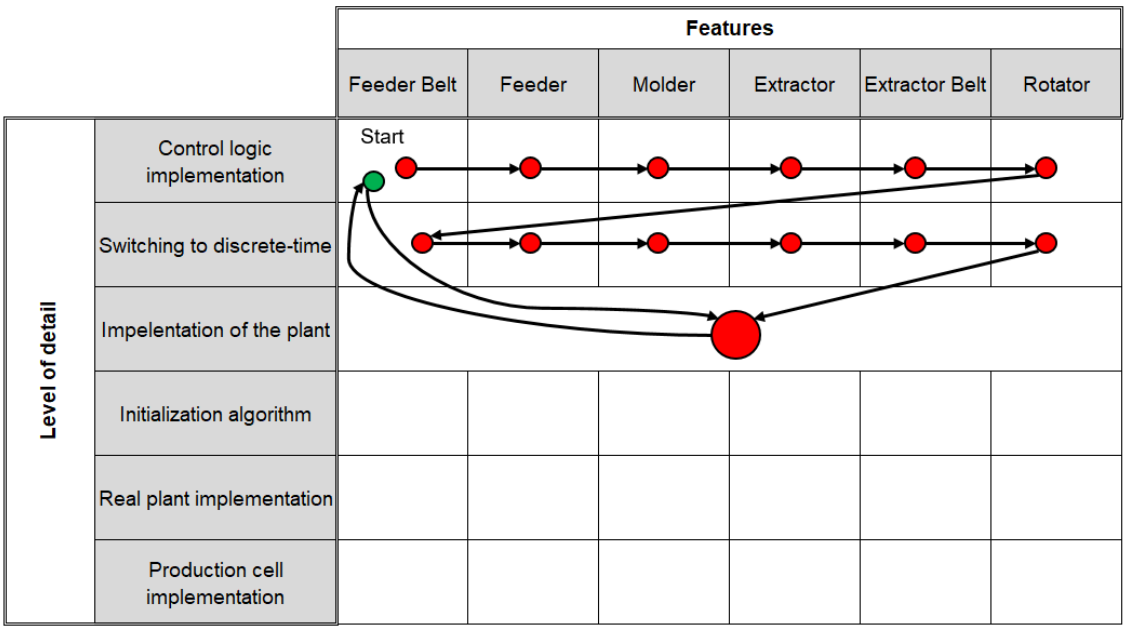


Table 4.2: This table shows the implementation that has been performed up to this point in the design. It can be seen that the propsed implementation order in Table 4.1 has been followed but in the third design cycle the level of detail is implemented in a single iteration. The green dot indicates the redesign that was necessary for the control logic of the Feeder Belt.

4.3.4 Level of detail 4: Initialization implementation

Step 1:

In this fourth design cycle the initialization algorithm for the different features is implemented. End-switches are implemented in the Feeder, Molder, Extractor and Rotator. To perform the initialization algorithm, FSM's are implemented in the loop controller and extra initialization states are implemented in the FSM of the Feeder, Molder, Extractor and Rotator. The overall controller FSM has also been changed, such that the initial state is INITIALIZING. When all systems have been initialized the production-cell starts operating.

Step 2: All Features

A_5 : First the end switches are implemented in the plant, this is done by setting the output to "high" if the plant is in a certain range. The extra states are added to the FSM of the Feeder, Molder, Extractor and Rotator. These are the states INITIALIZING, INIT_READY and STARTUP. The FSM has been implemented to the PID controllers such that it has the states ON, OFF and INIT.

$B_{5,1}$: The extra states and end switches have been implemented in the control logic.

$C_{5,1}$: The initialization is tested by setting the initial position of the Feeder, Molder, Extractor and Rotator to non-zero. When the ECS starts the system is first controlled to zero untill the end switches senses that the positions are (near) zero. The system has not been tested for one block, three blocks and deadlocking because this is not of interest for this iteration and that functionality has not been altered. In the simulation results it is shown that the initialization behaves correctly and that the desired behavior has been implemented. The simulation results are shown in Figure 4.24.

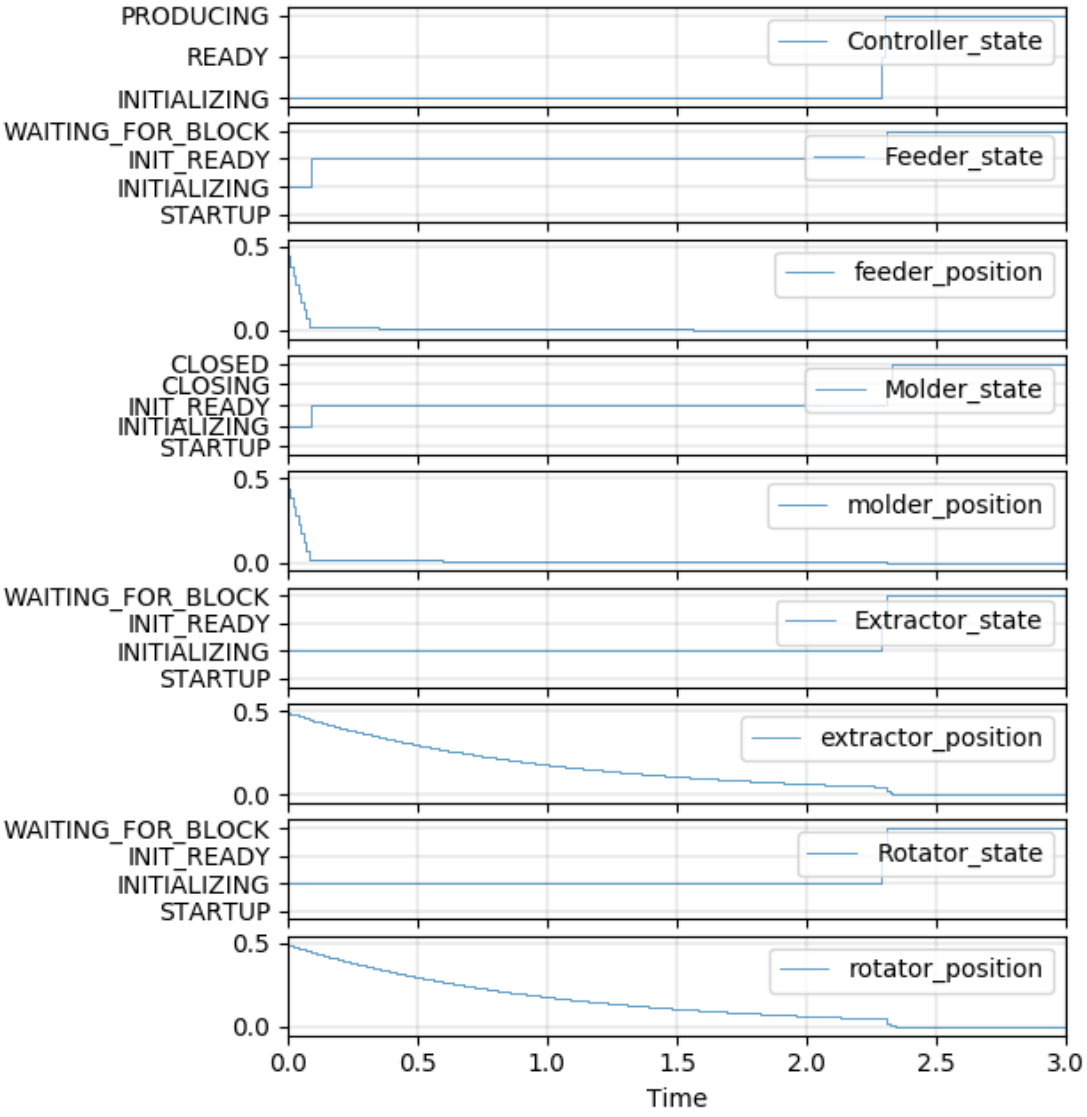


Figure 4.24: Simulation results of the initialization algorithm that is implemented in design cycle for the initialization algorithm. When the initialization starts, the discrete-time plants move slowely to the start position. When this start position has been reached, the state of the controllers change to INIT_READY. When all the plants are initialized, the overall controller starts the production-cell.

4.3.5 Conclusion on Iteration 4

In this design cycle, level of detail has been implemented for all the features that need initialization in one inner cycle, as was done in design cycle 3. This was possible because the implementation that has been done was the same for the features that needed an initialization. The implementation went very smooth and no significant errors have been encountered.

4.3.6 Level of detail 5: Real plant implementation

Step 1:

In this fifth design cycle, the models of the discrete-time plants have been changed to the models of the real plant. The dimensions of the transport belts have been changed, together with the positions of the block detection sensors such that the implementation of the virtual plant matches that of the real plant.

To tune the PID controllers a simple continuous-time model has been developed for all the systems and parameters from (van den Berg, 2006) have been used. The 20sim software package (Controllab Products, 2020) has been used for simulation and tuning of the loop controllers. The software has also been used to convert the continuous-time plant models into discrete-time models.

During the implementation of the dummy plants in design cycle 3, the two transport belts were implemented as if they were position controlled. In the real plant they are velocity controlled plants. This will also be implemented in this design cycle.

Step 2: All features

A_6 : In this design cycle the file of design cycle 4 has been used and adjusted. The parameters of the plant and the PID controller have been changed to the parameters that have been found during the tuning process using the software package 20sim. The belt dimensions have also been changed to the dimensions of the real plant.
The positions in the control logic, which are used as control decisions, have been changed from the positions of the dummy plants to the real encoder values.

$B_{6,1}$: The parameters of the discrete-time plants and PID controller have been changed. The setpoints in the control logic have been changed to the encoder values of the real plant.

$C_{6,1}$: The implementation of the parameters has obtained the desired behavior. The results are the same as the results that have been obtained in design cycle 4, only the shape of the graph of the positions is different.
The complete setup has not been tested with three blocks and for deadlock because only parameters have been changed and no aspects that influence deadlock have been altered. The calculation of the velocity for the belts have been altered, and therefore a test with a single block has been done to verify that the block moves through the entire setup in the correct way and that the velocity calculation is correct.

4.3.7 Conclusion on Iteration 5

During the implementation of the real plant parameters some issues were encountered for the velocity calculation in the controllers of the Feeder Belt and the Extractor Belt.

It could have been better to implement this level of detail seperate for every feature. Or seperate for the velocity controlled plants and the position controlled plants. A lot of debugging was necessary to find the issue which was eventually that the velocity was not calculated in the correct way.

From this design cycle it can be concluded that it is not always good practice to implement one level of detail for multiple features in one iteration.

Only one design cycle and no redesign has been used in this design cycle altough some errors have been encountered during the implementation of this level of detail. This has been done because the goal for this level of detail was the changing of the parameters such that the virtual plant matches the real plant. The goal was not to implement only the velocity controller for the belts for example. This could have done differently by adding an extra level of detail for the Feeder Belt and Extractor Belt such that this integration could have gone more smooth.

4.3.8 Level of detail 6: Implementation on the production-cell setup

Step 1:

In this sixth design cycle, the ECS is tested on the production-cell setup. Various functionality has been implemented in this cycle. During previous design cycles, it was concluded that the interchangeability of the chosen architecture was not as desired. Therefore an extra module

has been added which acts as an interface to the real plant. Values are written and read from this module during the execution of the various FSM's.

Different modules have been designed in this design cycle that are necessary to execute the ECS. Functionality has been added to make the complete controller runs at a rate of 100Hz. A small GUI has also been designed to start, stop and initialize the production-cell. Other functionality is the reading and writing to the FPGA using the GPMC interface. Because the pheripherals of the production-cell are distributed over two Ramstix boards communication between these boards has also been implemented using the ZMQ messaging library (ZeroMQ, 2020).

In this design cycle it has been chosen to implement this level of detail for all the features in one design cycle because the functionality is the same for the different features.

Step 2: All Features

A_7 : In this design cycle the communication with the pheripherals will be implemented, together with the GUI. A program for the second Ramstix board will be made that reads the different sensors and encoders that are connected to the second Ramstix board and send it to the first Ramstix board. This program receives new motor values which are written to the motors that are connected to the second Ramstix board.

$B_{7,1}$: All the functionality has been implemented into the ECS and a seperate program has been made for the second Ramstix board.

$C_{7,1}$: After testing, it has been observed that the desired behavior has not been obtained. The movement of all the subsystems was very oscillatory. After an analysis of the results it has been concluded that the execution time of a single cycle of the ECS takes longer than 10 milliseconds seconds which is related to 100Hz.

A_8 : In this redesign the initial rate of 100Hz of the ECS will be lowered to 25Hz. For this, new parameters for the PID controllers are necessary which were designed for a controller that runs on 100Hz.

$B_{8,1}$: The parameters of the PID controllers and the discrete-time plant have been changed. The rate of the ECS is lowered to 25Hz.

$C_{8,1}$: After running the ECS with the new rate of 25Hz it has been observed that the movement was still not as desired, it is still oscillatory but much less then in the second test $C_{7,1}$.

A_9 : A redesign will be made with less aggresive control law for all the subsystems and the maximum and minimum outputs of the PID controllers will be lowered.

$B_{9,1}$: In this redesign the maximum values that can be written to the motor are reduced. The initial values are between -1 and 1. This value is lowered between -0.3 and 0.3 for all the motors.

$C_{9,1}$: The redesign behaves as expected, the motion is quite choppy but the loop controllers are stable. The discrete-event control logic behaves as desired and the complete setup runs as expected. In Figure 4.25 the execution times are shown for multiple runs of the ECS and it can be seen that the maximum execution time of 40 milliseconds, which corresponds to 25Hz, is not exceeded. It can be seen in Figure 4.25 that the maximum execution time is around 6.9 millisecond. The maximum execution time of 6.9 milliseconds suggests that a rate of 100Hz, which corresponds to a maximum execution time of 10 milliseconds, would also have been fine. In this time the writing of the simulation results to the CSV-file are not taken into account. It is assumed that this takes more time than 3.1 milliseconds and that with this time the 10 millisecond bound would has been exceeded. With this assumption the lowering of the rate to 25Hz has been the right decision.

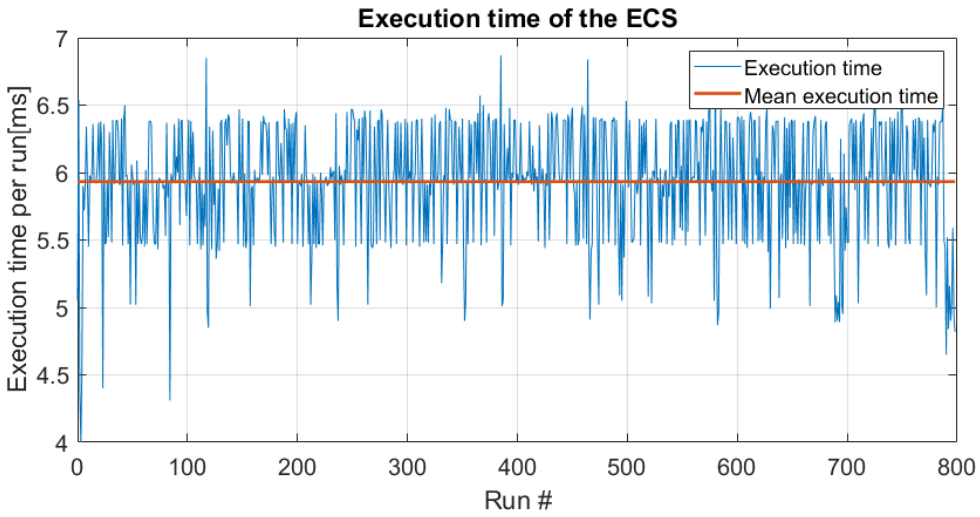


Figure 4.25: This figure shows the execution times for multiple runs of the controller. The mean execution time is 5.9 milliseconds the maximum execution time is 6.9 milliseconds and the minimum execution time is 4 milliseconds.

4.3.9 Conclusion on Iteration 6

In this design cycle it has been observed that the execution time of one cycle of the ECS was too long and that the desired rate of the ECS was not reached. This could have been tested when an extra level of detail was used for an Hardware-in-the-Loop simulation for example.

Some functionality had to be changed and added in the ECS to make the communication of the ECS work with the ZMQ message library. From these changes it can be concluded that the chosen architecture did not support interchangeability very well.

4.4 Evaluation and reflection

The complete design of the ECS has been implemented with succes. To validate if the order of implementation of the levels of detail was correct, the design cycles are plotted and is shown in Table 4.3.

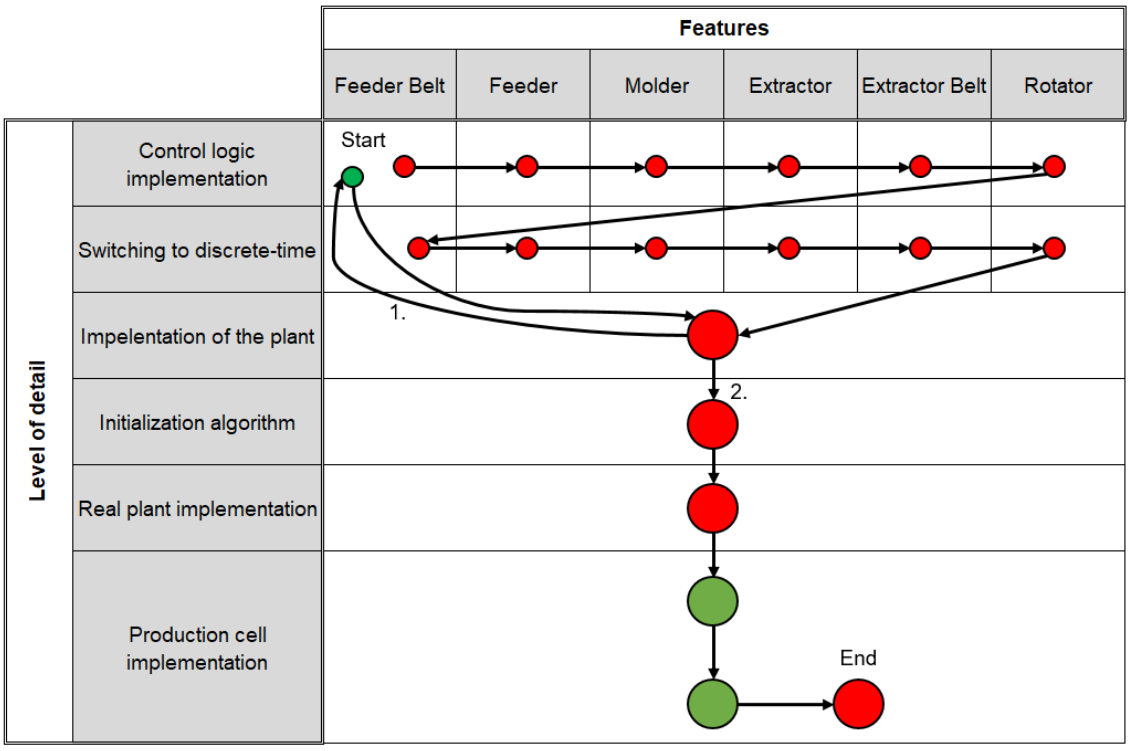


Table 4.3: This table shows the implementation order of all the design cycles, together with the redesigns that are shown in green. The numbers 1 and 2 indicate the order in which the arrows from the red dot have to be followed.

It can be seen in Table 4.3 that the first two levels of detail are integrated per features, but no redesigns were necessary here. In the latter four iterations the level of detail has been implemented for all the features in one design cycle. In the level of detail of the plant implementation and in the level of detail on the real plant a redesign was necessary. This inidicates that the design steps were too coarse and too much functionality was integrated in one cycle which lead to errors.

It can be concluded that implementing the levels of detail per feature could have been better for some design cycles. An approach is to have more levels of detail, which leads to design cycles where less functionality has to be implemented in a single cycle.

5 Evaluation

In Section 5.1 the aspects are evaluated from the application point of view and that have been observed during the application of the SDM on the production-cell setup. In in Section 5.2 the application of the SDM to the production-cell will be reflected from the point of view of the requirements that have been derived in the analysis in Section 3.2.

5.1 Evaluation of application aspects

During the execution of the Structured Design Method on the production-cell, aspects have been encountered which influence the application of the SDM to discrete-event signals, these aspects are discussed below.

Implementation order did not fit the SDM framework

In the application of the SDM on the Segway, the levels of detail are implemented one after another for one feature and when the complete feature is implemented then the next feature is implemented. This is done differently in the application of the SDM on the production-cell, where the features are implemented one after another for one level of detail and when the complete level of detail is implemented there is proceed to the next level of detail. So there is iterated over the levels of detail instead of the features.

This order is due to the dependencies that are different between designing a control law and a discrete-event controller. When in the case of a control law one level of detail is not correct, the simulations with the lower levels of detail also have to be adjusted to get the control law working correct, they depend on each other. In the case of the discrete-event controller, when one level of detail is not correct, the simulations with the lower level of detail do not have to be changed. This can be seen when for example the initialization algorithm would not have worked correct, then the control logic does not have to be adjusted, they are independent of each other.

This dependency is also different for features. In the case of the Segway the features depend on each other, the Segway robot cannot drive when it is not able to stand upright. But this does not hold for the production-cell. When the control logic feature is for example implemented for only the Feeder, than this feature can be tested. Also when the Feeder Belt or Molder are not present yet. The features are independent on each other.

It can be argued that the steps in the SDM, as they are described in (Broenink and Broenink, 2019), are a subset of the possible design steps that can be taken. This has been observed during the execution of the SDM. Therefore a new scheme is proposed for the Structured Design Method and is shown in Figure 5.1. This scheme fits the application for the design of a control law and a discrete-event controller.

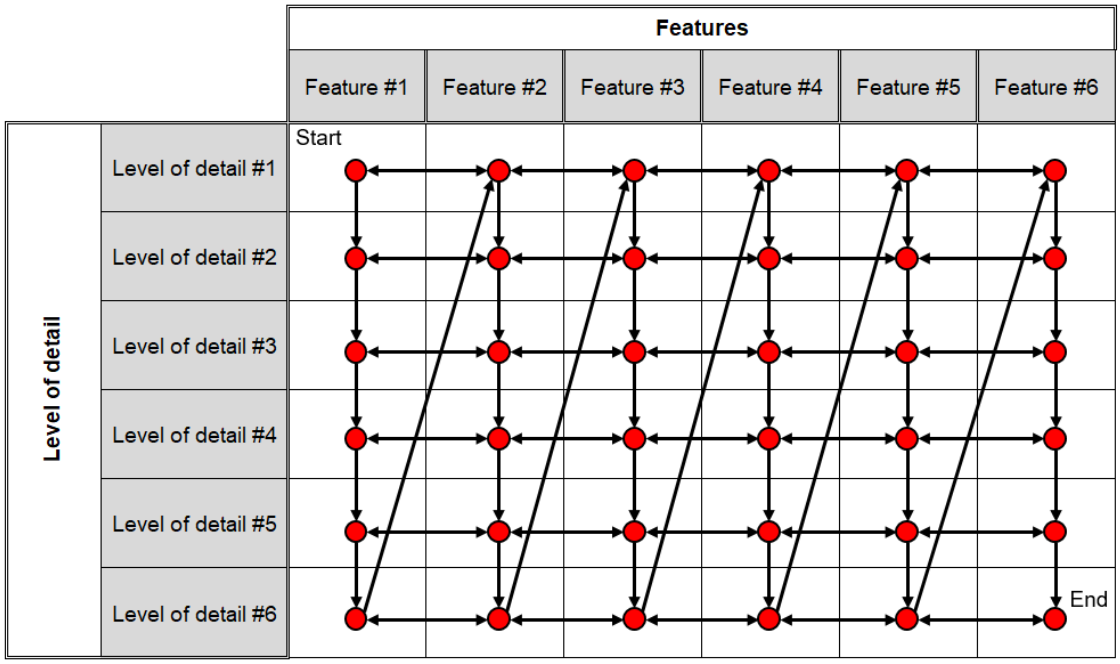


Figure 5.1: New proposed design scheme for the Structured Design Method. The rule for following this scheme is that all the circles must have been passed before reaching the end of the design cycles. With this scheme the SDM is applicable to both the design for control laws and discrete-event controllers.

A rule about this scheme is that before reaching "End", all the circles must have been passed, one can also walk over already implemented dots. This scheme fits both the Segway application and the production-cell application of the SDM. It is not possible to do both discrete-time and discrete-event applications in this SDM scheme at the same time. This is because in the application to a control law and a discrete-event controller, the levels of detail are different. That means that not all levels of detail can be used in the implementation for a feature.

The horizontal arrows in Figure 5.1 are bidirectional because the features are independent and feature #4 can be implemented before feature #3 for example.

Determination of the levels of detail

The Structured Design Method has been applied from the paradigm of (Broenink et al., 2016). In this paper a co-design process has been used. The levels of detail that have been chosen in the application of the SDM to the production-cell are derived from this paper and are shown in Figure 5.2.

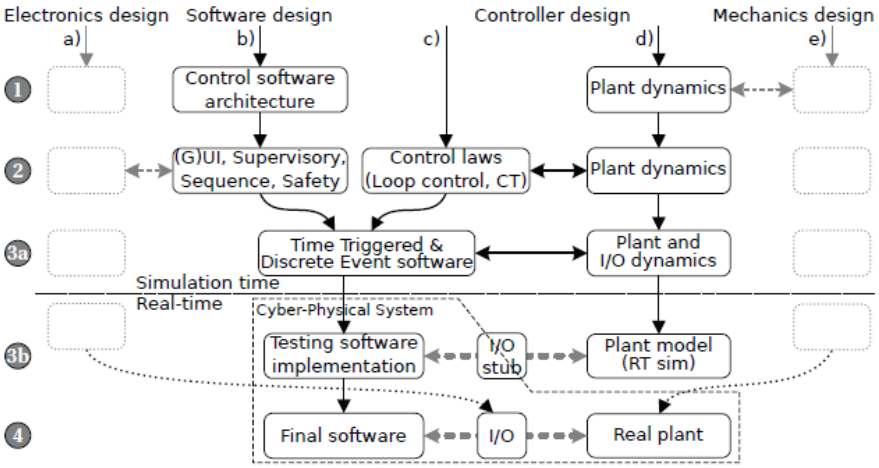


Figure 5.2: This co-design process (Broenink et al., 2016) is the perspective from which the levels of detail have been derived during the application of the SDM on the production-cell.

The simulation framework did not have the desired interchangeability

The simulation framework that has been proposed and implemented in the preparation phase did not had the desired interchangeability. Pointers were used to exchange information between the controller and the plant but this was not interchangeable because functions were hard-coded and this did not provided the desired flexibility. It has been found that the ZMQ messaging library (ZeroMQ, 2020) did provide the desired flexibility.

If separate software programs were used for the controller and the virtual plant and would communicate using this library, than there could be easily switched between the virtual plant and the real plant.

Another side effect of using this library would is that the communication in the architecture of the control software is more explicit implemented. As a result the level of detail on the implementation on the real plant would have gone smoother and no changes in the software regarding the communication would have been necessary. This automatically separates the programs for both the RaMstix boards because the interface of the ZMQ library requires it to communicate via this library instead of with pointers. No changes would have been necessary in the last level of detail design cycle, but was automatically implemented correct due to the interface of this messaging library.

Suboptimal choice of level of detail

From Table 4.3 it can be seen that some redesigns were necessary. One redesign was necessary due to an error in the Finite State Machine of the Feeder Belt. This behavior was overlooked during development, but due to testing the error was observed. The other redesigns in the level of detail of the implementation on the real plant was due to the implementation of too much functionality which lead to errors in the step from the virtual plant to the real plant. Also the failing support of the interchangeability of the software architecture made it that already implemented functionality had to be adapted in this level of detail.

If the interchangeability would have been better and only minor functionality had to be added or changed, this level of detail would have gone smoother. A level of detail that could have been added is a Hardware-In-the-Loop (HIL) simulation, this steps comes from the paradigm in Figure 5.2. During this level of detail, the implementation could have been verified on the target

device with the virtual plant. With the ZMQ message library easy interchangeability would have been possible.

The redesigns that are performed are different than the redesigns for the Segway. In the Segway there are dependencies inside one feature, so if one level of detail is not correct the complete features needs to be redesigned. In the case of the production-cell, the features are independent. This means that only one level of detail for one feature has to be redesigned, as was the case in design cycle 3 for the Feeder Belt and can be seen in Table 4.3.

Merging of the discrete-event control logic and the discrete-time loop controllers

In the analysis in Section 3.2 a dummy plant has been proposed which has been used as a temporary plant and controller. This dummy plant made it possible to postpone the merging of the discrete-time loop controllers and the discrete-event controller. In the case study of the production-cell this postponing was necessary because the PID control parameters and the plant parameters were not known at the time of the implementation of this functionality.

In a practical situation it can also occur that the design of the control laws is not completely finished. Then the proposed dummy plant can offer a solution to finish the discrete-event controller, while there is still worked on the control laws. In this way the control engineer and the embedded software engineer can work concurrently and can work more independent.

The dummy plants have been a success during the implementation of the ECS for the production-cell. It has solved the problem of not having the finished control laws available during the implementation of the loop controllers.

Implementing multiple features in one design cycle

In the latter four design cycles all the features have been implemented for a single level of detail in one design cycle. This has been done because the functionality that needed to be implemented in the features was the same for all features.

In design cycle 5 some errors were encountered during the implementation. This was partially due to the functionality that was not the same for all features, and partially due to functionality that had to be changed.

It is advised to implement multiple features for one level of detail only when the functionality for all features is completely the same. The multiple features can be seen as one large feature. Otherwise it is advised to implement the features for one level of detail after each other, as has been done in the first two design cycles.

The preparation phase has been used in a different way

During the preparation phase of the Segway, the assumption has been made that all the necessary models for the execution of the SDM were present. This assumption has not been made during the application on the production-cell. During the preparation phase, all the necessary information for the implementation of the ECS and the functionality of the event-based plant has been established. During some of the design cycles it was noted that specific modules, which support the ECS, were necessary and these were designed and tested in step *A* of these design cycles.

The preparation phase has thus been used for more steps than only “Order and split the features and levels of detail as preparation” (Broenink and Broenink, 2019).

Testing during the design cycles

In some situations it is not possible to test all the implemented functionality in a feature. This occurs when the simulation gets stuck due to missing functionality. This happens for example

in the feature of the Molder, where it cannot be tested if the Molder door closes again correctly. This is due to the signal from the Extractor that was not yet implemented and is used as a condition to close the Molder door.

In this case small functionalities could have been added that mimic the implementation of the Extractor and that can be used to test the complete behavior of the Molder door. This is extra work but give more information about the implementation of the Molder door. Another way is to just proceed and test the closing of the Molder door in the next feature. A balance has to be found in this.

The implemented functionality that cannot be tested during the design cycle can be identified in the preparation phase. During the design of the FSM's, the implementation order of the features and levels of detail is known. It can be seen in the FSM if a signal is present that will be implemented in a later feature. At that point extra functionality is necessary to mimic behavior of the next feature. This holds also for the FSM's that have been designed during the preparation phase of the production-cell. It can be seen in the FSM of the Molder, in Figure 4.7, that external signals are necessary for closing and opening the Molder door. This signal comes from the Extractor which is the next feature.

During testing of the implementation of the second level of detail some problems were encountered with the implementation. In this design cycle some extra functionality was added which mimicked functionality of the discrete-time plant, but this did not worked correct, but the overall behavior was correct. This level of detail was used as a transition from unitless steps to discrete-time steps. Due to the positive overall behavior in the simulation results it was decided to proceed.

From this decision it can be derived that sometimes it is more convenient to proceed to the next level of detail and implement missing or repair incorrect functionality later.

The second level of detail, where discrete-time steps are implemented, has not been very efficient because there is moved to the next design cycle with incomplete functionality. It could have been better to change the requirements and thus the functionality implementation in this design cycle. When only the implementation of discrete-time steps would have been done and no changes to the event-based plant. But instead in the next design cycle the changes to the event-based plant would have been performed in multiple design cycles for the third level of detail. The steps would have been more clear and the next step could have been made with a complete and working simulation.

The proposed event-based plant in Section 3.2, has helped to test the control logic without the presence of a discrete-time plant. This plant provided the ECS with signals that are used to make the discrete-event control decisions. This event-based plant has been a succes during the implementation of the ECS for the production-cell. It provided a implementation of the discrete-time plant when it was not available yet.

Other tests that have been performed are the small functionality tests on the modules that are designed and that is implemented and tested before a design cycle is started. When well-tested modules are integrated it is assumed that the implementation is easier because less problems can occur. In this case no problems on the modules can be encountered but only implementation errors of the modules into the ECS.

To perform good tests, the functionality that must be tested cannot be too complex. Otherwise it is hard to detect errors in the implementations. Therefore is it advised to make the levels of detail small enough such that tests are effective. This can be seen clearly in the implementation of the control logic. In that design cycle the control logic is implemented seperatly for the features, but at the end of the last feature a seperate test is performed for the complete behavior. If the levels of detail would have been implemented for all the features at once, it would have been hard to detect errors.

5.2 Evaluation on requirements of SDM

Evaluation of the stakeholder requirements

The requirements that are stated by the stakeholders are on traceability of the design process, a general applicable method such that it can be used throughout the design teams, a design error detection mechanism and it must be easy-to-use.

1. **Traceability**
   During the execution of the SDM, and especially in the preparation phase, it has been observed that no real design decisions have been made. This is partly because the production-cell did already exist and no electric motor had to be selected for example. The order in which the Feeder, Molder and Extractor operate together was developed, and this differed from earlier implementations.

   No real conclusion can be drawn on the traceability of the Structured Design Method because no real design decisions have been made. The development of the Finite State Machines is a design decision and these are clearly stated and can be retraced. But this information is less of concern for maintainers for example.

2. **General applicability**
   The Structured Design Method has been well applicable to the production-cell. Only the scheme in Figure 1.1 did not have been followed completely due to different dependencies. But with a different order, the SDM can be applied to the combination of a continuous-time plant and a discrete-time controller and the combination of an event-based plant and a discrete-event controller.

3. **Design error detection mechanism**
   During the execution of the Structured Design Method, an error in the design of the FSM Feeder Belt have been detected. This detection did not occur during the implementation of this level of detail but during the implementation of a later level of detail. Therefore it has not been detected in an early stage but two levels of detail later. The late detection of the design error is also due to the lower level of detail that has been used during the implementation of the control logic level of detail. Because the features and levels of detail are independent, no complete redesign was necessary. When designing control logic, not all the functionality is known during the preparation phase and it is therefore hard to make a complete design. Due to independencies between the features, the design error detection mechanism is of less concern than in for example control laws where all levels of detail are dependent and the design error detection mechanism saves resources during design.

4. **Easy-to-use method**
   This is a very abstract requirement, but can be answered in an qualitative way. During the application of the SDM, the steps that had to be taken were very clear. It was somewhat hard to determine what the step was in terms of *A*, *B* and *C* but this was due to the iterations over the levels of detail instead of the features. But overall the method was easy to apply to the production-cell.

Evaluation of the SDM application aspects

The requirements that are stated about the application of the SDM are on extensibility, interchangeability and easy integration of discrete-event and discrete-time systems.

1. **Extensibility**
   The extensibility of the software architecture during the application of the SDM has been

well supported by the framework. It was easy to add new modules and to test them. Depending on the usage of different software architectures for the ECS this can be different, but in the usage of this simple framework the extensibility was good.

2. **Interchangeability**
   The interchangeability of the software architecture was not as expected during the application of the SDM. Functionality had to be changed and added in a later stage and this was not very supportive for the interchangeability. The usage of the ZMQ message library could be helpful, this is described in Section 5.1.

3. **Easy integration of discrete-event systems and discrete-time systems**
   The integration of the discrete-event controller and discrete-time plant did go smooth. The dummy plant tool helped in the integration of the plant in the ECS and changing the parameters to the real plant parameters was easy. To test the discrete-event controller without the presence of any discrete-time plant or dummy plant, the event-based plant provided a mechanism for this.

6 Conclusion

6.1 Conclusion

The research objective of this project is to investigate the applicability of the Structured Design Method for the design of Electronic Control Software for physical systems that contain discrete-events. An analysis has been performed from the point of view of the stakeholders on the SDM and on the aspects that are of interest during the application of the SDM on the design of ECS for discrete-event controllers.

A case study has been performed to test the application of the SDM to the design of ECS for a production-cell setup which consist of a physical plant that is controlled by a discrete-event controller.

It can be concluded from the analysis that the applicability of the SDM on discrete-event systems depend on the extensibility of the software architecture of the ECS under design and the interchangeability of the architecture with the virtual and the real plant. Also the integration between discrete-event control logic and discrete-time loop control is of importance because this is where different components which are designed, often concurrently, come together.

It can be concluded from the case study that the Structured Design Method is suitable for the design of ECS for an physical system containing discrete-events. The order in which the features and the levels of detail are implemented is only adapted and multiple features can be implemented for a single level of detail in one design cycle.

Using tools that can help during the implementation, such as the ZMQ message library, dummy plant and event-based plant, an efficient implementation can be achieved with a small number of redesigns and an easy merging of the discrete-event control logic and the discrete-time loop control.

6.2 Future Work

To further extend the applicability of the SDM, it can be usefull to use the SDM in the development phase of a CPS. When the SDM would be tested for the applicability to answer design questions, the SDM would be applicable in more stages of the design of a CPS.

An example of a project that can be used to test the SDM during the development phase is making the mechanical design of a robot arm. When design requirements are given on budgets, so for example the mass to handle, the motion profiles and motion accuracy, one can use this information to make optimal design decisions on the motor specifications or arm stifness for example to make a complete design. The general idea is to use the SDM go from budgets to a mechanical or electrical design.

A Demonstration instructions

Follow the steps below to start the demonstration on the production-cell. To run the demonstration on the production-cell demo setup, seperate programs on the NUC and the two RaMstix boards have to be started.

Preparation steps:

1. Make sure the right controller inputs are selected, do this by setting the dip-switches on the different multiplexing boards. Dip-switch 1 must be ON and dip-switch 2 must be OFF.
2. Power up the production-cell power supply, the two RaMstix boards and the NUC.
3. Boot the RaMstix boards and the NUC
4. Login to RaMstix board #1 with the NUC in a command window. Login with the command: *ssh ram@192.168.33.101* with password: *ram*.
5. Login to RaMstix board #2 with the NUC in a seperate command window. Login with the command: *ssh ram@192.168.33.103* with password: *ram*.
6. Navigate to the folder in a seperate command window which contains the file *GUI_production_cell.py*

Starting the demonstration:

1. Run the binary *main_101* on the RaMstix board #1 with the command: *sudo ./main_101*.
2. Run the binary *main_103* on the RaMstix board #2 with the command: *sudo ./main_103*.
3. Run the GUI on the NUC in the command window with the command: *GUI_production_cell.py*
4. Now the production-cell can be initialized in the GUI by pressing INIT. When the initialization has been performed, the production-cell can be started by pressing START. During production the process can be stopped by pressing STOP. The PAUSE button has not been implemented. In Figure A.1 the GUI is shown

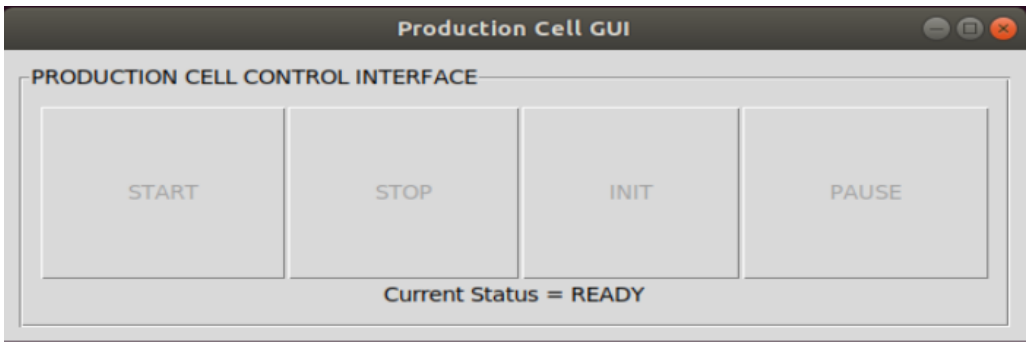


Figure A.1: Overview of the GUI that can be used to initialize, start and stop the production-cell.

B Hardware configuration

This chapter gives an overview of how the electrical connections are between the two RaMstix boards and the production-cell mechanics.

RaMstix board #1 login: ssh ram@192.168.33.101, password: ram			
Molder Motor encoder	Input	ENC 3	
Molder Motor PWM	Output	PWM 1	
Molder door end-switch	Input	Pin 15	Active high
Extractor Motor encoder	Input	ENC 2	
Extractor Motor PWM	Output	PWM 0	By a negative input the Extractor moves toward position P4
Extractor end-switch front	Input	Pin 9	Active low
Extractor end-switch back	Input	Pin 10	Active low
Extractor magnet	Output	Pin 11	Active low
Feeder Belt Motor encoder	Input	ENC 1	
Feeder Belt Motor PWM	Output	PWM 3	By a negative input the belt moves the block to the Feeder
Feeder Motor encoder	Input	ENC 0	
Feeder Motor PWM	Output	PWM 2	By a positive input the Feeder moves toward position P4
Block detection sensor, position P1	Input	Pin 6	Active low
Block detection sensor, position P6	Input	Pin 8	Active low
Block detection sensor, position P7	Input	Pin 7	Active low

Table B.1: Overview of the electrical connections to RaMstix board #1

RaMstix board #2 login: ssh ram@192.168.33.103, password: ram			
Extractor Belt Motor encoder	Input	ENC 1	
Extractor Belt Motor PWM	Output	PWM 3	By a negative input the belt moves the block to the Rotator
Rotator Motor encoder	Input	ENC 0	
Rotator Motor PWM	Output	PWM 2	By a negative input the Rotator moves the block to position P1
Rotator end-switch start	Input	Pin 2	Active low
Rotator end-switch end	Input	Pin 1	Active low
Rotator magnet	Output	Pin 3	Active low
Block detection sensor, position P2	Input	Pin 7	Active low
Block detection sensor, position P3	Input	Pin 6	Active low
Block detection sensor, position P5	Input	Pin 14	Active low

Table B.2: Overview of the electrical connections to RaMstix board #2

In the overview in Figure B.1, the electrical connections are given in the model of the production-cell.

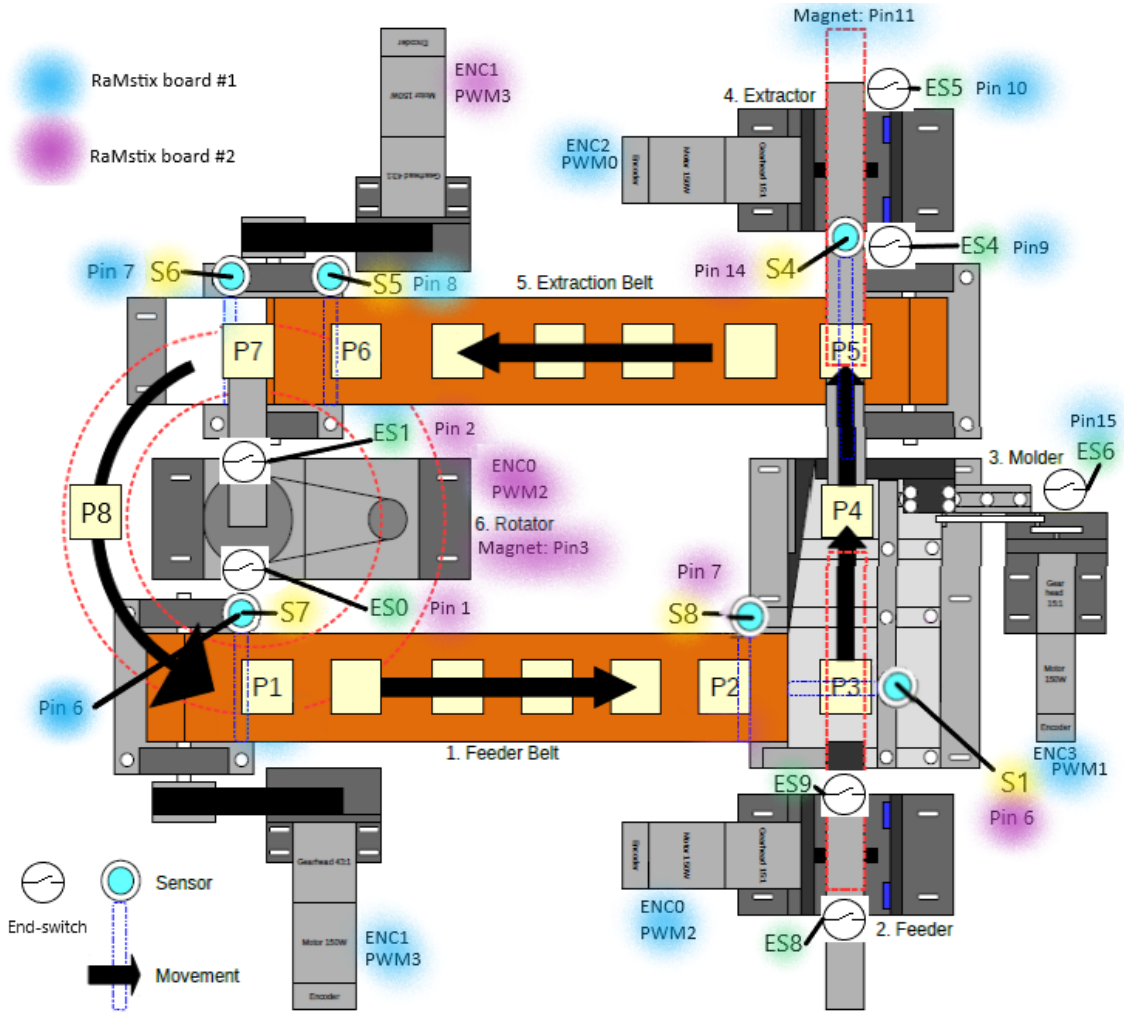


Figure B.1: Overview of the production-cell with the electrical connections to RaMstix boards.

C Design method log form

In this appendix the log form, which is used to log the Structured Design Method design cycles is described and shown. The columns of the form contain the feature number, the design cycle, the feature, the level of detail, a description on the execution of the design cycle and the models/documents that apply to the design cycle. As an example the first design cycle of the Feeder Belt is given in the form.

Implementation of features

Project: Production cell demo setup
Date: 18-05-2020
Author: K.J.C. den Hollander
Document No:

Description: In this document the implementation has been logged of the Feeder Belt.

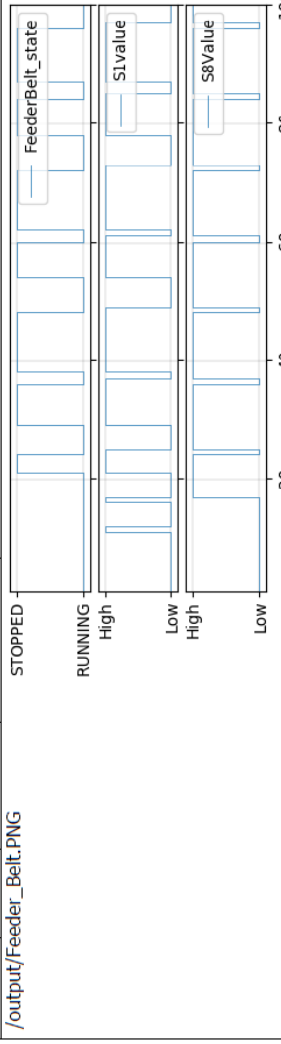
Feature No.	Design cycle	Feature:	Level of detail:	Description/Notes:	Applicable models/documents
1.	A_1	Feeder Belt	Control logic implementation	For this first feature not only the controller and the plant, but also the simulation framework is implemented which can be used during the implementation. The feature is tested by analysis of the output of the simulation to see if the Feeder Belt stops rotating when blocks are seen by sensor S8 on position P2 and sensor S1 on position P3.	
1.	B_1_1	Feeder Belt	Control logic implementation	The Finite State Machine of the Feeder Belt is implemented.	/output/Feeder_Belt.PNG /Code/iteration_1/Feeder_Belt/run
/output/Feeder_Belt.PNG					
1.	C_1_1	Feeder Belt	Control logic implementation	It can be seen in the figure above that when both sensor S1 and sensor S8 detect a block, the Feeder Belt changes to the stopped state. When a block is removed from position P3, the belt starts rotating again. The desired behavior has been implemented.	/output/Feeder_Belt.PNG /Code/iteration_1/Feeder_Belt/run

Figure C.1: Form that can be used to log the design cycles in the Structured Design Method.

Bibliography

van den Berg, B. (2006), *"Design of a Production Cell Setup"*, Msc report, University of Twente.

Broenink, J., P. Vos, Z. Lu and M. Bezemer (2016), A co-design approach for embedded control software of cyber-physical systems, in *2016 11th System of Systems Engineering Conference (SoSE)*, IEEE Circuits and Systems Society, pp. 1–5, ISBN 978-1-4673-8727-9, doi:10.1109/sysose.2016.7542927, 10.1109/sysose.2016.7542927.

Broenink, T. and J. Broenink (2018), A variable detail model simulation methodology for cyber-physical systems, in *Proceedings of the 32nd European Conference on Modelling and Simulation ECMS 2018*, pp. 219–225, ISBN 978-0-9932440-6-3.

Broenink, T. and J. Broenink (2019), Rapid development of embedded control software using variable-detail modelling and model-to-code transformation, in *Communications of the ECMS*, volume 33, pp. 151–157, ISBN 978-3-937436-65.

Controllab Products (2020), 20-sim.
https://www.20sim.com/

Fitzgerald, J., C. Gamble, P. G. Larsen, K. Pierce and J. Woodcock (2015), Cyber-Physical Systems Design: Formal Foundations, Methods and Integrated Tool Chains, in *2015 IEEE/ACM 3rd FME Workshop on Formal Methods in Software Engineering*, pp. 40–46, doi:10.1109/FormaliSE.2015.14.

Gumstix (2020), Overo series.
https://www.gumstix.com/support/hardware/overo/

Isermann, R., J. Schaffnit and S. Sinsel (1998), Hardware-in-the-loop simulation for the design and testing of engine-control systems.

Lee, E. A. (2008), Cyber Physical Systems: Design Challenges, Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley.
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html

Mavin, A. and P. Wilkinson (2010), Big Ears (The Return of "Easy Approach to Requirements Engineering"), in *2010 18th IEEE International Requirements Engineering Conference*, pp. 277–282, doi:10.1109/RE.2010.39.

Meijer, M. (2013), *"Finite State Machines and the Embedded System Design Flow"*, Msc report, University of Twente.

Morsinkhof, N. (2019), *"Testing the structured method to design embedded control software"*, Bsc report, University of Twente.

NWO-AES (2018), *Final Report Perspectief Programme Robust design of Cyber-Physical Systems(CPS)*, Netherlands Organisation for Scientific Research, Applied and Engineering Sciences.

van de Ridder, W. (2018), *"Improvements to a tool-chain for model-driven design of Embedded Control Software"*, Msc report, University of Twente.

ZeroMQ (2020), An open-source universal messaging library.
https://zeromq.org/