A Model Based Design approach to improve the design procedure of the Tubtrax Baggage Handling System

Hanssen, Marijn W.M.

Award date:
2022

A Model Based Design approach to improve the design procedure of the Tubtrax Baggage Handling System

Master graduation project

Master:	Manufacturing Systems Engineering - Mechanical Engineering
Department:	Mechanical Engineering
Research group:	Dynamics and Control
Report number:	DC 2022.058

Author:	M.W.M. Hanssen (0900580)
Internal supervisor:	prof.dr.ir. I.J.B.F. Adan
Internal supervisor:	dr. A.Y. Pogromskiy
External supervisor:	ir. J.J.M.J.P Meens
External supervisor:	ing. V.J.M. Meijer

Veghel, Thursday 1st September, 2022

Abstract

This graduation project is conducted at Vanderlande to investigate how the current design procedure of arbitrary functionalities of the Tubtrax baggage handling system can be improved. The current design procedure has multiple disadvantages that complicate the design process and makes this design approach unattractive. Therefore, a model based design procedure is proposed to circumvent these difficulties. With this approach, a simulation framework is developed that enables the possibility to define arbitrary configurations via a quick procedure and offers the ability to simulate these configurations, implying that the calculations are no longer performed by hand. This saves time and prevents that human calculation errors are made. Moreover, this framework offers the ability to post-process configurations, enabling the possibility to study the robustness of a design (concerning the positions of speed change or dieback points and varying controller sample times). Additionally, the manual design procedure has no confirmation that the manual-defined solution is optimal. Therefore, a feature that finds a satisfactory set of design parameters is missing and simulation based optimization (SBO) algorithms are studied. The hybrid genetic algorithm is identified as the most suitable SBO algorithm for the current project. To solve the problem concerning the missing optimization feature, an optimization framework is constructed that offers the possibility to solve an optimization problem and find a set of design parameters that meet the requirements and do not violate the constraints. Herein, two fitness functions are defined that shall guide solving the optimization problem. The first fitness function considers the requirement, capacity and constraints of the system. The second fitness function considers the cost of the configurations. A resourceful procedure is defined that starts its search at the most promising configuration (based on its costs) and proceeds to the next whenever no satisfactory solution is found. In the end, the developed optimization framework is used to successfully improve two use cases.

Contents

List of symbols

Symbol	Description
a	Acceleration
c_{conveyor}	Cost value of a single conveyor
$c_{\text{footprint}}$	Cost value per meter
C_{elite}	Elite chromosome
$C_{\text{neighbour}}$	Neighbour chromosome
CI	Confidence interval
d_{attained}	Attained gap
d_{desired}	Desired gap
$f(\vec{x})$	Fitness function
f_{best}	Best fitness value
$f_{\text{neighbour}}$	Fitness value of the neighbour
$g_i(\vec{x})$	Inequality constraint i
$h_j(\vec{x})$	Equality constraint j
k	Total number of decision variables
$K(\vec{x})$	Capacity function
l	Total number of equality constraints
m	Total number of inequality constraints
max_{LS}	Maximum number of no improvement iterations during the local search operator
n	Number of measurements
n_f	Number varied factors
n_r	Number of replications
n_s	Total number of used sections a of configuration
num_{LS}	Current number of no improvement iterations during the local search operator
N_p	Population size
p	Acceptance probability
p_c	Crossover rate
p_e	Elitism rate
p_m	Mutation rate
$P(\vec{x})$	Penalty function
P_{factor}	Penalty multiplication factor
P_{total}	Absolute penalty value
$S_{\text{lengths},i}$	Length value of conveyor i
S_n	Sample standard deviation of measurement 1 up and until n
t_{brake}	Brake time
$t_{n-1,\alpha/2}$	Student t-distribution for $n-1$ degrees of freedom and a significance level of $\alpha/2$
$T_{\text{controller}}$	Cycle time of the PLC
T_{current}	Current temperature value
T_{update}	Updated temperature value
v	Velocity
x_i	Decision variable i
$\vec{x}$	Vector with all decision variables
$\bar{X}_n$	Sample mean of measurement 1 up and until n
α	Significance level
β	Factor that decreases the temperature value of the SA algorithm with a specific rate
$\delta_{\text{interface}}$	Distance between a decision and a dieback point
Δf	Difference between the fitness value of the neighbour and the current best solution
μ	Capacity of the system in seconds per carrier

List of abbreviations

Abbreviation	**Meaning**
BHS	Baggage handling system
DOE	Design of experiments
DVA	Desired velocity attained
FIFO	First in first out
GA	Genetic algorithm
GUI	Graphical user interface
HGA	Hybrid genetic algorithm
ICS	Individual carrier system
KPI	Key performance indicator
MBD	Model based design
PLC	Programmable logic controller
PSE	Pseudo standard error
SA	Simulated annealing
SBO	Simulation based optimization
SKU	Stock keeping unit
TICO	Toyota Industries Corporation

Chapter 1
Background and problem statement

This chapter includes an introduction of Vanderlande Industries B.V.. The history of the company and the core business segments are briefly discussed. Furthermore, the system that is considered is elaborated. Moreover, the problem is discussed that is researched during this thesis and an illustrative example design is presented. Thereafter, the problem statement is manifested. Herein, the scope and the current design methodology are detailed. Moreover, the shortcomings of the current design procedure are highlighted. In the remainder of the chapter, the goals and research questions of the graduation project are formulated.

1.1 Vanderlande Industries B.V.

This graduation project is conducted at Vanderlande Industries B.V. (in the remainder of the thesis referred to as Vanderlande). The company is the global market leader for future-proof logistic process automation at airports. Moreover, Vanderlande is also a leading supplier of process automation solutions for warehouses and in the parcel market. Vanderlande's baggage handling systems are capable of moving over 4 billion pieces of baggage around the world per year. Its systems are active in more than 600 airports, including 12 of the world's top 20 biggest airports. Additionally, more than 52 million parcels are sorted by its systems every day. Furthermore, many of the largest e-commerce retailers have confidence in Vanderlande's efficient and reliable solutions [1].

1.1.1 History of Vanderlande

Vanderlande is established by Eddie van der Lande in 1949. The core business at that time was to refurbish and later produce machines for the textile industry. Thereafter, the company moved to the manufacturing of hoisting apparatus, cranes and conveyor belts for bulky materials and barrels of oil. The first revolution of the company came in 1963 when Vanderlande entered into a partnership with an American company called Rapistan Incorporated. After the fusion, Vanderlande began to develop and manufacture customized transportation systems. This change of scope and collaboration marked the start of a successful global organisation. In 1988, NPM Capital acquired a majority of the company's shares. This has enabled Vanderlande to achieve its long-term goals and expand on an international scale. The most significant change in management occurred in 2017. In 2017, Toyota Industries Corporation (TICO) acquired Vanderlande. The cooperation between both companies exists with the cross-selling of products and solutions; product innovations; and joining forces in research and development. To date, Vanderlande is present in each of the world's growing market segments. The company has grown from a machinery supplier to a highly reliable partner that develops and produces logistic process automation solutions [2].

1.1.2 Logistic automation solutions

Vanderlande splits its core business into three market segments, i.e. warehousing, parcel and airports. Each of the market segments is briefly discussed in the text here below.

Warehousing market segment

The warehousing market segment is the first market segment of Vanderlande. This segment is growing every year and is involved in the e-commerce, fashion and food retail companies. The growth of this market segment resulted in an increasing demand for warehousing solutions.

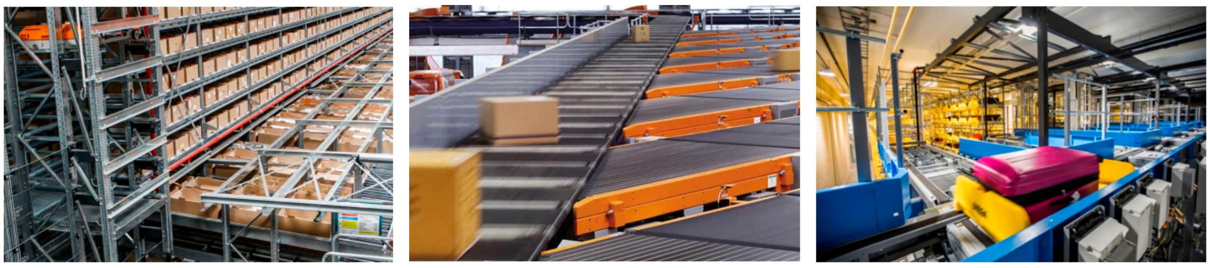

(a) Warehousing. (b) Parcel. (c) Airports.

Figure 1.1: Three core business market segments of Vanderlande [6].

These customers demand flexible solutions concerning handling the levels of orders and SKUs (Stock-Keeping Units); fulfilling online orders and store deliveries; and coping with a lack of available workforce. A part of a warehouse developed by Vanderlande is presented in Figure 1.1a. These solutions are integrated into various warehouses belonging to, for example, Alibaba. Such companies need to continuously improve their supply chains and invest in innovative technologies. Therefore, these companies require a reliable partner that delivers future-proof logistic process automation solutions. Vanderlande acts therefore accordingly. Vanderlande understands the complexities associated with operating a successful warehouse operation and can help to exceed the customers' expectations and improve their competitive position [3].

Parcel market segment

The parcel industry is continuously expanding, due to an increase in online orders. As a result, meeting service level demands is becoming increasingly tough. Parcel customers demand shorter lead times, later cut-off times, smaller orders and increasing product diversity. These demands increase the complexity of the systems considerably. Moreover, the need for maximum delivery reliability, in combination with zero systems faults, is one of the most important KPIs (Key Performance Indicators).

To satisfy these demands, Vanderlande strives for the improvement of operational activities and the expansion of logistical achievements. Vanderlande does not believe in standard designs and has therefore an extensive portfolio of integrated solutions with innovative systems, intelligent software and life-cycle services. The combinations of these aspects result in fast, reliable and efficient automation technology [4]. An example of a part of a parcel solution is illustrated in Figure 1.1b. Vanderlande's parcel solutions are integrated for world-leading parcel companies, including UPS, DHL, FedEx and DPD.

Airports market segment

The airport's division is the third and last market segment of Vanderlande. Vanderlande can deliver an entire airport's solution, ranging from passenger to baggage handling. To exceed customers' expectations, optimize the passenger experience and enhance the airport's proposition to its stakeholders, Vanderlande has established many successful partnerships. As well as in the parcel market segment, Vanderlande does not believe in standard designs. Therefore, Vanderlande has an extensive portfolio of integrated solutions, including innovative systems, intelligent software and life-cycle services that eventually result in fast, reliable and efficient automation technology [5]. An example of a part of an airport baggage handling solution is illustrated in Figure 1.1c. Vanderlande's airport solutions are integrated into, for example, Schiphol Amsterdam, LAX Los Angeles International Airport and Heathrow Airport London.

1.2 Tubtrax baggage handling system

The Tubtrax Baggage Handling System (BHS) is one of the main systems Vanderlande develops. Tubtrax is Vanderlande's Individual Carrier System (ICS) for fast and high-volume transport. The BHS processes all baggage in an airport. The system checks in, screens, transports, sorts and stores all in- and outgoing baggage. A flow diagram of the BHS is presented in Figure 1.2.

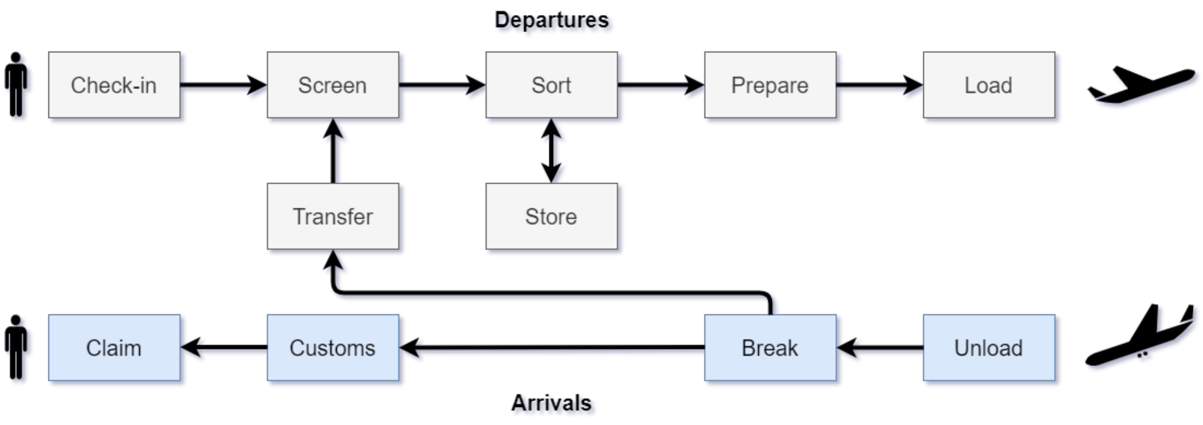


Figure 1.2: High-level flow diagram of the BHS.

The system is a carrier-based system that brings uniformity to the baggage handling process, enabling a more reliable operation, higher speeds and higher throughput. Each transported bag is transported in a carrier. A series of carriers loaded with baggage is presented by Figure 1.3. Each bag stays entirely within its carrier throughout the transportation through the Tubtrax system. As a result, the risk of damaged bags and system interruptions is negligible. Moreover, this individual carrier system enables continuous tracking of baggage items, which minimizes the number of mishandled baggage items. The system includes a choice of bag store options, which enables fast bulk or individual storage, retrieval and batch building capabilities [5].

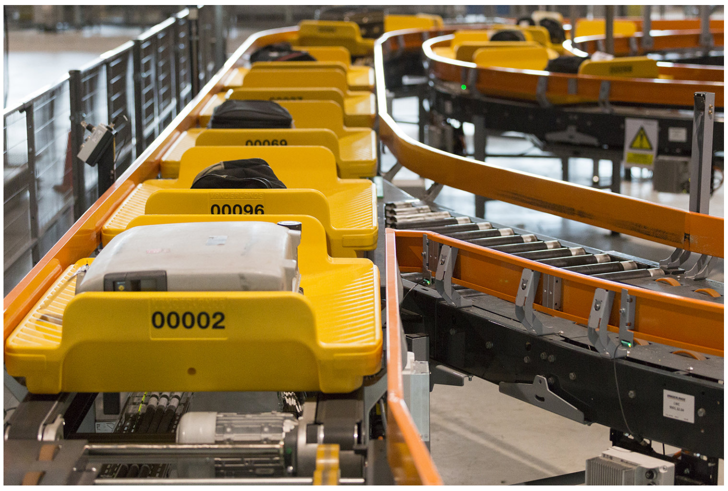
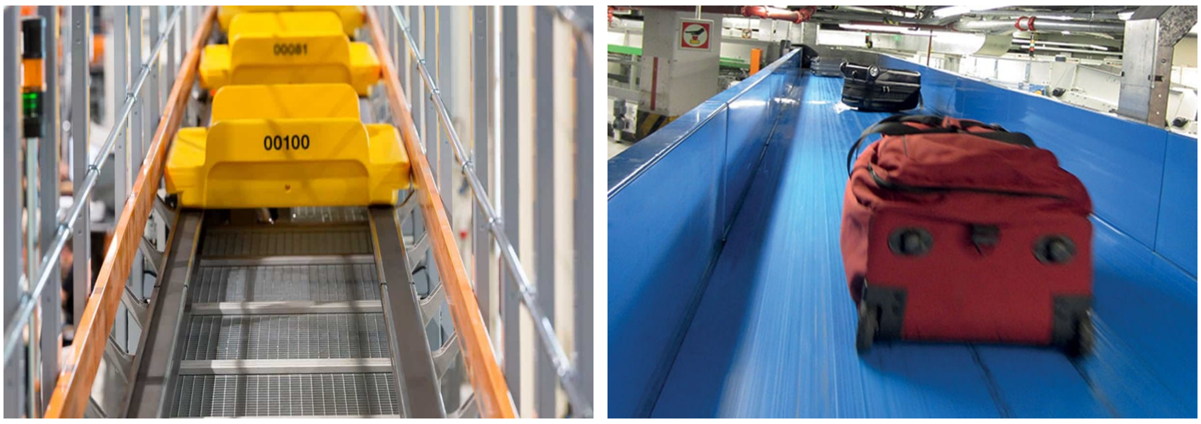


Figure 1.3: Yellow carriers with baggage used in the Tubtrax system.

The BHS is an advanced and flexible system that can fulfil numerous functions. The BHS is built out of a set of building blocks, i.e. integrated system components. The functions of the system can be categorized into three fundamental categories, i.e. straight transport; sorting and merging; and loading and unloading. The main characteristics of the functions of each category are discussed in the remainder of this section.

1.2.1 Straight transport functionality

The first main function that is fulfilled during the baggage handling process is the transportation of baggage items. This process is executed via different principles, i.e. carrier-based or high-speed ICS to transport a bag individually or via efficient conveyor belts to transport raw baggage. A suitable solution is defined by a combination of technical and operational considerations. During the design process of an appropriate solution, the required system capacity, baggage type and system constraints are considered to define a future-proof solution. The individual carrier transportation system is illustrated by Figure 1.4a and the raw baggage conveyor is illustrated by Figure 1.4b.



(a) Individual carrier system transportation. (b) Raw conveyor transportation.

Figure 1.4: Transportation systems Vanderlande [6].

1.2.2 Sorting and merging functionality

The second main function that is fulfilled during the baggage handling process is the sorting and merging process of baggage items. These functions are the fundament of the system concerning the routing of baggage items through the system. These functions make sure the baggage items end up at their intended destination within the airport. The sorting process can sort a carrier and route this carrier to a specified destination at the airport. Furthermore, the merge process merges a carrier into the system, like a merging line on the highway. A merge section is depicted in Figure 1.3.

1.2.3 Loading and unloading functionality

The third main function that is fulfilled during the baggage handling process is the loading and unloading process of baggage items. This function is fundamental for the BHS as well. The loading function loads baggage items on the carriers that are transported through the system. Moreover, the unloading function unloads baggage items from the carrier. The unloading process is carried out whenever a piece of baggage arrives at its final destination within the airport.

1.2.4 System design

The Tubtrax baggage handling system is composed of individual conveyors that can be connected in an arbitrary order. An example configuration (top view) is depicted in Figure 1.5. The most fundamental conveyor type of the Tubtrax system is the twin belt section. This conveyor type does not sort or merge streams of carriers but only applies the straight transport functionality. Each conveyor has its individual properties, i.e. length, a dieback point, a decision point and optionally, a speed change point. The dieback point represents the position on a conveyor where a carrier stops in a controlled procedure. Whenever a carrier passes the decision point, the system's controller determines if the passing carrier is required to go in dieback or is allowed to proceed to the downstream conveyor. The system makes this decision based on multiple configurable conditions. The speed change point is responsible for the speed change functionality. Whenever a carrier passes this point (and is allowed to proceed), the corresponding conveyor changes its velocity. The presented example possesses a second conveyor type, i.e. a merge section. As mentioned before, merge and sort sections have the functionality to merge and sort streams of carriers. These types of conveyors have the same fundamental properties as twin belt sections. The presented configuration is just an example. Different configurations of the system could satisfy imposed requirements and constraints as well. Therefore, a satisfactory and feasible solution should be found that meets these requirements and does not violate the constraints.

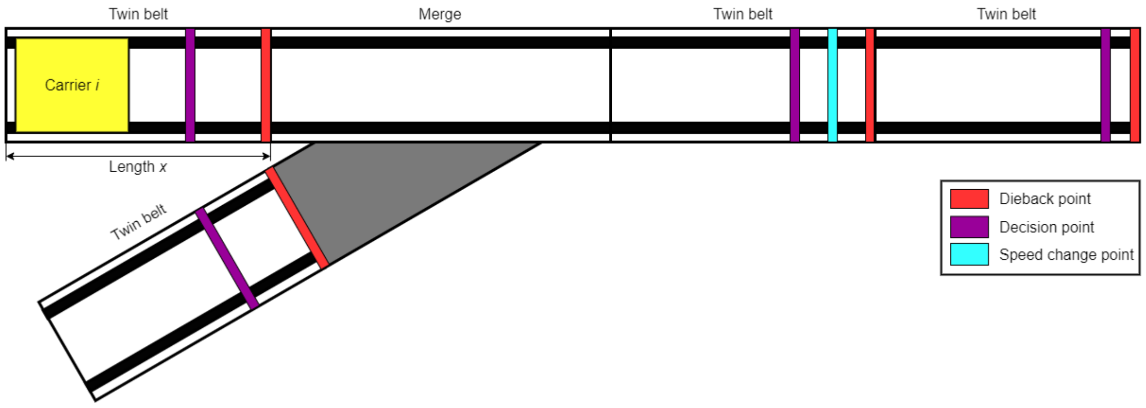


Figure 1.5: Schematic representation of an arbitrary Tubtrax configuration.

1.3 Problem statement

Baggage handling system solutions are different for every customer. Such a solution is constructed with the help of building blocks. Most of these blocks are standardized. However, some are customized. Using a standard or customized building block depends on the requirements and constraints of the requested solution. Each block has its own task, subjected to functional requirements and constraints. A high-level functionality is a segment of the design solution and is composed of one or more tasks. Additionally, this functionality describes a certain behaviour that is expressed by the system and possesses design requirements. These design requirements are translated into lower-level design requirements belonging to the corresponding building block. The system is constructed via this procedure to make the solutions modular. Standard building blocks can be reused in different design solutions (subjected to minor configuration changes). Modularity eases the job of the system designer. The responsibility of the system designer is thereafter to configure all functional building blocks accordingly (i.e. develop standardized or customize a specific building block for a customer). All tasks are organized such that the lower-level design requirements and constraints are met and satisfied.

The described design framework can best be illustrated via an example, see Figure 1.6. The upper part of this figure presents the system that is designed based on the high-level functionalities, i.e. the orange three-dimensional blocks. The functional building blocks represent a functionality, e.g. the screening functionality. The screening functionality is constructed of the following consecutive tasks which are configured by the system designer, i.e. a gap creation task, a screening task and an acceleration task. The high-level design requirement of the functionality is that carriers have to be screened by the screening machine with a minimal relative gap of a specified distance. The high-level design requirement of the functionality is translated to lower-level design requirements belonging to the building blocks of the functionality. First of all, the gap creation task has to release carriers such that there is a relative gap of a specified distance. Secondly, the screening task has to translate carriers with a constant velocity. Lastly, the acceleration task has to release the carriers at their nominal velocity value again.

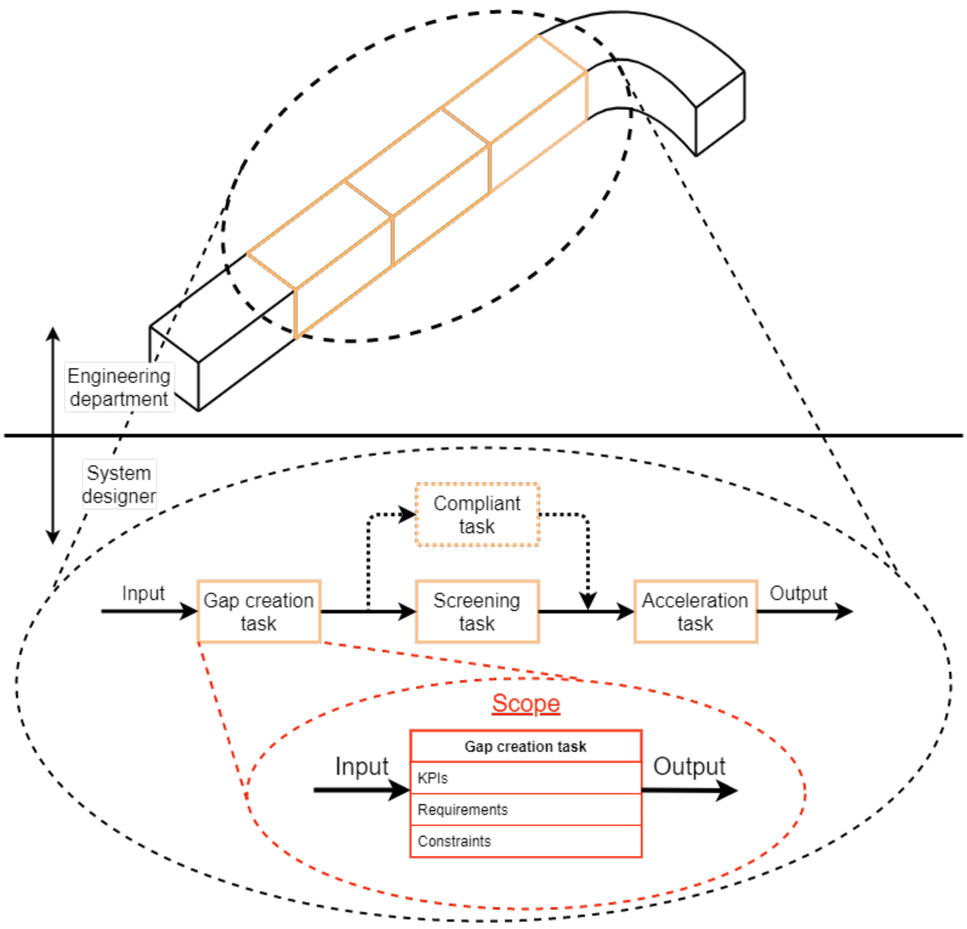


Figure 1.6: Screening functionality.

The scope of this research lies within the domain of the system designer. The system designer shapes the design solutions of the functional building blocks based on KPIs, requirements and constraints. Especially, the designer determines the mechanical configuration of the conveyors and objects (e.g. sensors). Moreover, the system designer specifies the velocity profiles of the conveyors. Currently, the mechanical layout and parameter specification is performed manually. All calculations, feasibility checks and requirement validation are also conducted by hand. Therefore, the current design methodology has multiple disadvantages that shall be tackled. These disadvantages are mainly the result of the iterative manual process. As a result, the design process is time-consuming; prone to human calculation and validation errors; and not flexible. It is not possible to observe automatically the results whenever parameters are changed, because the design has to be recalculated by hand again. The possibility to conduct a sensitivity analysis to study the robustness of the design is therefore not possible in the early stage of designing. Furthermore, it is not possible to automatically check the feasibility or to perform a requirement validation of a configured design. It would be time-efficient if the possibility arise to validate these facets automatically. Lastly, the system designer has no confirmation if the by hand defined solution for an arbitrary task is optimal. Therefore, a feature that finds a satisfactory set of design parameters for the design solution is missing.

1.4 Research questions

This research aims to develop a numerical tool that improves the design procedure of the Tubtrax baggage handling system by eliminating the manual iterative design process. Additionally, the numerical tool has to be capable of analysing the system's performance via the model (e.g. capacity, sensitivity analysis and requirement validation) of an arbitrary task or set of tasks. Furthermore, it should be possible to find a satisfactory set of design parameters for an arbitrary task that is subjected to requirements and constraints. To succeed, the framework's development is divided into two sequential steps. First of all, a simulation framework is developed that can analyze an user-defined design solution, i.e. user-defined structural layout and set of system parameters. Secondly, an optimization framework is developed that uses the simulation framework to evaluate defined solutions. The goal of the optimization framework is to find satisfactory solutions that meet and does not violate the imposed requirements and constraints.

Therefore, the research questions are formulated as:

1. How can model based design improve the design procedure of a task of the Tubtrax baggage handling system compared to the current manual design process?
2. Which optimization algorithms are most suitable to solve a simulation based optimization problem and how do they compare to each other (i.e. the number of evaluations and performance)?
3. How can model based design aid in the search for a satisfactory set of design parameters by optimizing the capacity for an arbitrary task that is subjected to imposed requirements and constraints; and how should the optimization framework be constructed?

1.5 Optimization challenges

Optimization studies are conducted since the definition of calculus. The most simple form is by trial and error. Engineers tried to find the best solution by evaluating different solutions and thereafter selecting the best among them. Another classical example is finding the best value for input variables given a convex function. For example, these kinds of analyses are conducted to find the maximum height that an object can reach after throwing an object in the air. The goal of this study is to find the optimal value for the angle given some input variables [7]. An important characteristic of these kinds of studies is that a closed-form expression of the problem is defined via mathematical expressions. These expressions are used to find the most optimal values for a given problem, depending on the requirements and imposed constraints. The main feature of these studies is that the derivative is used to study the system's response and to find appropriate input variables. This topic is used in classical optimization problems and further elaborated in section 4.2.

However, a closed-form expression regarding a certain system response cannot always be defined. Often, this is the case for complex dynamical stochastic systems. The responses of these systems are non-linear, have probabilistic elements and have a large number of random variables. As a result, defining a closed-form expression becomes cumbersome and too complex [8]. Therefore, a different technique is required to solve optimization problems that cannot express the system's response via closed-form expressions. Therefore, simulation based optimization methods are proposed. This optimization technique gains ground due to the increasing available computational power, offering the possibility to define a simulation model of the system's response and simulate it with different inputs. Then, this technique can identify satisfactory solutions. Simulation based optimization techniques are further elaborated in section 4.2.

1.6 Report layout

The remainder of the report is divided into two fundamental parts. During the first part, the first research question is answered. Herein is explained how model based design improves the design procedure of an arbitrary task of the Tubtrax baggage handling system compared to the current manual design procedure. Moreover, a literature review is presented concerning this topic. Thereafter, a simulation framework is developed to attain this goal. The results of the simulation framework are generated via a simulation. The simulation is a numerical model that is modeled via certain principles. In the end, the first research question is answered.

The second part of the report elaborates on the optimization research questions. In this part, a literature review of simulation optimization techniques is presented. This literature study is presented after the simulation framework is introduced because the simulation framework is required to solve optimization problems. The decision is made to divide the literature study into two parts to improve the presentation. Moreover, a proper simulation framework is required before the optimization techniques are discussed. Hereafter, suitable optimization algorithms are presented and a general conclusion is constructed concerning which optimization technique and algorithm are most suitable for the current optimization problem. Additionally, this conclusion is motivated by advantages, limitations and experimental results. Hereafter, the optimization framework is presented that is used to find a satisfactory set of design parameters and two use cases are solved to demonstrate the practicality of the optimization framework.

Chapter 2
Model based design methodology

This chapter introduces the transition from the traditional design methodology to a more sophisticated design methodology. Furthermore, the simulation model requirements are presented. Additionally, the purpose of the simulation model is scoped. The simulation model requirements, variables and purpose determine which simulation modeling technique is most suitable. Finally, different modeling techniques and programs are presented. The most suitable technique and program are motivated.

2.1 Design methodology transition

Systems that are designed are becoming more complex. The complexity increases due to the application of more functionalities, larger systems with more dependencies and systems that possess more possibilities. The design methodology is transformed from a traditional to a more sophisticated design methodology to keep the design process manageable, quick and feasible. The transformation arises from opportunities for improving performance, safety, and maintenance as a consequence of the increasing complexity of the designed systems. The increase in system complexity poses a significant challenge to the capabilities of the traditional design methodology [9]. The traditional design methodology is split into four phases. In the first phase, the definition and design of the objective system are defined. Herein, the system specifications are particularized. Secondly, the specified system specifications are implemented and the real system is built. Thereafter, the system is tested. The possibility may arise that the results do not meet the system requirements. This difference could be the result of an inappropriate definition of the system requirements or incorrect implementation of the system specifications. If this is the case, the design should be respecified or reimplemented and tested again. This iterative process is repeated up and until the desired results are realized. A suitable solution definition erupts only after extensive testing and evaluation. The complexity of this iterative process makes project management ineffective and burdensome [10].

The fundamental problem of the traditional design methodology may be the result of interpretation differences during the design and implementation phase. Additionally, not having the possibility to test implemented (sub)systems quickly, but only after it is implemented, is a major disadvantage that is time-consuming and costly. Furthermore, the traditional design methodology restricts the system's flexibility at a further stage in the development process. It is cost-inefficient to change certain aspects of a design if particular choices are already realized.

The model based design (MBD) methodology is developed to circumvent the difficulties induced by the traditional design methodology. It is a mathematically based method for designing complex control systems, by creating a mathematical representation of real-world systems. In the MBD methodology, the mathematical model is used throughout the design process. The model is the heart during this process. Furthermore, the model is used to formalize the system specifications; used to design; and used to test and verify the complex system all in parallel. This methodology integrates all phases and provides a generic approach for communication throughout the entire design process. Additionally, MBD assesses the integration of features into a larger system at each stage of the development, before the physical system is created. Using this methodology reduces the number of development iterations by combining the design, implementation and test phase into a single process [11].

The most important advantages of the model based design methodology are presented here below [11]. Designing a solution with the help of the model based design methodology is:

- *Flexible.* Design changes are made quickly and evaluated via experiments within the virtual environment;
- *Efficient.* This methodology provides the possibility for early design requirement validation, which decreases the development time and costs; and reduces dependence on physical prototypes. Iteratively building, destroying or making changes is time and cost-inefficient. Additionally, using the MBD methodology is convenient due to the major availability of computational power;
- *Accurate.* The created model represents the behaviour of the actual system. The accuracy is determined by applying the correct amount of abstraction. Only details that are believed to be important are included and this depends on the purpose of the model;
- *Modular.* Parts of a previous solution can be reused to define a new solution for a particular problem. This design process enables to define quickly individual solutions for particular problems.

2.2 Model specifications

This section presents the simulation model's specifications. These specifications include the model purpose and requirements. Herein, the goals of the simulation model are presented. This includes the purpose of the simulation. Additionally, the requirements of the simulation model include what the desired output of the simulation is and what features are desired in the model.

2.2.1 Model purpose

The purpose of the simulation model depends on the identified problems and shortcomings. The current design procedure is mainly conducted manually, implying that all calculations are conducted by hand. Hence, human calculation errors can be made easily and this procedure is time-inefficient. Furthermore, the current design procedure cannot evaluate the designed system's performance in an early design phase. The performance of this system is encapsulated by the throughput measure and the interrelationships between the carriers, like the gap between consecutive carriers. Additionally, the purpose of the current simulation model is to design systems that achieve certain requirements and that are restricted by certain constraints. These systems are developed by placing conveyors behind each other and specifying certain actions on these conveyors, i.e. defining their velocity profile. These actions are the result of the placement of dieback points, speed change points and interrelationships between carriers. Therefore, the system response depends on the behaviour of the carriers that are transported over the baggage handling configurations.

Dynamical systems are composed of a plant and a controller. The plant is responsible for the system's response which depends on its inputs. The plant is the physical system that is controlled. In the current case, the plant encloses the conveyors and the carriers. The controller is responsible for the control of the carrier flow on the conveyors by controlling the actuators based on sensor input and tracking. The controller sends control signals based on the properties or states of the plant's subsystems. For example, the controller determines if a carrier is allowed to proceed to the next conveyor whenever a carrier reaches the conveyor's decision point. Whenever a carrier is not allowed to proceed (due to any reason) the controller changes the velocity of the conveyor to zero. As a result, the carrier stops at the dieback point.

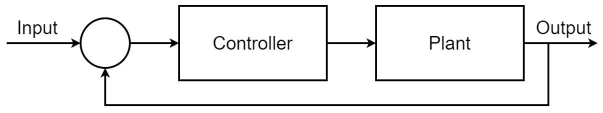


Figure 2.1: Feedback loop of the design mode.

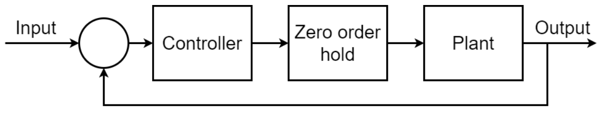


Figure 2.2: Feedback loop of the performance mode.

The simulation model has two modes, i.e. a design mode and a performance mode. The design mode produces the results via the same approach as the manual design process. Herein, the controller's and the plant's sample time is equal to each other. The sample time shall be chosen small enough to produce accurate enough results. The plant responds immediately whenever a certain threshold value is reached. The response is not delayed as there is no difference between the controller's and plant's sample time. This principle is illustrated in Figure 2.1. The system runs with a constant sample time. The sample time's value plays an important role in the simulation output. Herein, a trade-off shall be made between the precision of the simulation results and the actual calculation time. A study concerning the sample time is conducted in section 3.3. The design mode is useful in the validation process of the simulation model. The results obtained via the simulation model can be compared to the manually created results (by Vanderlande). Moreover, there is no deviation in the system's output due to the controller's influence. Therefore, the resulting capacity value is (in steady-state) always the same.

The performance mode enables the ability to analyze the system's response, in the early design phase, when a controller with a specific sample time is deployed. Within the process control domain, two sample times are used, i.e. control loop sample time and plant data sampling time. The control loop sample time regulates how often the controller determines what the controller output signal will be. Secondly, the plant data sampling time is the rate at which the output data of the plant is sampled. This principle is illustrated in Figure 2.2. The controller is sampled at a different rate than the plant. Therefore the controller output signal is saved via the zero-order hold principle. During this project, the plant's sample time is chosen small enough. The term "small enough" is arbitrary, therefore a sensitivity analysis is performed to determine this value. Again, herein a trade-off is made between computation time and performance. Moreover, the controller sampling time impacts the system response. This sample time is different at every site. This value cannot be chosen. Therefore, a system shall be designed that is able to produce satisfactory results with a varying controller sampling time. It is possible to study the effect of the system settings in an early design phase with the help of this performance mode. These results aid to accept a certain design or in further investigating the possibilities. The system response in the performance mode is not always the same due to the controller's influence. For this purpose, the carrier receives an offset in the position whenever a carrier is generated. This ensures that the response is distributed (a carrier arrives distributed at the first decision or speed change point). Therefore, the time required to complete a series of actions is different per carrier. Consequently, the capacity value is the average value of multiple carrier measurements.

2.2.2 Model requirements

As mentioned before, a simulation model is required to evaluate the system's response to develop a feasible and viable design. The requirements of the simulation model can be divided into two parts, i.e. simulation model output and features that the simulation model possesses. First of all, the simulation model's output is presented. With the help of the simulation model, the system's response can be mapped. These measurable output values are referred to as KPIs. The most fundamental KPI is the capacity of the system. The capacity value is an indicator that specifies the number of carriers that can be processed per time unit. Indirectly, different KPIs can be derived from the capacity value, e.g. the gap between carriers. For certain applications, the gap between the carriers is stated as a requirement. Therefore, what happens inside the system is of great importance. Secondly, the position of the individual carriers per moment in time is relevant. With the help of this data, it is possible to visualize the system's response and to check various design constraints.

Secondly, the simulation model should possess some modeling features. During the problem definition, different model features have been identified that shall be included as a requirement. The following features shall be included:

- The layout shall be modular. All conveyors can be configured in any order and variation, i.e. lengths;
- The parameters shall be modular. All input variables can be changed by the designer, e.g. dieback position, speed change position;
- It should be possible to implement customized functionalities;
- The simulation model should be able to determine the system's performance, i.e. KPIs;
- The simulation should be fast enough;
- The simulation model should be able to automatically validate requirements and design rules;
- It should be possible to implement a Graphical User Interface (GUI);
- The simulation model should possess the possibility to generate a visual representation of the system;
- It should be possible to implement an optimization framework to find a solution that satisfies the requirements and does not violate the constraints.

2.3 Modeling techniques

A mathematical framework is required to determine the system's response. This framework depicts the mathematical representation of the actual system and is required to construct design solutions for the imposed problems. As mentioned before, a model mimics a system by defining its behaviour via mathematical expressions and relations. Additionally, a mathematical model possesses a set of measurable variables. A subset of these variables may be varied and serve as input variables, i.e. the stimulus for the system. Another subset of the measurable variables is referred to as output variables and may directly be measured while the input variables are varied, i.e. the response of the system. Essentially, a mathematical model is an abstraction of a system. Therefore, the level of abstraction should be thoroughly considered as well. Abstraction results in loss of information but offers the ability to design and analyze a system. Moreover, abstraction provides flexibility and offers the ability to change the main characteristics with little changes to the model. Furthermore, abstraction reduces the computation and development time. The system should be studied thoroughly to determine what should be included in detail and what can be abstracted. Herein, as mentioned before, the compromise should be made between the precision of results and the speed of the simulation (chapter 1 of [12]) [13]. In the literature, different modeling techniques are found [13]. This section presents various fundamental modeling techniques which are used in practice. Additionally, a conclusion is drawn on which modeling technique is most suitable to construct the framework to solve the problem of this project.

2.3.1 Experimental design

As mentioned before, the traditional design method is mainly conducted in combination with experimental design. Herein, the final design is developed via prototyping and evaluation. After a design is created, an experiment is conducted on this physical system. Thereafter, the results are evaluated and the system is adapted accordingly. This process is repeated up and until the design satisfies the requirements. This design method became cumbersome over time due to increasing system complexity. Additionally, experimentation is not always possible for every system (e.g. large systems that do not exist yet) and could be time and cost-inefficient.

2.3.2 Analytical modeling

Two fundamental different techniques are formulated to define a model of a system. One of these techniques creates a model via differential equations, which uses proven laws and relations, i.e. formal modeling or analytical modeling. The system's data is obtained by experimenting with the system to identify system characteristics that are used to define the differential equations. The analytical model is thereafter validated by performing experiments on the system using the same inputs and boundary conditions. In general, using the analytical model results in precise outputs. However, finding the differential equations could be a problem. Especially, when systems become more and more complex.

2.3.3 Numerical modeling

Whenever the differential equations of the behaviour of a complex system are too hard to define, a numerical model can be used to simulate the behaviour of the complex system. The differential equations are harder to define whenever the system exhibits non-linear behaviour; has non-intuitive influences between variables; has time and causal dependencies; and when the system possesses a large number of parameters. These kinds of systems are approximated by discretizing their behaviour and their output values are determined via difference equations. This implies that the values of the variables are calculated per time step as opposed to the analytical model, which calculates the variables continuously. As a result, the precision is less compared to the analytical model's results. Herein, the precision of the numerical model depends on the level of abstraction.

A simulation is an executable model that possesses a set of rules that tells the model how to proceed from the current moment in time to the next moment in time. The rules can take many forms, including difference equations, state charts and process flowcharts. The output values of the numerical model are stored and used as the model executes. Numerical models have some advantages compared to analytical models. Firstly, a simulation model requires less effort to define. Secondly, the development process is scalable, incremental and modular. Thirdly, the output values of the numerical models can be used to perform statistical analyses or create animations for demonstration [14]. However, incorporating less detail could result in less accurate results but in general a faster model. The amount of detail that is included depends on the purpose of the system and model.

2.3.4 Queuing systems

Alternatively, a queuing system can be considered to model the behaviour of a manufacturing line. A queuing system can be considered as an architecture that is composed of a network of queues. This architecture allows for any number of servers, each with its individual queue, probabilistic and routing between servers. A queuing system is a stochastic model to study arrival and service processes; specifications of the structural parameters of the system (e.g. storage capacity of queue, number of servers); and specifications of the operating policies (e.g.

conditions under which arriving customers are accepted, serving policy) (chapter 8 of [12]). This architecture allows cycles in the network and is convenient for modeling manufacturing networks. Via this approach, the expected number of jobs in a system and the expected cycle time can be determined [15]. This modeling approach abstracts the system heavily. Herein, the detailed physical layout of the system is not of importance. It is cumbersome to include the effect of layout differences. Queuing theory is mainly used for scheduling and inventory control. Additionally, what is designed can be considered as a single workstation in the queuing theory. The performance of the entire BHS is not of importance during the design of the current system. Herein, the design of a specific operation inside the BHS is crucial. Therefore, the goal is to design a certain task (i.e. workstation) within the BHS as convenient as possible.

2.3.5 Model technique conclusion

The appropriate technique to model a specific problem depends on the model's purpose. Whenever the capacity value of the entire manufacturing line with a high-level of abstraction would be of interest, then the queuing network would be suitable. However, if more details shall be included and what happens inside the workstation is of importance, then a very detailed model of the system would be suitable. Therefore, identifying the purpose of the model is crucial to determine how much detail should be included and which modeling technique would be suitable for the problem.

Selecting the appropriate modeling technique depends on the system's complexity and the level of abstraction. The system exhibits non-intuitive influences between variables. The behaviour of the system is influenced by numerous rules and causal dependencies (e.g. events and constraints). Additionally, the system possesses numerous variables that influence each other. The goal of the framework is to design a satisfactory solution. Herewith, some level of detail is incorporated (e.g. conveyor lengths and velocity profiles of the conveyors). These aspects cannot conveniently be implemented via the queuing network model. A queuing model abstracts these crucial parameters. Moreover, creating an animation of the designed task is a cumbersome task due to this abstraction. The queuing model's purpose is to optimize a certain scheduling policy instead of designing the workstation in detail. Using the analytical model is also not convenient due to the system's complexity. Defining differential equations is a cumbersome task. Therefore, the behaviour of the system is simulated via a numerical model. Additionally, the framework shall not mimic the behaviour of the system precisely, i.e. including all losses. The goal is to design a suitable solution for a certain task. Therefore, some losses are not incorporated in this model, like losses due to friction.

2.4 Modeling program

Modeling dynamical systems can be done via various software programs. The decision to use a specific software program depends on multiple criteria. First of all, the software program should be able to model the current system. Secondly, using a single software program is favourable. Thirdly, the software program should be easily available within the company. The choice could be made to use a commercial or to use an open-source software program. Fourthly, the experience of working with a software program plays a role. Using a familiar programming environment is more favourable than a software program that is evenly suitable but less familiar to the user. Fifthly, the software program should be customizable. It should possess the ability to implement features. Sixthly, the software program should be modular. The numerical model shall be easily modified and scaled. At last, the software program should have external support. Multiple software programs have been identified that are potentially suitable to model BHS. The most important advantages and disadvantages are reported in the following subsections.

2.4.1 Matlab

Matlab is a convenient commercial software program to create numerical models. Within Matlab, it is possible to model systems that react based on events. Simulink can be used in combination with *SimEvents* and the coding environment in combination with handle classes. Moreover, Matlab is a familiar software program within Vanderlande. Furthermore, Matlab offers the ability to customize the code as the user wants (e.g. customized configuration of conveyors and carriers). Additionally, a GUI can very intuitively be developed via a Matlab plug-in. Simulink has a disadvantage over the coding environment. Within the coding environment, it is more convenient to create scalable systems. Creating layouts of any scale, order or variation can be done conveniently within this environment. This is, however, more difficult in the Simulink environment. Herein, the layout has to be predefined and, therefore, less modular.

2.4.2 Anylogic

Anylogic is a commercial software program that offers the ability to analyze manufacturing lines. This software program is simulation based and offers the possibility to model a system based on discrete events [14], but not owned by Vanderlande. Anylogic is mostly used to analyze entire high-level manufacturing lines. The low-level system design is of less importance. The software program cannot easily include reaction times or to customize conveyors. Additionally, a separate software program shall be used to create an optimization script to vary the layout and parameters. This cannot be done within the environment of Anylogic.

2.4.3 Open source software program

Alternatively, open source software programs could be used. First of all, the program CIF runs within the Eclipse environment. Herein, automata are defined that represent the plant and the controller. The system can be analyzed after combining the plant and the controller. Unfortunately, it is not possible to change the layout in a modular procedure. The layout shall be defined by hand, which complicates the synthesis of a viable and good solution design. Furthermore, Python has also a package which enables discrete event modeling. Python offers the same abilities as Matlab. However, Python is an open-source software program and has only support possibilities via the Python community. Therefore, Matlab is preferred over Python.

Table 2.1 presents a comparison of the different modeling programs. To clarify, the experience criterion reflects on the experience with Vanderlande and mine. The software programs are ranked based on six criteria. A plus sign indicates that the program is favourable, a plus/minus sign indicates that the program has some limitations and a minus sign implies that the program is unfavourable concerning a criterion. With the help of this comparison, the conclusion is drawn that the coding environment in combination with handle classes in Matlab is the most suitable modeling program for the simulation model. Matlab possesses all criteria that result in the successful development of the simulation model.

Table 2.1: Comparison of different modeling programs against the predefined criteria.

	Matlab	Anylogic	CIF	Python
Suitability	+	+/-	+	+
Availability	+	-	+	+
Experience	+	-	+/-	+/-
Customizability	+	-	+/-	+
Modularity	+	-	-	+
External support	+	+	-	+/-

Chapter 3
Tubtrax simulation framework

This chapter introduces the simulation model framework. This chapter presents how the simulation model is constructed. First of all, the simulation model's characteristics are presented. This section elaborates on the simulation's output, requirements, assumptions and variables. Thereafter, the design of the framework is presented. This section explains how the simulation is constructed. Thenceforth, a sensitivity analysis is presented that determines an adequate value for the system's sample time and the number of measurements. Hereafter, the simulation model is validated. Herein, the results of the simulation model are compared to the results of the manual-created design document and the experimental results. In the end, the first research question is answered, i.e. how can model based design improve the design process of a configuration of the baggage handling system compared to the current manual design process.

3.1 Tubtrax simulation model characteristics

This section presents the characteristics of the simulation model. Herein, the fundament of the simulation model is manifested, i.e. model's output, requirements, assumptions and variables.

3.1.1 Simulation output and requirements

As mentioned before, the capacity value is the most important KPI. The capacity is determined via the output variables of the simulation. These variables are properties of the conveyors (i.e. velocity and acceleration values) and carriers (i.e. position value). During the simulations, it is ensured that the upstream flow is configured sufficiently high, such that a carrier is waiting on an upstream conveyor at the moment that the downstream system can receive a new carrier. This assures that the capacity is determined for a system under maximum occupancy. Moreover, two modes are presented in the project that result in different capacities. The system's response is deterministic during simulations run under the design mode. Therefore, every carrier has the same cycle time (i.e. time to execute a certain series of actions) if the system is operating in a steady state. Consequently, only one cycle of carriers has to be timed. This capacity term is validated with the help of the already created manual designs by Vanderlande. This validation is performed in section 3.4.

The performance mode, however, could add more value to the simulation model as compared to the design mode. The performance mode mimics the behaviour of the actual system by using a controller with its own sample time, i.e. the plant's and the controller's sample time are uncoupled and can be different. Additionally, an offset in the starting position is applied whenever a carrier is generated. This results that the output of the simulation has in some sense stochasticity. Consequently, a carrier passes certain threshold points at a different moment in time. The simulation response is distributed and, therefore, more representative of the actual behaviour. The results obtained via this mode are validated via experiments on a setup of the actual system. This validation is performed in section 3.4. Furthermore, problem-specific requirements (i.e. design rules) are considered during the design process of a system. These rules are developed to ascertain the viability of a design. Poor quality designs are developed whenever these design rules are not followed. These rules encapsulate the required capacity that should be attained; the minimal distances that a carrier has to be physically on a conveyor before a change in the velocity is allowed; and that no collision between carriers is allowed.

3.1.2 Simulation assumptions

The simulation model possesses some assumptions. These assumptions are defined to keep the simulation model scalable and computational costly effective. The resolution of the simulation model is a result of the level of detail that is incorporated. Furthermore, the model scope only affects the size of the model. However, the level of detail affects not only the size but also the complexity. Determining the correct level of detail is of importance. Too much detail makes the simulation unnecessary time-consuming. Nevertheless, incorporating too little detail makes the model unrepresentative. Therefore, a sufficient amount of detail has to be included to meet the objectives of the study. The current design procedure and project scope are considered to define the suitable amount of detail. The current project's goal is not to mimic the system response precisely or to determine the influence of the losses produced due to friction between the carriers and the conveyors, but to study the response of a flow of carriers on an arbitrary system configuration. Therefore, the velocity profile of the conveyors is determined via uniform acceleration (with an infinite jerk value). Additionally, the motors' response that drive the conveyors is perfect (i.e. no overshoot of the velocity value). Furthermore, it is assumed that a carrier copies the velocity value of the conveyor where the highest fraction of the carrier's length is located. This interaction is assumed because slip in the motion profile is not considered. Additionally, a carrier that is entering the system is controlled by the first conveyor even whenever the carrier is not yet for 50% physically on the first conveyor. Moreover, a carrier that is leaving the system is controlled by the last conveyor even whenever the carrier is less than 50% physically on the last conveyor. Furthermore, the motion of a carrier that is sorted or merged via a 30 degrees sort or merge section is modeled as straight transport. The angle of the carrier changes whenever the carrier passes the sort or merge point.

3.1.3 Simulation variables

This subsection presents the simulation variables of the system. The Tubtrax BHS can be composed of carriers and conveyors. First of all, the parameters of the carrier are discussed. A carrier triggers changes in the system's response. Furthermore, the parameters of the different conveyor types are discussed. Additionally, the interface on a conveyor is elaborated. Lastly, the system's reaction time is discussed.

Carrier

The carrier's position is the trigger to execute certain actions on the system. The high-level goal of the Tubtrax BHS is to transport carriers through the system subjected to certain requirements and constraints. The system's controller reacts based on the carrier's position, i.e. when a carrier passes a decision or a speed change point. The carrier can have different dimensions and can be added to the system with variable lengths. Moreover, carriers can be generated with or without baggage items.

Twin belt section

The twin belt section is the most fundamental conveyor type. A twin belt section is a straight conveyor and is only allowed to transport loaded, unloaded and stacks of carriers. An example configuration with twin belt sections is presented in Figure 3.1. This figure presents the top view of two twin belt sections behind each other, each with its individual length. Additionally, both conveyors have an interface at the end of the sections. An interface is composed of a decision point and a dieback point. These coordinates can be configured as desired. Optional, a speed change point can be defined as observed on the last conveyor. This coordinate is also configurable. Moreover, each conveyor has its individual velocity and acceleration value.

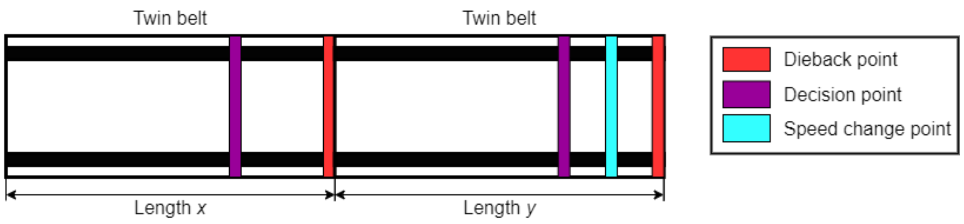


Figure 3.1: An example configuration of two twin belt sections behind each other.

Raw belt section

A raw belt section is a straight conveyor that transports baggage items. A raw belt is placed above a twin belt to load baggage items into a carrier. A raw belt section is presented in Figure 3.2. This figure presents the top view of a single raw belt with a decision and a dieback point. A raw belt has a specified length, individual velocity and acceleration value.

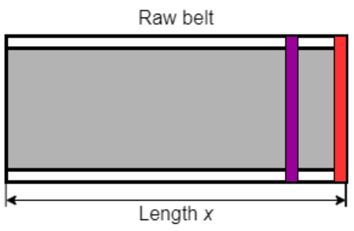


Figure 3.2: An example configuration of a single raw belt.

Merge section

The merge section is a conveyor that merges two streams of carriers into a single stream of carriers. Within Vanderlande, two types of merge sections are available, i.e. a 30 and 90 degrees merge section. The top view of both variants are presented in Figure 3.3 and Figure 3.4, respectively. A configuration that contains a merge section is always combined with twin belt sections. The streams of carriers meet at the merge point. This coordinate and the total length of the merge section are configurable.

First of all, the 30 degrees merging procedure. This procedure is conducted in a fluent motion. A carrier is not stopped (if the handover is allowed) and transported lengthwise (orientation is

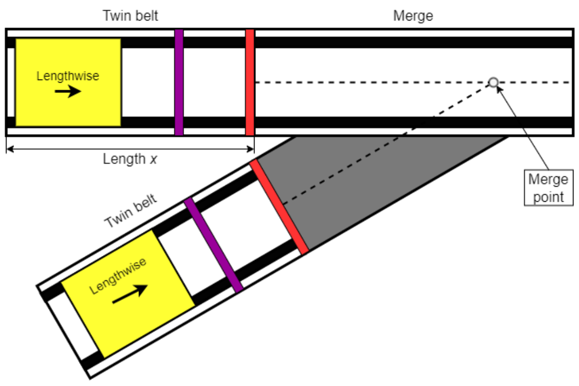


Figure 3.3: An example configuration of a 30 degrees merge section.

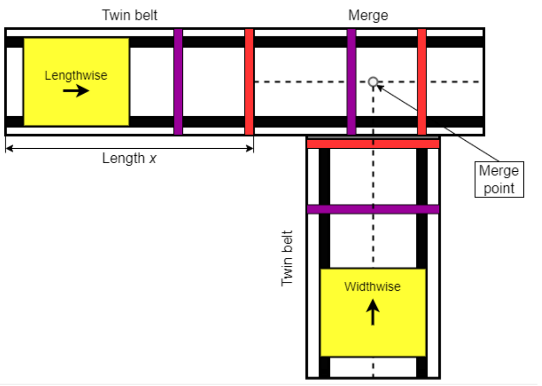


Figure 3.4: An example configuration of a 90 degrees merge section.

not changed). The angle of the carrier is changed to the angle of the main spur whenever the carrier comes from the side spur and passes the merging point. Additionally, the 30 degrees merge section has two different merging policies to merge the carriers, i.e. first in first out (FIFO) and reserved window merging policy. At the FIFO policy, the carrier that first reaches the decision point is merged first. At the reserved window policy, carriers that are merged via the side spur are assigned to a free window (depending on the occupancy of the main spur). Secondly, the 90 degrees merging procedure. This procedure is not conducted in a fluent motion. If a carrier is merged via the main spur, normal transport is applied. However, some additional steps are taken when a carrier is merged via the side spur. First of all, the transfer (which is located on the main spur) is elevated. Secondly, the carrier is transported on that transfer and goes in dieback at the end of the transfer. Thereafter, the transfer lowers and whenever the transfer is entirely lowered, the carrier is allowed to proceed. Additionally, the movement orientation of the carrier changes from widthwise to lengthwise transport. For this purpose, an additional decision and dieback point are located on the transfer. Moreover, the 90 degrees merge section has only one merging policy, i.e. the FIFO merging policy.

Sort section

The sort section is a conveyor that can sort a single stream of carriers into two streams of carriers. Again, two types of sort sections are available within Vanderlande, i.e. a 30 and 90 degrees sort section. The top view of both variants are presented in Figure 3.5 and Figure 3.6, respectively. A configuration that contains a sort section is always combined with twin belt sections. The stream of carriers is sorted at the sort point. This coordinate and the total length of the sort section are configurable.

First of all, the 30 degrees sorting procedure. This procedure is conducted in a fluent motion. A carrier is not stopped (if the handover is allowed) and transported lengthwise. The carrier's angle is changed whenever the carrier is sorted. The carrier's angle is changed to the side spur's angle whenever a carrier passes the sort point. The 30 degrees sort section does not use special sort policies. The decision if the carrier is allowed to proceed is made at the decision point, just in front of the sort point. Secondly, the 90 degrees sorting procedure. This procedure is not conducted in a fluent motion. Two different things can happen. If the carrier is not assigned to be sorted normal transport is conducted. However, when the carrier has to be sorted, the carrier is stopped on the transfer, elevated and thereafter allowed to continue via the side spur. Additionally, the movement orientation of the carrier changes from lengthwise to widthwise transport. To facilitate this, an extra dieback and decision point are added to the sort section. Additionally, no special sort policies are used. Again, the decision if the carrier is allowed to proceed is made at the decision point, just in front of the sort point.

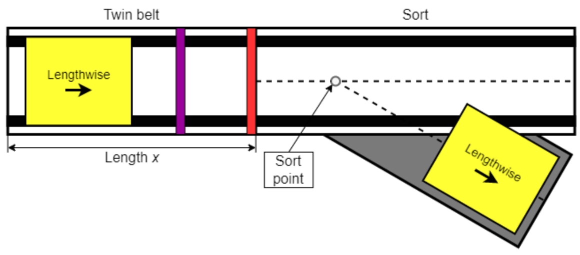


Figure 3.5: An example configuration of a 30 degrees sort section.

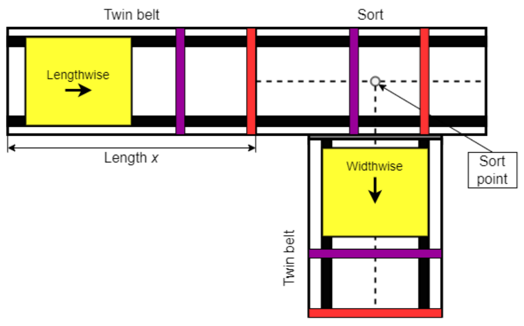


Figure 3.6: An example configuration of a 90 degrees sort section.

Conveyor's interface

A conveyor's interface is defined as a combination of a dieback point and its corresponding decision point. Additionally, if at an interface speed change is applied, the interface is extended with a speed change point. The stop in dieback action is the most fundamental action that the system can execute. The dieback point is the position on a conveyor where a carrier or baggage item stops in a controlled manner. The speed change action is the second fundamental action that the system can execute. Whenever a carrier reaches the speed change point and the carrier is allowed to proceed, the velocity of the conveyor is changed from a certain begin velocity to a certain end velocity. The conveyor's velocity is reset to its begin velocity whenever the carrier has left the conveyor for a specified fraction of its length. At each interface, the controller determines if a carrier is allowed to proceed or should go in dieback. Multiple interface flow rules can be specified to control this signal. The controller checks these flow rules whenever a carrier passes the decision point. The following basic flow rules can be defined.

1. **Downstream release gap:**
   The downstream release gap flow rule provides a positive proceed signal whenever a specified minimum distance between a waiting carrier and the downstream carrier is observed. An example of this flow rule is presented in Figure 3.7. Whenever the gap between carrier j and $j+1$ is equal to or larger than a specified value, the first conveyor receives a positive proceed signal and starts accelerating.

2. **Upstream release gap:**
   The upstream release gap flow rule provides a positive proceed signal whenever a specified maximum distance between a waiting carrier and the upstream carrier is observed. Herein, a carrier downstream on the system is waiting. Whenever an upstream carrier is close enough, the waiting carrier is allowed to proceed and the corresponding conveyor starts accelerating.

3. **Train builder release gap:**
   The train builder release gap is similar to the downstream release gap. However, this flow rule is only applied whenever the train builder functionality is used (to create trains of carriers). This flow rule indicates whenever a new carrier is allowed to enter the conveyor where the trains are created whenever a complete train is released.

4. **Timeout:**
   The timeout flow rule provides a positive proceed signal to a carrier at the interface whenever the carrier has processed forced dieback for a specified time.

It is possible to apply multiple flow rules at a single interface. Whenever multiple flow rules are applied, a carrier is only allowed to proceed whenever all specified flow rules provide a positive proceed signal. Whenever one or more flow rules yield in a negative proceed signal, the carrier remains or goes in dieback. Additionally, it is possible to define multiple interfaces on a single conveyor. A conveyor only proceeds whenever all interfaces yield in a positive proceed signal.

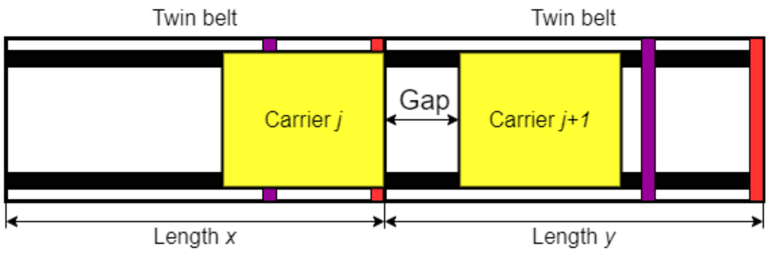


Figure 3.7: An example configuration to illustrate the downstream release gap flow rule.

System's reaction time

In practice, a controller determines the control signals to control the plant. A PLC is a Programmable Logic Controller used in many industrial applications. The communication between the controller and the plant is not instant. Therefore, reaction time is incorporated whenever the controller changes the control signals. The value of the reaction time is equal to one controller cycle (i.e. the controller's sample time) plus an additional motor activation term (i.e. to send the signals from the controller to the motor).

The reaction time influences the distance between the decision and the dieback point. By default, this distance equals the brake distance plus the reaction distance. Herein, the reaction distance equals the reaction time multiplied by the beginning velocity. Furthermore, the brake distance is calculated via the uniformly accelerated motion equation. This distance is calculated via Equation 3.1. Herein, $\delta_{\text{interface}}$ is the distance between the decision and the dieback point, v is the conveyor's nominal velocity value, $T_{\text{controller}}$ is the controller's cycle time, a is the conveyor's deceleration value and t_{brake} is the time required to go from the nominal velocity to zero velocity. Moreover, the brake distance is equal to $t_{\text{brake}} = v/a$. Keep in mind that the distance between the dieback and the decision point becomes larger whenever the controller's sample time increases.

$$\delta_{\text{interface}} = v \cdot T_{\text{controller}} + \frac{1}{2} \cdot a \cdot t_{\text{brake}}^2 = v \cdot T_{\text{controller}} + \frac{1}{2} \cdot \frac{v^2}{a} \tag{3.1}$$

3.2 Tubtrax simulation framework design

This section presents the design of the simulation framework of the Tubtrax system. Herein, the global architecture of the simulation model is introduced. Hereafter, the different automata and the fundamental state space possibilities are presented. In the end, the influence of the reaction time is discussed for the different modes, i.e. the design and performance mode.

3.2.1 Simulation layout

The global architecture of the simulation model is depicted in Figure 3.8. This flow diagram depicts how the controller and plant communicate with each other. The system is run with a sample time that equals the plant's sample time, implying that the plant is updated during each iteration. The control signal is updated depending on the controller's sample time. Whenever the control signal is not updated, the plant uses the control signal of the previous controller update that is stored in the zero-order hold structure. The design mode is created whenever

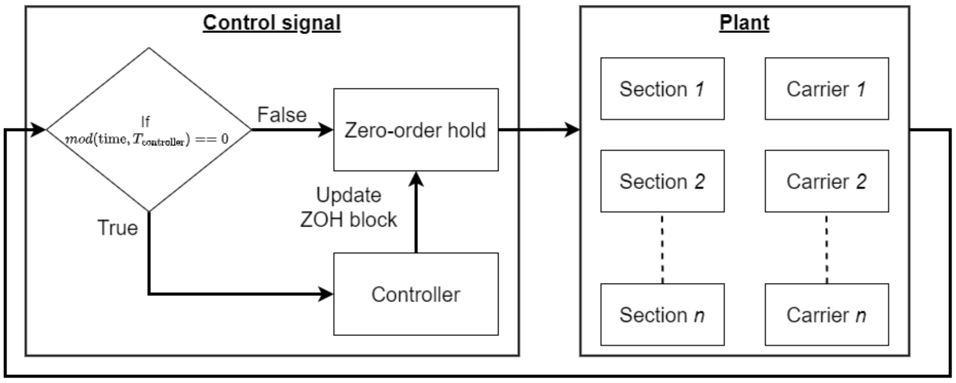


Figure 3.8: Global architecture of the simulation model.

the controller's and the plant's sample time are equalized. Then, the controller is updated every iteration and, therefore, reacts immediately. However, discretization deviations can erupt when the plant's sample time increases. Therefore, a case study is conducted to conclude what a proper plant's sample time is. The performance mode is created whenever the controller's sample time is larger than the plant's sample time. Then, the actual response of the system can be mimicked and analyzed.

3.2.2 Plant

As mentioned before, the plant is composed of conveyors and carriers. These objects are modelled as automata. An automaton depicts the state space transitions of an object and has multiple properties, i.e. events and locations. An event models actions that can take place in a system representing state changes. Furthermore, locations represent the states of the automaton. All states of an automaton are grouped in the state space. An automaton is only able to be in a certain state one at a time [16]. The state space flow diagram with all possible events of a conveyor is presented in Figure 3.9. This automaton is responsible for the acceleration and velocity profile of a conveyor. The circles depict the possible states or locations of the object. The arrows between these states illustrate the possible state transitions, i.e. events. Additionally, some states are circled twice, indicating that these states are desired states and that the system would like to end up in these locations. Finally, the arrow without a source presents the initial state. The state space of a conveyor comprises the idle, accelerating, decelerating or constant velocity state. Moreover, different state transitions are available to accelerate; decelerate; or stop accelerating or decelerating because the desired velocity value is attained. An event triggers a corresponding callback during each plant iteration. Therefore, events are present that points to their own location, enabling the possibility to update the conveyor's acceleration and velocity value during each iteration.

The automaton of a carrier object exists only out of a single location with a single event to the same location. This automaton is presented by Figure 3.10. This event triggers a callback to update the properties of the carrier, e.g. carrier's physical location, velocity value, position on the grid, etc.. Additionally, the arrival process of the carriers to the system is administered in the plant. It is possible to specify the arrival process of the carrier via multiple arrival principles. Whenever multiple principles are applied, a carrier is only generated whenever all enabled principles hold. These principles are presented here below.

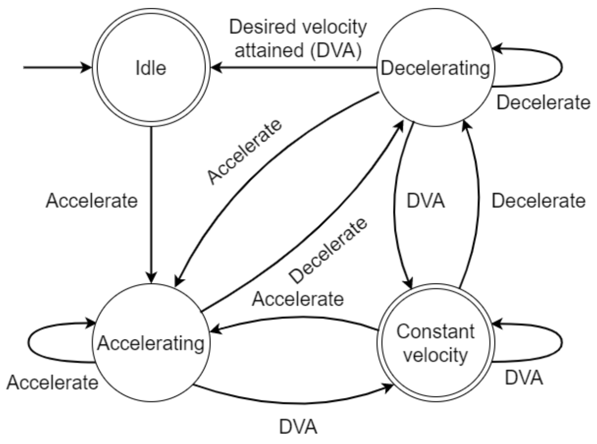


Figure 3.9: State space diagram of a conveyor.

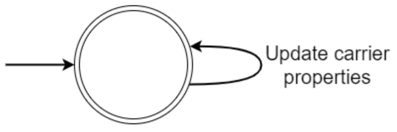


Figure 3.10: State space diagram of a carrier.

1. **Release gap:**
   A carrier is generated with a specified gap between the newly generated carrier and its downstream carrier;
2. **Head-to-head:**
   A carrier is generated with a specified head-to-head distance between the newly generated carrier and its downstream carrier;
3. **Capacity:**
   A carrier is generated with a specific capacity;
4. **Start delay**
   The carrier generation process starts only after the specified delay is processed and influences only the generation of the first carrier.

The 90 degree sort and merge sections possess an additional automaton that controls the transfer. The state space diagram of this transfer is depicted in Figure 3.11. As can be observed, this transfer has four states, i.e. lowered, lowering, elevated and elevating. A carrier is only allowed to perform certain actions whenever the transfer is in a certain state.

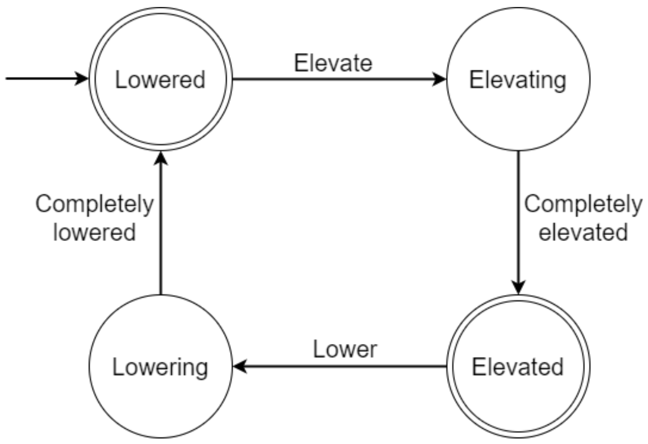


Figure 3.11: State space diagram of a sort/merge transfer.

3.2.3 Controller

The system's controller is responsible for controlling the plant using the output data of the conveyors and the carriers. The controller only controls the conveyors and the carriers respond based on the actions performed by the conveyors. The controller uses the following data of the plant to determine what the controller signal should be.

- **Carrier position jointly with the conveyor's interface:**
  The carrier's position is the initiator to perform changes in the velocity profile. Whenever a carrier passes the decision point, the controller determines if a carrier is allowed to proceed to the downstream conveyor, or should go in dieback. Additionally, whenever a speed change action is required, a carrier performs that action whenever it passes the speed change point. Again, this action changes the velocity profile.
- **Conveyor's ready-to-receive signal:**
  The conveyor's ready-to-receive signal indicates if a conveyor can accept a new carrier and is independent of the flow rules. This signal is negative whenever a conveyor is in dieback and a waiting carrier is not allowed to proceed. This signal is also negative whenever the transfer of a 90 degrees sort or merge section is occupied. The transfer should be in a certain state before a waiting carrier is allowed to be accepted.

- **Imposed flow rules:**
  Whenever a carrier passes the decision point, the controller determines if that carrier is allowed to proceed. In order to determine this, the spacing between two consecutive carriers is of importance, i.e. flow rules as discussed in the conveyor's interface paragraph.

The plant's and controller's sample times are equal whenever the design mode is used. This implies that the controller checks every iteration what the plant does and, therefore, responds immediately. Hence, the response is accurate and no error in the position occurs due to latency. However, if the sample time is chosen "large", an error in the position could occur due to discretization. A smaller sample time results in a more accurate response. Hence, a decision is made on what an appropriate value for the sample time is. Herein, the precision and computation time is the trade-off.

In the performance mode, the plant's and controller's sample times are not equal. Herein, the chance erupts that the controller observes that the decision or speed change point is passed later than actually happens. This phenomena is illustrated by Figure 3.12. This figure depicts the side view of a conveyor with a decision point, a carrier and two different sample patterns. The black lines represent the controller's sample moments and the grey lines represent the plant's sample moments. As can be observed, the plant's sample rate is higher than the controller's sample rate, indicating that the plant's properties are updated more often than the controller determines if actions have to be undertaken. The figure presents a carrier that passes a decision point. In theory, the controller should first process the reaction time and, thereafter, react. The controller's sample time plays a significant role. Sample pattern 1 depicts a situation where the controller has sampled just before the carrier has reached the decision point, implying that the controller observes that the carrier has passed the decision point at the next sample moment (the yellow line). Hereafter, the controller processes the required reaction time before reacting (green line). Sample pattern 1 is referred to as the worst-case scenario. In practice, the controller has processed two sample times. Sample pattern 2 depicts the best-case scenario. Herein, the carrier passes the decision point just before the controller samples the output signals of the plant, implying that the controller observes that the carrier has passed the decision point immediately (yellow line). Again first, the reaction time is processed before the conveyor decelerates (green line). In the current situation, the controller has processed one controller sample time as reaction time.

The two presented scenarios are boundary cases, implying that the system has a possibility to overshoot in the case of a stop in dieback action. The position error depends on the controller's sample time and the velocity of the conveyor.

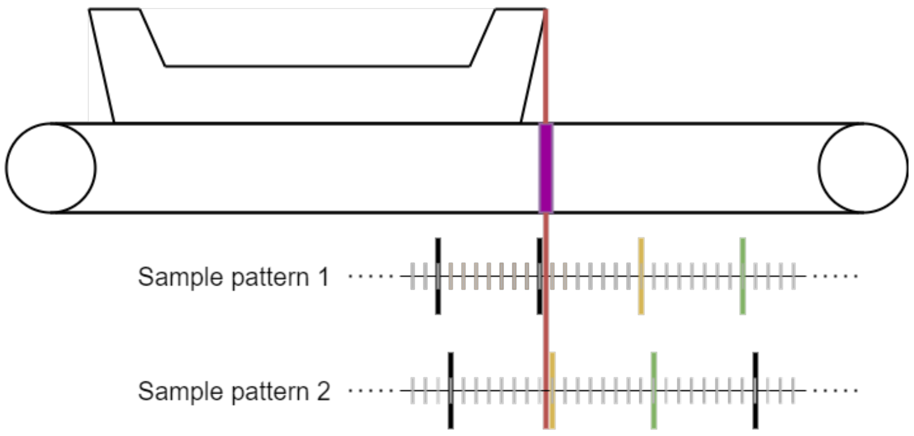


Figure 3.12: Boundary conditions concerning the system's sampling time.

3.2.4 Object oriented programming

A convenient method in Matlab to construct systems or objects that are modeled as automata is via the Stateflow toolbox. This toolbox enables the possibility to construct state transition diagrams, flow charts or state transition tables that are used to simulate a simulation model. The Stateflow toolbox uses mainly illustrative state diagrams that are constructed in the Simulink environment. This approach, however, is not convenient to be applied in the current project. One of the requirements of the simulation model is that the simulated system is able to be modular constructible. It shall be able to construct any system configuration as desired, i.e. any order of conveyors with any length and properties. Therefore, creating state diagrams in Simulink is not favourable. Object-oriented programming is applied to tackle this problem. This method is a convenient methodology to define arbitrary systems in a modular manner. The conveyors can be coupled in any order with a specified length and properties. The begin and end coordinates of each conveyor are determined programmatically and added to the conveyor class as a property of the object.

Each conveyor is modeled as a handle class. These handle classes can define automata as objects. Examples of these objects with some of its main characteristics are presented in Figure 3.13 and Figure 3.14. These objects have their individual properties, methods, events and corresponding callbacks. The properties of, for example, a conveyor class are static and dynamic data. The static data encapsulates the length of the conveyor, the coordinates of the conveyor on the grid, the coordinates of the decision point(s), the coordinates of the dieback point(s), and its log ID. Furthermore, the dynamic data structure encapsulates data at the current moment in time, like its velocity value, acceleration value, ready-to-receive state, etc.. Methods of a class define object-specific functions. A fundamental method is the constructor function. This function creates and initializes the object. Moreover, the handle classes have the ability to define class-specific events that trigger corresponding callbacks. The events of the conveyor and carrier objects correspond to the events defined in the automata as presented in Figure 3.9 and Figure 3.10, respectively. Whenever an event is triggered, the object responds to this via an event callback. As a result, this callback updates the value of a property or executes a coupled function.

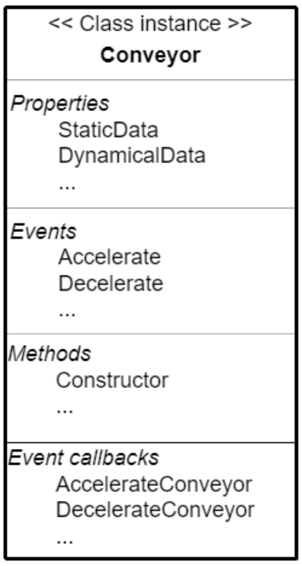


Figure 3.13: High-level structure of the conveyor class.

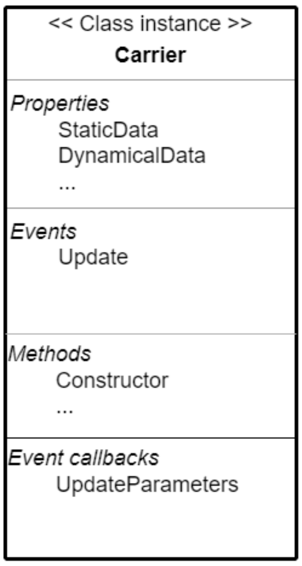


Figure 3.14: High-level structure of the carrier class.

3.3 Sensitivity analysis

A sensitivity analysis is conducted to select appropriate values for fundamental system parameters and to determine the necessary number of measurements. First of all, the system that is used to perform the sensitivity analysis is presented. Thereafter, the system parameters of the design mode are discussed. Within this mode, only the simulation's sample time is examined and a proper value is determined. Thenceforth, the performance mode's system parameters are discussed. Herein, the plant's cycle time is examined and determined. Moreover, the controller's cycle time is fixed because this value is case-dependent. At last, the number of measurements is discussed due to the output variable is distributed.

3.3.1 Sensitivity analysis system

The system used to perform the sensitivity analysis is depicted by Figure 3.15. This system is constructed of three twin belt sections. The first conveyor has an interface with a decision point and a dieback. The second conveyor has an interface with a decision, a speed change and a dieback point. The last conveyor has no interface. The goal of this system is to create a specified gap between the carriers at the last conveyor. The carrier arrival is controlled such that a carrier is waiting in dieback at the first interface whenever the second conveyor allows a carrier to enter. Therefore, the system is designed for a system that is maximally occupied.

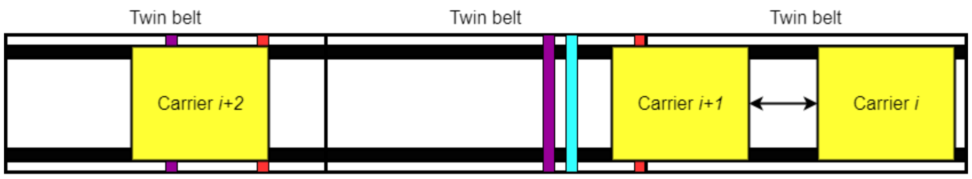


Figure 3.15: System used to perform the sensitivity analysis.

3.3.2 Design mode

A simulation run under the design mode utilizes a principle wherein the plant's and controller's sample times are equal. Hence, the simulation has a specified simulation cycle time. As a result, the controller responds immediately whenever the plant reaches a certain threshold as discussed in subsection 2.2.1. This subsection claims that the simulation cycle time shall be chosen small enough to ensure representative results. Additionally, the simulation cycle time shall not be chosen too small due to this results in a waste of computational resources. Increasing the simulation cycle time results in different output values. This deviation is the result of the sampling moment in the system. The chance exists that whenever the cycle time is too large, overshoot occurs. Consequently, the controller observes a certain threshold too late. As a result, the system responds later than designed. This effect causes overshoot and deviations in the output, i.e. discretization deviations or latency.

Multiple simulations are conducted to study the influence of different cycle times. During these experiments, the simulation end time is set to 60 seconds. Moreover, the output value and the CPU calculation time are accumulated. The gap between the carriers at the last conveyor is the output value of the simulation. The CPU time is used to provide insight into the time required to simulate 60 seconds. The results of these simulations are presented in Figure 3.16. This figure presents the gap between the carriers and the CPU time of the simulations against the simulation cycle time. Herein, x is equal to the desired gap. Conclusions concerning the simulation cycle time can be drawn with the help of this figure. Using small simulation cycle times results in the correct output values. However, for small cycle times, the calculation time grows exponentially. Using large cycle times decreases the CPU time drastically, but results in inaccurate simulation results. Multiple suitable options are identified, including 0.001, 0.002, 0.004, 0.005, 0.010 and 0.025 seconds. The latter option only results in the present case in a correct output. However, this is not the case for different cases and settings. Therefore, a cycle time of 0.010 seconds is appropriate. The output value is still as required and the CPU time is the lowest of the suitable possibilities. Moreover, the output value of the design mode is always the same whenever the system is in a steady state. Therefore, it is not necessary to investigate the number of measurements. It is required to inspect the number of measurements for the performance mode, due to these output values are distributed.

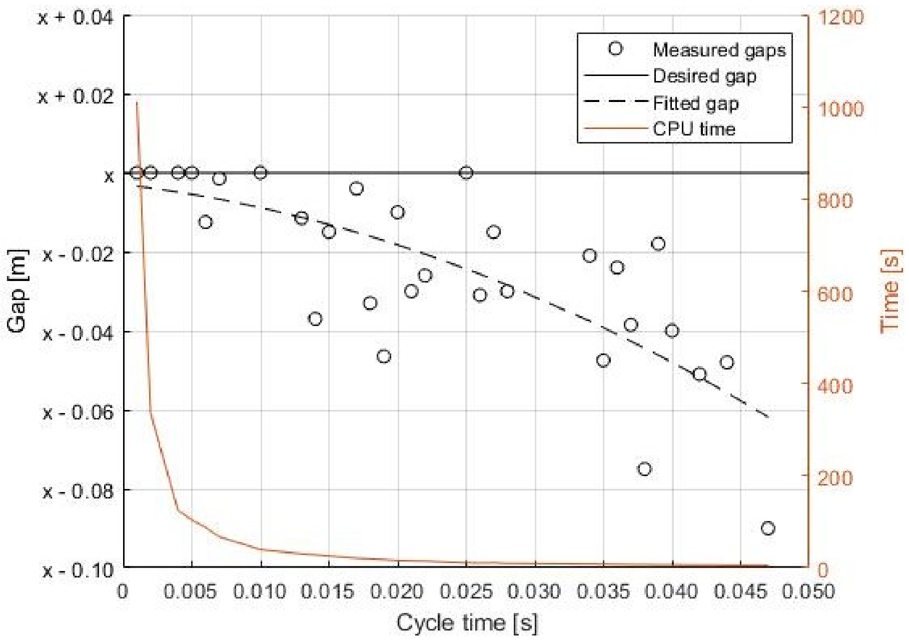


Figure 3.16: Simulation output vs. the cycle times for a system run under the design mode.

3.3.3 Performance mode

A simulation run under the performance mode utilizes a principle wherein the plant and the controller do not have the same cycle time. In theory, the plant has a cycle time that is (near) continuous. However, in practice, it is not possible to define the plant's cycle time as continuous. Therefore, again a study is conducted to choose a suitable value for the plant's cycle time. The cycle time of the plant should be chosen small enough to ensure representative results. Using a too large sample time results in discretization deviations as mentioned before. Moreover, the output values of the simulation are distributed. Therefore, the number of measurements is studied to determine an appropriate value.

Plant's cycle time

Again, multiple simulations are conducted to study the influence of different cycle times. The results of this study are presented in Table 3.1. This table presents the average gap, including the confidence intervals and the computational time for simulations run with different cycle times. Herein, x is the desired gap and the value within the brackets is the average measured gap value. The confidence interval is calculated via Equation 3.2. Moreover, the value of x and the table with the actual results can be found in Appendix A. Only potentially suitable cycle times are considered in this study. In most cases, the sample time of the controller is maximally equal to 0.020 seconds. Therefore, the plant's cycle time cannot exceed that value. Moreover, the plant's cycle time has to be dividable by the controller's sample time without rest. If this

Table 3.1: Simulation results for simulations run with different cycle times.

Cycle time [s]	**Mean gap** [m]	**CPU Time** [s]
0.001	$(x + 0.008) \pm 0.007$	928
0.005	$(x + 0.012) \pm 0.006$	89
0.010	$(x + 0.011) \pm 0.007$	32
0.020	$(x - 0.007) \pm 0.003$	15

Table 3.2: Student-t test results of simulations run with different cycle times.

Cycle time [s]	*0.001*	*0.005*	*0.010*	*0.020*
0.001	-			
0.005	0.323	-		
0.010	0.465	0.760	-	
0.020	0	0	0	-

is not the case, a controller update could be skipped. Additionally, the plant's cycle time is not allowed to be too small due to the computational cost (as observed in Figure 3.16).

The student-t test is conducted to determine if the results of the simulations are statistically different from each other. For this purpose, a null hypothesis is used. This hypothesis proposes that no statistical significance exists in a set of observations. The t-test provides the ability to determine if the difference in experimental results is due to changes in the system or due to sampling chance. This method uses the mean and the variability of the output value. The results of the student-t test are presented in Table 3.2. As can be observed, the results of the simulations run with a plant's sample time of 0.001 up and until 0.010 seconds are not significantly different. The results of the simulation run with a plant's sample time of 0.020 seconds are significantly different compared to all other results. Therefore, the conclusion is drawn that using a sample time of 0.001, 0.005 and 0.010 seconds results not in different output values. Only the calculation time is affected. The results obtained with a plant's sample time of 0.020 seconds result in an output value that has lost accuracy. Herein, the mean value is underestimated. This underestimation is the result of overshoot that happens in the system. Therefore, a sample time of 0.010 seconds is appropriate. The output value is still representative and the calculation time is the smallest of the three options.

Number of measurements

Whenever the performance mode is applied, the simulation's output value is distributed. Therefore, selecting an appropriate number of measurements is of importance. In the current project, the number of processed carriers by the system is the key indicator to determine the output value of the simulation, i.e. gap between carriers or the throughput. Processing too few carriers results in less confident results. However, processing too many carriers is a waste of computational resources. Therefore, the appropriate number of carriers that are processed by the system is chosen thoroughly. The results of the simulation are normally distributed. Therefore, the standard confidence relation is applied to provide some insights into the behaviour of the measured output variable. The simulation's output variable is given by a standard confidence interval (CI). This value is the mean of the measured data plus and minus the variation of these measurements. This value indicates that whenever a measurement is executed again, the measured data will have a value in the range of this interval within a certain level of confidence. Thus, the confidence interval provides only the range of values you can expect whenever a measurement is conducted again. The confidence interval relation is presented by Equation 3.2. Where $\bar{X}_n$ is the sample mean of measurement 1 up and until n, $t_{n-1,\alpha/2}$ is the student t-distribution for $n-1$ degrees of freedom and a significance level of $\alpha/2$, S_n is the sample standard deviation of measurement 1 up and until n and n is the number of measurements.

$$CI = \bar{X}_n \pm t_{n-1,\alpha/2}\frac{S_n}{\sqrt{n}} \tag{3.2}$$

Multiple methods are proposed to determine an appropriate number of measurements. First of all, a graphical method is proposed. This method plots the cumulative mean value of the output variable against the number of measurements. Hereafter, the point where the cumulative mean is converged to a constant value can be identified. Secondly, the confidence interval method is proposed. This method is based on the first method and plots the cumulative mean and the confidence interval against the number of measurements. This method utilizes a precision term. The precision is defined as the ratio between the variation and the cumulative mean of the measurements, i.e. the half-width of the confidence interval and the cumulative mean [17]. The advantage of this metric is that a metric is based on simulation output values and provides a statistical measure of precision that guides the selection of the number of measurements. Figure 3.17 presents a plot that depicts the cumulative output value of the simulation. Moreover, the lower and upper confidence intervals and the precision value are plotted. Furthermore, the y-axis presents the measured gap where x equals the desired gap. The figure shows that the results converge as the number of measurements increases. Based on the precision value and the precision threshold (typically 5%), the conclusion is drawn that at least 10 gaps have to be measured. Measuring more gaps results in a waste of computational effort.

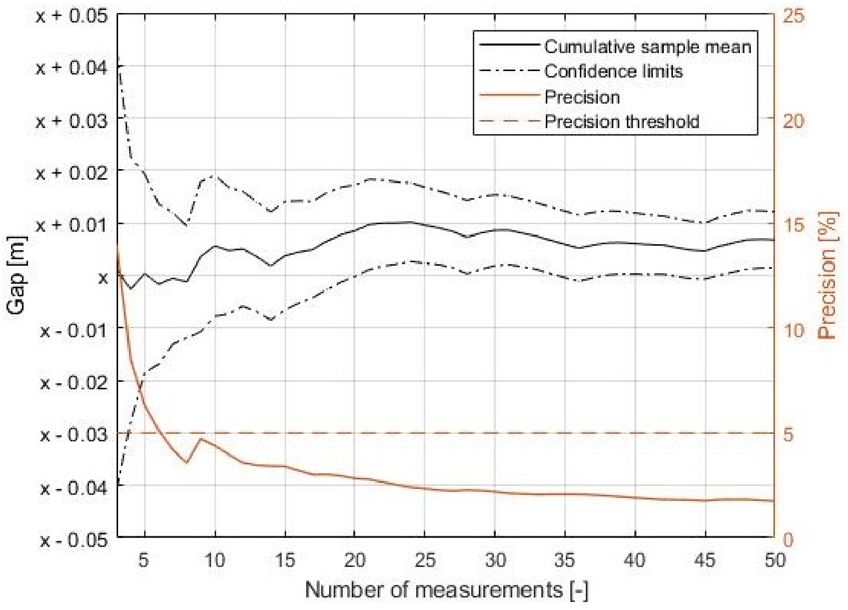


Figure 3.17: Confidence intervals of the measured gaps between the carriers at the last conveyor.

3.4 Simulation model validation

Validation is performed to determine if the results of the model are representative to the actual system. Therefore, the results of the simulation model are compared to experimental results. To this end, the straight transport and the sorting and merging functionality are validated.

3.4.1 Straight transport

Straight transport is defined as all transport that is conducted in the same orientation and where a carrier is not sorted or merged. To validate this, the same case is considered as during the sensitivity analysis. This case includes the fundamental actions that can be undertaken during this type of transport, i.e. dieback and speed change action. Two validation experiments are conducted. The first type of validation is performed via the design mode of the model and the manually created design document by Vanderlande. The second type of validation is performed via the performance mode of the model and actual measurements on the system.

Virtual validation

During the virtual validation, the manually created design document is compared to the results of the simulation model run for the design mode. The design mode is suitable to be compared to this document due to the purpose of this design document relies on the same principle, i.e. the system is simulated with a single simulation cycle time and a fixed reaction time is applied before every state change. There are three phases in this process that governs the response of the system. First of all, the carrier starts up from dieback at the first conveyor and reaches the speed change point at the second conveyor. Secondly, the carrier executes the speed change. Lastly, the gap between the current carrier and the carrier waiting in dieback on the first conveyor is attained. These phases are defined as one cycle. A visual representation of these phases are depicted in Figure 3.18.

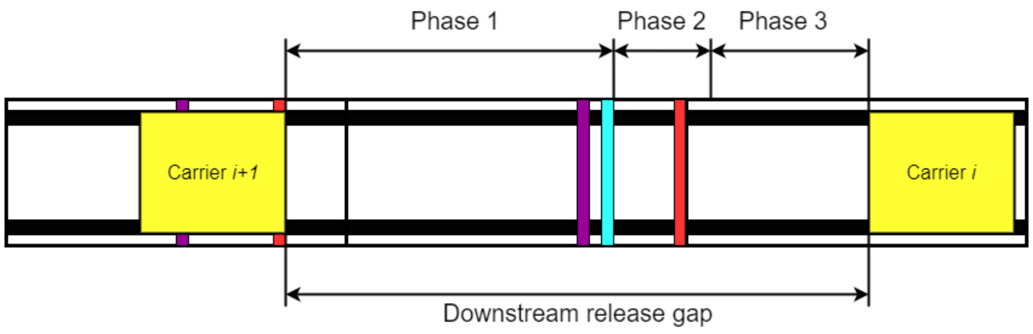


Figure 3.18: All phases of one cycle of a straight transport system.

The results of the straight transport case of the design document and the simulation are presented in Table 3.3. This table presents the time required for every phase and is observed that the results between the manual design document and the simulation are the same for every phase. Therefore, it is concluded that the straight transport functionality is modelled correctly.

Table 3.3: Straight transport virtual validation results.

Phase		**Manual design document** Time [s]	**Simulation** Time [s]
1		1.05	1.05
2		0.48	0.48
3		1.47	1.47
Total		3.00	3.00

Experimental validation

In reality, the system is not deterministic. Therefore, simulations with the performance mode are executed to enable the possibility to compare the results of the experiments with the results of the simulation. The virtual validation confirmed that straight transport is modelled correctly. Therefore, the goal of the experimental validation is to confirm that the simulation model can produce results that mimic the behaviour of the actual system. During the experiment, the gap between the carriers at the last conveyor is measured. The number of measurements is higher as established during the sensitivity analysis to obtain more confident results. The results are normally distributed which enables the ability to create the confidence intervals as suggested by Equation 3.2. The results of the experiment and simulation are presented by Figure 3.19 and Figure 3.20, respectively. The figures present bar charts of the measured gaps, the average gap value, the three sigma values and a fitted normal distribution. Moreover, the results are summarized in Table 3.4. The table presents the average gap value including, the confidence

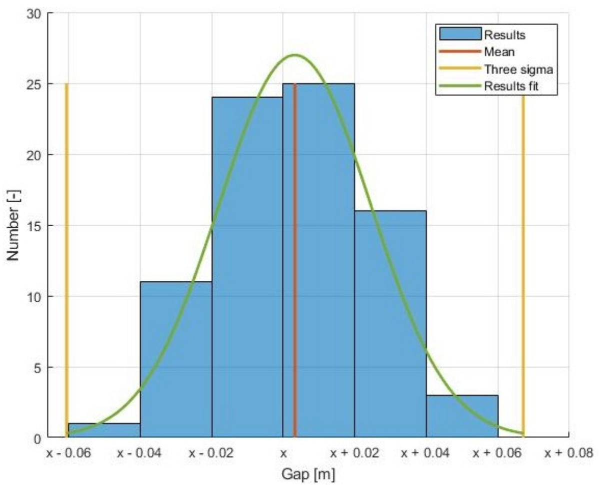


Figure 3.19: Experiment results straight transport validation.

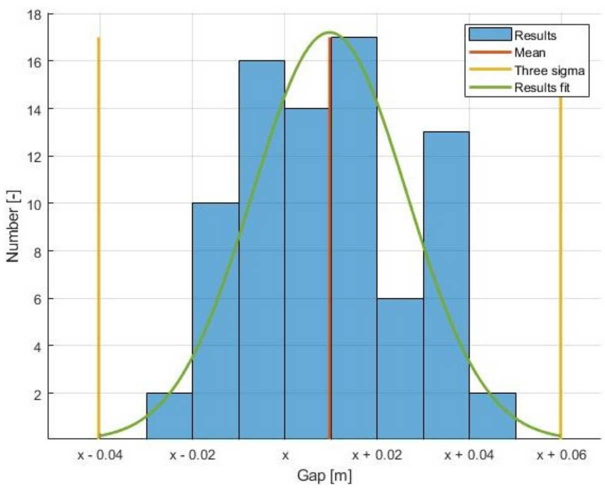


Figure 3.20: Simulation results straight transport validation.

intervals of the results. Moreover, x represents the desired gap and the value between the brackets represents the measured gap. As can be observed, the results obtained via the simulation are overestimated by 6 millimeters compared to the experimental result. In the model, no losses are incorporated and the velocity profile is simplified. Therefore, this difference is acceptable and the simulation mimics the actual system's behaviour correctly for the purpose of the project. Bar charts of the actual results of the measurements and the summarized table with the actual results are presented in Appendix A.

Table 3.4: Summary of the normal transport validation (performance mode).

	Experiment	Simulation
Average gap [m]	$(x + 0.003) \pm 0.007$	$(x + 0.009) \pm 0.005$

3.4.2 30 degrees sort & merge section

Sorting or merging streams of carriers can be executed via two distinct out- and infeed angles, i.e. 30 and 90 degrees. Within the simulation model, the 30 degrees sort and merge action is modelled as straight transport where the angle of the carrier is equal to the angle of the conveyor where the carrier is physically present. The angle of the carrier is changed whenever the carrier passes the sort or merge point. This principle was illustrated in Figure 3.3 and Figure 3.5. In reality, the change of the carrier's angle is governed by a fluent motion (the angle changes with a small delta). Therefore, an experiment is executed to motivate that the abrupt change of angle assumption is representative. For this purpose, two sensor trigger moments are measured, i.e. one sensor in front of the 30 degrees sort section and one sensor after the 30 degrees sort section. This experiment is executed on the actual system and via the simulation model. The results of the experiment are normally distributed and presented by Figure 3.21. Furthermore, the results of the simulation are always identical. There are no actions (dieback or speed change) applied between the measured sensors. Hence, no deviation is created in the simulation and the time between the sensors is always the same. The results are summarized in Table 3.5. The difference in mean results equals 5 milliseconds, resulting in a mean deviation of 1 centimeter. During experimentation is observed that the sensor's position in the experimental setup may have some uncertainty in the position. Additionally, the reflector of the sensor is approximately 2 centimeters wide. It is unknown when the sensor is triggered precisely. Due to these considerations, the difference is acceptable and the assumption concerning the 30 degrees sort and merge section is motivated.

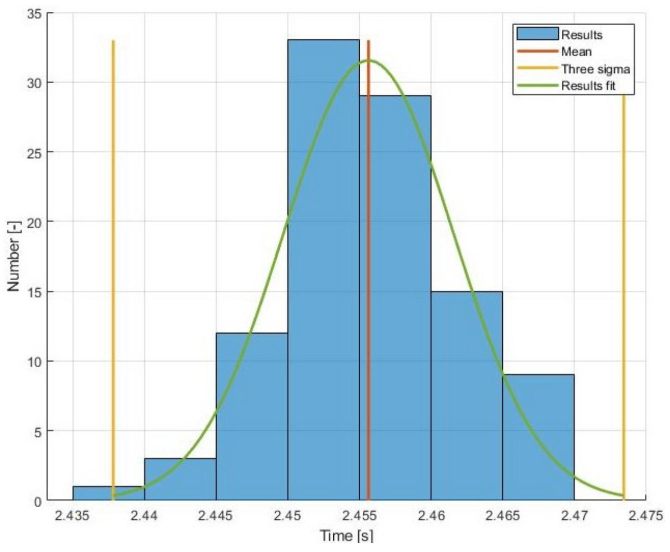


Figure 3.21: Experiment results 30 degrees sort functionality.

Table 3.5: Summary of the 30 degrees sort section validation.

	Experiment	Simulation
Time delta [s]	2.455 ± 0.002	2.460 ± 0.000
Difference [%]	100.00	100.20

3.4.3 90 degrees sort section

A 90 degrees sorting section can sort a carrier arriving in the lengthwise and leaving in the widthwise orientation via the side spur. The orientation is changed via a transfer, as mentioned before. The same setup is created as is available in the test center of Vanderlande to validate this category of transport. The general layout of this system is presented by Figure 3.22. This setup contains a 90 degrees sort and merge section combined and follows the route indicated by the orange arrow. The first type of validation is performed via the design mode and the manually created design document by Vanderlande. The second type of validation is performed via the performance mode and actual measurements on the system.

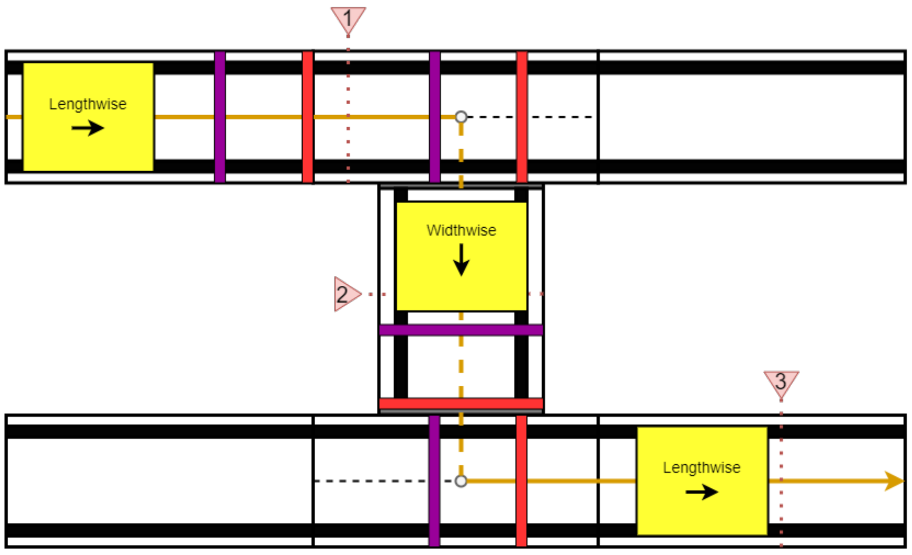


Figure 3.22: General layout used to validate the 90 degrees sort and merge functionality.

Virtual validation

As during the virtual validation of the straight transport, the manual design document is compared to the results of the simulation model run for the design mode. Within the design document, the assumption is made that the transfer movement time equals a prespecified value. As a result, this value is used in the simulation to be able to properly compare the results. Three phases govern the response of the 90 degrees sort functionality. First of all, a carrier is allowed to enter the sort section and goes in dieback on the transfer. Secondly, the system observes that the carrier is in dieback on the transfer, reacts and moves the transfer to the elevated position. Lastly, the carrier leaves the transfer. The last phase ends whenever a new carrier is allowed to enter the sort section. All phases combined equals a single cycle. The results of the virtual validation are summarized in Table 3.6. The table presents the time every phase requires. The differences between the results are minimal (at most 0.01 seconds per phase). The observed differences are the result of the discretization of the simulation model. Therefore, the conclusion is drawn that the 90 degrees sort motion is modelled correctly.

Table 3.6: 90 degrees sort virtual validation results.

Phase		**Manual design document**	**Simulation**
		Time [s]	Time [s]
1		1.65	1.66
2		0.33	0.33
3		0.86	0.86
Total		2.84	2.85

Experimental validation

In reality, the system is not deterministic. Therefore, simulations with the performance mode are executed to enable comparison between the results of the experiments and the simulation. The virtual validation confirmed that 90 degrees sorting functionality is modeled correctly. Therefore, the goal of the experimental validation is to confirm that the simulation produces results that mimic the behaviour of the actual system. Therefore, different sensor signals are captured to determine the time required to perform certain actions. First of all, signals that indicate the state of the transfer are captured. With these signals, it is possible to determine the time required to elevate and lower the transfer. Secondly, the time required to execute the sorting process is mapped by capturing the sensor signal at the end and begin of the sort section. The sensor at the begin and end of the sort section is depicted in Figure 3.22 by sensor numbers 1 and 2, respectively. In total, three sensor deltas are determined. Sensor delta 1 is the time between triggering sensor 1 and the moment that the transfer starts elevating. Sensor delta 2 is the time between triggering sensor 1 and the first moment that the transfer is entirely elevated. Sensor delta 3 is the time between triggering sensor 1 and sensor 2.

Bar charts with the results of the experiments and simulations of each sensor delta are presented in section B.1. Moreover, the results are summarized in Table 3.7. The table presents the mean delta times, including the confidence intervals. With the obtained results, it is possible to determine the average time the transfer requires to change its state. It is assumed that both transformations require the same amount of time, i.e. elevated to lowered and vice versa. The average transfer movement time is equal to subtracting sensor delta 1 from sensor delta 2. This time is pure movement. Herein, no reaction time is incorporated. Furthermore, this value is used in the simulation to enable accurate comparison. Moreover, the sensor delta times of the simulation and the experiment are compared to each other. The mean difference in results between the experiment and the simulation of sensor delta 1, 2 and 3 equals 0.007, 0.010

and 0.011 seconds, respectively. The position error will be less than 2 centimeters due to the varying velocities. Again, the position of the sensor in the experimental setup may have some uncertainty. Additionally, the reflector of the sensor is approximately 2 centimeters wide. It is unknown when the sensor is triggered precisely. Due to these considerations, the differences are acceptable and is concluded that the performance mode of the simulation model mimics the behaviour of the actual system correctly for the purpose of the project.

Table 3.7: 90 degrees sort experiment vs. simulation results.

Sensor delta		**Experiment** Time [s]	**Simulation** Time [s]	**Difference** [s]
1		1.312 ± 0.003	1.305 ± 0.005	-0.007
2		1.415 ± 0.003	1.405 ± 0.005	-0.010
3		2.344 ± 0.006	2.355 ± 0.005	$+0.011$

3.4.4 90 degrees merge section

A 90 degrees merging section can merge a carrier arriving in the widthwise and leaving in the lengthwise orientation via the main spur. The orientation is changed via a transfer, as mentioned before. To validate this category of transport, the same setup is used as during the 90 degrees sort validation and is presented by Figure 3.22. The first type of validation is performed via the design mode and the manually created design document by Vanderlande. The second type of validation is performed via the performance mode and actual measurements on the system.

Virtual validation

As during the virtual validation of the normal transport and sort section, the manual design document is compared to the results of the simulation model ran under the design mode. Within the design document, the assumption is made that the transfer movement time equals a prespecified value. As a result, this value is used in the simulation to properly compare the results. Three phases govern the response of the 90 degrees merge functionality. The first phase starts whenever a carrier is waiting in front of the merge section and is allowed to enter the merge section. The first phase ends whenever the carrier is in dieback at the transfer. The second phase regards the transfer movement process. The last phase starts whenever the carrier starts accelerating to leave the merge section in the lengthwise direction. The last phase ends whenever the carrier has left the transfer and a new carrier is allowed to enter the merge section. The above-described phases present a single cycle. The results of the virtual validation are summarized in Table 3.8. The table presents the required time for every phase. There are no differences between the results of the manual design document and the results of the simulation. Therefore, the conclusion can be drawn that the 90 degrees merge motion is modelled correctly.

Table 3.8: 90 degrees merge virtual validation.

Transition		**Manual design document** Time [s]	**Simulation** Time [s]
1		1.82	1.82
2		0.30	0.30
3		0.85	0.85
Total		2.97	2.97

Experimental validation

The virtual validation has confirmed that the 90 degrees merging functionality is modeled correctly. Therefore, the goal of the experimental validation is to confirm that the simulation can produce results that mimic the behaviour of the actual system. Again, different sensor signals are captured to determine the time required to perform certain actions. First of all, sensor signals that indicate the state of the transfer are captured. Secondly, the time required to execute the merging process is mapped by capturing the sensor signals at the begin and the end of the merge section. These sensors are depicted in Figure 3.22 by sensor numbers 2 and 3, respectively. Again, three sensor deltas are captured. Sensor delta 1 is the time between triggering sensor 2 and the moment that the transfer reached the elevated state. Sensor delta 2 is the time between triggering sensor 2 and the moment that the transfer reached the lowered state. Sensor delta 3 is the time between triggering sensor 2 and sensor 3.

Bar charts with the results of the experiments and the simulations are presented in section B.2. Moreover, the results are summarized in Table 3.9. With these results, it is possible to determine the time required for the transfer to change its state from lowered to elevated and vice versa. The average transfer movement time is equal to subtracting sensor delta 2 from sensor delta 1. This resulted in a time that is different to the transfer time used in the sorting functionality. Furthermore, this value is used in the simulation to enable accurate comparison. Moreover, the sensor delta times of the simulation and the experiment are compared to each other. The difference in results of sensor delta 1, 2 and 3 equals 0.006, 0.005 and 0.007 seconds, respectively. The position error is less than 2 centimeters due to the varying velocities. Again, the sensor's position in the experimental setup has some uncertainty. Additionally, the reflector of the sensor is approximately 2 centimeters wide. It is unknown when the sensor is triggered precisely. Due to these considerations, the differences are acceptable and is concluded that the performance mode of the simulation model mimics the merge behaviour of the actual system correctly for the purpose of the project.

Table 3.9: 90 degrees merge experiment vs. simulation results.

Sensor delta	**Experiment** Time [s]	**Simulation** Time [s]	**Difference** [s]
1	0.290 ± 0.002	0.296 ± 0.006	0.006
2	0.391 ± 0.002	0.396 ± 0.006	0.005
3	3.383 ± 0.005	3.390 ± 0.000	0.007

3.5 Improved design procedure

This section elaborates on the functional abilities of the simulation model. Moreover, the post-process features of the simulation model are presented. In the end, it is motivated why the developed Tubtrax simulation framework improves the design process of the Tubtrax BHS.

3.5.1 System features

One of the bottlenecks concerning the current design procedure is that defining configurations takes a lot of time. Everything is done by hand. Therefore, the developed Tubtrax simulation framework provides the ability to define arbitrary configurations via a quick and easy procedure. Moreover, it is possible to specify certain conveyor settings that apply to all conveyors to define arbitrary configurations. The following aspects can be specified to achieve this.

- **Number of used conveyors:**
  The number of used conveyors has to be specified, implying that a system can be defined as small or as large as possible.

- **Conveyor lengths:**
  Each conveyor has its length value and has to be specified, offering the possibility to create different configurations with just some minor changes.

- **Types of conveyors:**
  Each added conveyor to the system can have any functionality, e.g. twin belt section, sort section, merge section, unload section. This enables the possibility to couple multiple functionalities with each other. Moreover, it becomes possible to check the feasibility of a coupled system in an early design stage. Coupling multiple building blocks is nearly impossible by means of the current design procedure, as the system becomes too large to analyze by hand. Moreover, the Tubtrax simulation framework provides the ability to extend the library of conveyor types and functionalities, due to its modular art.

- **Specify the number and position of interfaces:**
  It is possible to specify the number and position of an interface on each conveyor. As mentioned before, an interface consists of a dieback point, a decision point and optionally a speed change point.

- **Velocity and acceleration settings:**
  The dynamical properties of a conveyor have to be configured individually. The model provides the ability to specify an acceleration and deceleration value. Additionally, the used velocity value is specified beforehand. Whenever the speed change functionality is used, it is required to specify the nominal (start) velocity and the velocity after the speed change action.

- **Quick stop and quick speed change:**
  As an addition and to illustrate that the simulation model is modular and thus can be extended easily, the quick stop and quick speed change functionalities are added. This functionality circumvents the use of a centralized controller. The conveyor goes in dieback or performs speed change whenever a carrier triggers a sensor. This functionality reduces the position error and attains a higher accuracy, due to these actions are not influenced by the cycle time of the controller.

The general conveyor settings apply to all conveyors in the system. The straight transport section has no additional settings that can be changed. However, the sort and merge sections have some additional settings that should be considered whenever a system is configured. These additional settings are presented here below.

- **Sort and merge angle:**
  The simulation framework provides the ability to specify the sort and merge angle, which can have a value of 30 or 90 degrees.

- **Sort and merge point:**
  The position where the perpendicular bisector of the main and side spur of the sort and merge section intersects has to be configured. This position specifies the location of the side spur.

- **Transfer settings:**
  The transfer settings specify the transfer movement time; when the transfer is allowed to change its state to its initial state; and when a new carrier is allowed to enter the conveyor.

Moreover, the load and unload sections have some additional conveyor settings. The load functionality uses a sensor in the system that synchronizes the baggage item with its coupled carrier. Additionally, the unload functionality utilizes a tilt mechanism that unloads a baggage item from its coupled carrier. This mechanism possesses the following properties: the mechanism movement time and when a new carrier is allowed to enter the section. The unloader functionality is also a convenient example of how the system can easily be extended. Different conveyor types can be defined and coupled to other conveyors in the system, implying that the simulation framework always can be extended whenever new conveyor types or functionalities are developed. Moreover, arbitrary systems can be quickly created with the help of the simulation model. Thereafter, the defined configuration is simulated and evaluated. Whenever the requirements are not yet satisfied, applicable changes can be made and the system is re-evaluated. This process is conducted up and until the results are satisfactory. This method enables the possibility to quickly study and update system configurations. To make the simulation model more accessible, a graphical user interface (GUI) is developed. Defining new system configurations can be done without any programming or script knowledge. A user only has to specify conveyor and system settings to create a configuration. A detailed description of the GUI is presented in Appendix A.

3.5.2 Post-process abilities

Another bottleneck concerning the current design procedure is that the system should be evaluated by hand, implying that all calculations are done by hand (concerning the carrier's position on the configuration). Therefore, the Tubtrax simulation framework provides the ability to automatically evaluate the dynamic response of the conveyors, carriers and baggage items. Moreover, the framework provides the ability to post-process the results of a simulated system, which is required to analyze the results and optionally update the system configurations and settings. Therefore, the following post-process features are developed.

- **Capacity:**
  The simulation model determines the capacity values at each conveyor in the system.
- **Determine the gaps between carriers or baggage items:**
  This feature determines the gap value between an object and its upstream object.
- **Snapshot:**
  This features creates a snapshot of the system at any moment in time and an anonymized example in Figure 3.23. Herein, all the system's properties are presented, i.e. the system, velocity values, gaps and capacity of the system.

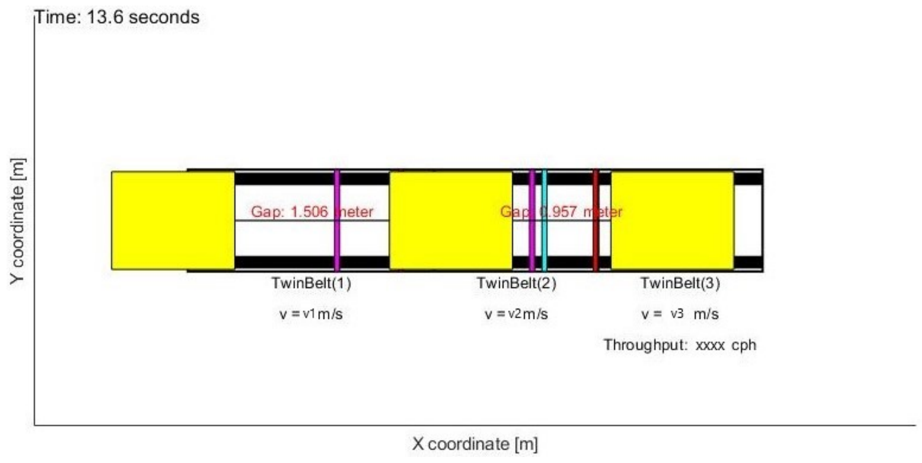


Figure 3.23: A snapshot of a simulated system.

- **Dynamic representation:**
  The simulation framework provides the ability to create a dynamic representation of the simulated system. Moreover, the dynamic representation can be saved. The representation presents the conveyors; the conveyors' velocity values; the carriers' or baggage items' position values; the gaps between the carriers or baggage items; the capacity value at the last conveyor of the system; and the state of any 90 degrees sort or merge transfer.

- **Plots of the output variables:**
  The simulation framework provides the ability to create different plots of output variables. An acceleration and velocity plot versus the time can be created for every conveyor. Additionally, a velocity and position plot versus the time can be created for every carrier and baggage item.

- **Carrier velocity plot:**
  The carrier velocity plot feature creates a plot that displays the velocity profile of a carrier against the length of each conveyor in the system. An example of this plot is depicted by Figure 3.24. Herein, the system's layout with its properties is presented. Moreover, the blue line presents the velocity profile of a carrier mapped to the X coordinate of the system.

- **Slip indicator:**
  The slip indicator analyzes potential slip in the system. This slip term is identified as the event when the carrier's contact surface experiences different velocities due to different conveyors, happening whenever the carrier's front is in contact with conveyor *i+1* and the carrier's rear is in contact with conveyor *i*. Uncertainty in the carrier's position can arise due to this effect. Therefore, mapping these events is of great value. Keep in mind, that the carriers' contact surface is not equal to the carrier's total length. The carrier's shape is responsible for this phenomenon.

- **Design rules validation:**
  This dialog presents if the defined design rules and requirements are met.

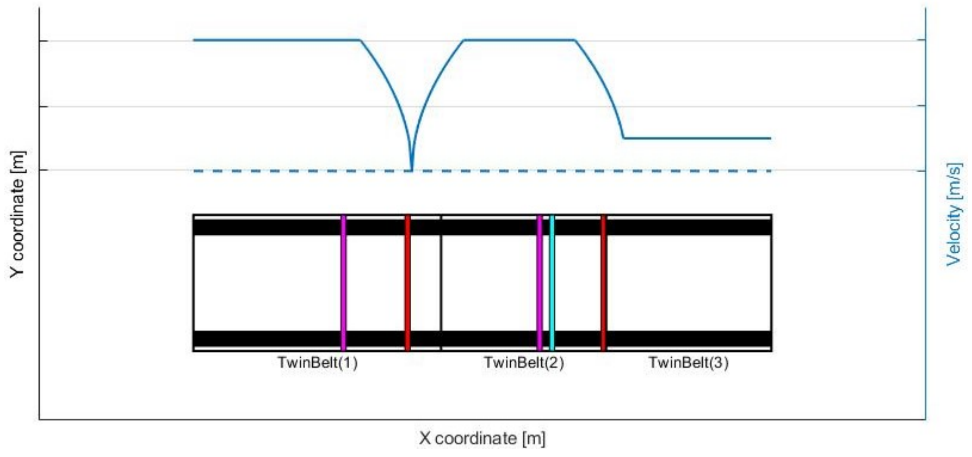


Figure 3.24: A carrier velocity plot of a simulated system.

3.5.3 Current vs. MBD design procedure

The current design procedure for the Tubtrax baggage handling system within Vanderlande is designing configurations by hand. A system is defined by hand and all the calculations are conducted by hand. As concluded in the problem statement, this approach is quite cumbersome and time-consuming, implying that creating or updating a system requires a lot of resources. Therefore, a simulation framework is designed to ease the job of the system designer. This framework provides the possibility to quickly define configurations and set all system parameters. Furthermore, the simulation framework provides the ability to simulate the created system, implying that all calculations are no longer required to be conducted by hand. This saves time and prevents that human calculation errors are made. Moreover, the simulation framework provides the ability to analyze the results via different post-process features. These features help to make decisions on how the system shall be updated or if the system satisfies the requirements. Additionally, a GUI is created to accelerate the definition process of a system configuration even more. The GUI provides the ability to create system configurations with limited programming skills or script knowledge.

The current design procedure offers only the ability to evaluate deterministic systems. Herein, a fixed reaction time is utilized whereas in reality this value is distributed. Therefore, the influence of the controller's cycle time plays a significant role in the system's behaviour. With the current design process, it is only possible to study the effect of the controller's cycle time via experiments on the actual system, implying that studying the system's behaviour in an early stage of the design phase is missing. With the help of the created simulation framework, it is possible to analyze this phenomenon. The framework provides the possibility to run a simulation as a deterministic system (to evaluate a theoretical model) and to run a simulation to mimic the behaviour of the actual system (to evaluate the actual system performance). Moreover, it is possible to study the system response in more detail with the results of the simulation. The simulation framework provides the possibility to, for example, study the effect of potential slip. A convenient feature of using a model is that certain effects (like the slip indicator) can be designed once and reused thereafter, implying that the results are automatically generated after every simulation. Additionally, the framework provides the ability to create systems as big as the designer wishes. The model is scalable, implying that multiple functionalities can be coupled with each other. This would be a cumbersome task with the current design procedure.

Lastly, the definition of the simulation framework provides the possibility to define an optimization framework. An optimization framework can be applied to find automatically feasible solutions that are restricted by constraints and should meet imposed requirements. Hence, the simulation framework is used in the remainder of the report to obtain the results used in the optimization framework, implying that the optimization framework is designed via simulation based optimization principles.

Chapter 4
Optimization techniques and algorithms

Simulations are fundamentally used to evaluate the system's performance and to aid in the design process. The simulation's results are an estimation of the results of the actual system. Additionally, the traditional design process is fundamentally a manual iterative process. Currently, researchers are trying to combine simulation and optimization procedures to provide an entire solution. Missing is often a proper technique that can utilize the information of the simulation to make a decision. Therefore, simulation based optimization (SBO) provides a structured approach for designing and configuring a physical system whenever the analytical expression is missing [18]. According to Carson and Maria [19], simulation optimization is the procedure of finding the best input variables values from among all possibilities without explicitly evaluating each possibility. The objective of simulation optimization is to minimize the resources spent while maximizing the information obtained in a simulation experiment. This chapter provides a brief overview of optimization techniques and algorithms suitable for simulation based optimization problems. Exhaustive literature review and more detailed books with respect to simulation based optimization are available in [20] [21] [8]. This chapter introduces the optimization purpose and techniques. Finally, this chapter presents gradient-free optimization algorithms that are suitable for the current simulation based optimization problem.

4.1 General optimization problem

Given that a system can be designed and controlled to achieve some desirable performance, the question may arise of how this system should be designed. This requires the development of additional analytical or experimental techniques for efficiently determining satisfactory system configuration. To find such a system configuration, a performance measure should be proposed. This performance measure is often referred to as a performance metric, fitness function or cost function (chapter 1 of [12]). Consider that during an optimization study not always a strictly optimal solution have to be found. Often, near-optimal solutions or satisfactory solutions have to be found. Therefore, a solution is good enough whenever the requirements are met, the unrelaxable constraints are not violated and the relaxable constraints are violated minimally.

An optimization problem in its most general form is presented by Equation 4.1. The goal of this problem is to find the best solution among all feasible solutions. Specifically, the goal is to minimize (or maximize) the fitness function ($f(\vec{x})$ in the current example). Furthermore, $\vec{x}$ are the input parameters or decision variables, $g_i(\vec{x})$ is inequality constraint i, $h_j(\vec{x})$ is equality constraint i, m are the number of inequality constraints and l are the number equality constraints. The fitness function is a function of these input parameters. Varying these parameters result in different output values. Additionally, the fitness function is subjected to certain constraints (e.g. system-specific constraints). Furthermore, optimization problems can be divided into continuous and discrete optimization problems. Continuous optimization problems have continuous (in)equality constraints and variables, whereas all variables and constraints in a discrete optimization problem are restricted to discrete values.

$$\begin{aligned} \underset{\vec{x}}{minimize} \quad & f(\vec{x}) \\ subjected\ to \quad & g_i(\vec{x}) \leq 0,\ i = 1, ..., m \\ & h_j(\vec{x}) = 0,\ j = 1, ..., l \end{aligned} \tag{4.1}$$

4.1.1 Optimization objective

As mentioned before, the fitness function is a measure concerning the system's performance and is classed depending on the system characteristics, decision variables and constraints. A fitness function can be linear or non-linear; convex or concave; differential or non-differential; or continuous or discrete. During the current simulation based optimization problem, the system response cannot be determined algebraically due to the system exhibiting non-linear behaviour; having non-intuitive influences between variables; having time and causal dependencies; and possessing a large number of parameters. Therefore, the system response is estimated via simulation, implying that nothing can be concluded concerning linearity, differentially and convexity. A model of the actual system is created to estimate the system response. Thereafter, this model is transformed into an executable simulation model. Simulation based optimization could find a satisfactory model configuration by minimizing (or maximizing) a function of output variables determined via a simulated model. Thereafter, an optimization framework can be established that evaluates the fitness function and updates the values of the decision variables based on the results. This optimization framework shall be developed to solve the optimization problem. The simulator (that evaluates the defined system) and the optimization method (that consists of one or more optimization algorithms) should be coupled with each other to define such a framework. Therefore, a suitable modeling and simulation method; a suitable optimization method; a proper optimization objective; and sufficient computation power are required (chapter 1 of [8]).

4.1.2 Optimization constraints

The optimization problem within the simulation based optimization domain possesses constraints. These constraints are classed to be considered and evaluated by the optimization framework. These constraints are classed via the so called *QRAK* taxonomy. The letters representing the composition of the taxonomy are corresponding to properties belonging to each constraint. The first letter indicates if the constraint is Quantifiable or Non-quantifiable. The second letter specifies if the constraint is Relaxable or Unrelaxable. The third letter pinpoints if the constraint is A priori known constraint or a Simulation based constraint. The fourth letter indicates if the constraint is Known or Hidden. The constraint classifications are further elaborated here below and adopted from [22].

Quantifiable vs. non-quantifiable

A quantifiable constraint is a constraint that is able to quantify if a constraint is violated or not. Quantifiability is indicated in the taxonomy by the letter Q. A non-quantifiable constraint is a constraint where it is not possible to quantify the feasibility or if the constraint is violated. Non-quantifiability is indicated in the taxonomy by the letter N. An example of a quantifiable constraint is the maximum and minimum velocity value of an object. These limits are bounded by numerical values.

Relaxable vs. unrelaxable

A relaxable constraint is a constraint that is allowed to be violated to obtain feasible solutions. Relaxability is indicated in the taxonomy by the letter R. An unrelaxable constraint is a constraint that is not allowed to be violated to obtain a feasible solution. Unrelaxability is indicated in the taxonomy by the letter U. Typically, unrelaxable constraints are constraints belonging to the physical system, whereas relaxable constraints often belong to customer-specific requests (e.g. budget or weight restrictions).

A priori vs. simulation based

A priori constraint is a constraint that can be checked without simulating the system. The a priori property is indicated in the taxonomy by the letter *A*. A simulation based constraint is a constraint that only can be checked after running a simulation. The simulation based constraint property is indicated in the taxonomy by the letter *S*. The disadvantage of simulation based constraints is that to check if such a constraint is violated, a simulation has to be conducted. Running a simulation is always costly concerning time. An example of a simulation based constraint is that a certain output value of the system is bounded and that this limit is exceeded. A convenient property of the a priori constraint is that this constraint can be checked before the simulation is run and thus saves time. An example of a priori constraint is the upper or lower limit of a decision variable.

Known vs. hidden

A known constraint is a constraint that is explicitly given during the problem definition. The known constraint property is indicated in the taxonomy by the letter *K*. A hidden constraint is a constraint that is not explicitly given during the problem definition. The hidden constraint property is indicated in the taxonomy by the letter *H*. A typical hidden simulation based constraint appears whenever the simulation crashes. These crashes are not necessarily faults in the simulator, but could also be created by the programmer to restrict certain actions during a simulation.

4.1.3 Optimization penalty function

The constraints in an optimization problem can be dealt with via multiple approaches. These approaches are grouped into the following categories, i.e. methods based on penalty functions; methods based on a search for feasible solutions; or a combination of those [23]. Methods based on penalty functions, add a penalty value to the objective value to penalize solutions that violate relaxable constraints. This summation is also referred to as the fitness value, i.e. this value is composed of the objective value and the penalty value. This implies that these solutions are feasible but less desired. Moreover, methods that are based on a search for feasible solutions, only consider solutions that do not violate any constraint. Herein, these solutions are ”repaired” and thus replaced by solutions that do not violate any constraint. This method can only be applied whenever a constraint is a priori, i.e. can be checked before simulating the system. Applying the most suitable technique depends on the classification of the constraints. Whenever a constraint is unrelaxable, the death penalty is applied. However, if a constraint is relaxable, a finite penalty value is utilized.

4.2 Optimization techniques

As mentioned in section 2.3, different types of modeling techniques are available to approach the response of a system. The most suitable method depends on the purpose of the model, the abstraction level of the model and the resources that are available. Therefore, the decision is made to model the system response via a numerical simulation model. Herein, the system is modeled as a stochastic black box. Therefore, creating a mathematical closed-form expression of the fitness function is not possible. As a result, the optimization framework is based on simulation based optimization methods. This section presents a classification of different simulation based optimization techniques. Moreover, gradient-based and gradient-free optimization techniques are elaborated. Lastly, this section presents potentially interesting gradient-free optimization algorithms.

4.2.1 Simulation based optimization technique classification

Simulation based optimization techniques are classified into different domains, depending on the properties of the problem at stake. A simulation based optimization classification can be defined based on the information obtained out of the papers and book of [24] [18] [19] [8]. These techniques are classified based on the type of the decision variables. The decision variables can be distinguished into discrete or continuous decision variables. Problems containing discrete decision variables can be further divided into problems having a finite (or small) and an infinite (or large) solution space. Whenever the solution space is considerably small, full enumeration or ranking and selection method can be applied. The full enumeration method considers all possible solutions in the system and selects the best among them. The fundament of ranking and selection is to rank the fitness values of a subset of competing candidates and to find a better alternative [25]. Whenever the solution space is considerably large or infinite, neither ranking and selection nor full enumeration can be applied directly. Using these methods results in an overwhelming required amount of computational power. To find satisfactory solutions, metaheuristics, stochastic adaptive search techniques or random search can be applied. These types of search methods only consider a subset of the entire solution space via a resourceful procedure to obtain a satisfactory solution. However, most discrete simulation based optimization techniques are unsuitable for optimization problems having continuous decision variables. The number of solutions becomes infinite due to the continual property of the decision variable. These kinds of optimization problems can be further divided into gradient-based and gradient-free optimization methods. Gradient-based methods utilize mathematical information concerning the closed-form system response expression, whereas gradient-free optimization methods do not use this information due to its absence. An example of a gradient-based optimization technique is the gradient descent technique. Examples of gradient-free optimization techniques are simulation based steepest descent or non-derivative continuous techniques. A diagram of this classification is presented in Figure 4.1.

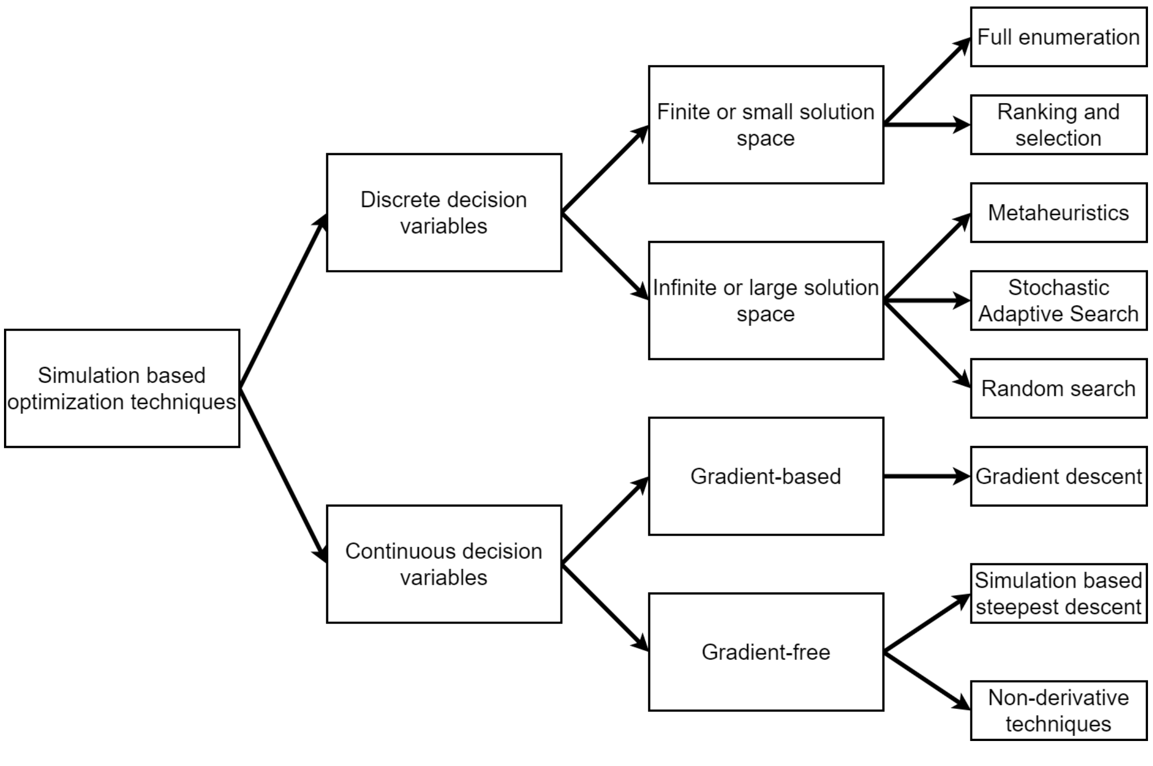

Simulation based optimization techniques
Discrete decision variables
Continuous decision variables
Finite or small solution space
Infinite or large solution space
Gradient-based
Gradient-free
Full enumeration
Ranking and selection
Metaheuristics
Stochastic Adaptive Search
Random search
Gradient descent
Simulation based steepest descent
Non-derivative techniques

Figure 4.1: Simulation based optimization techniques classification.

4.2.2 Gradient-based techniques

Gradient-based search techniques utilize the gradient of the closed-form fitness function to obtain insights concerning the optima of the system. The gradients are determined with the help of the first forward finite difference approximation of the derivative of the closed-form expression. These techniques exploit the calculated gradients to find the optima of the system response. Different methods exist that employ gradient-based techniques. The most important techniques are briefly discussed here below.

The most fundamental gradient-based method is the gradient descent method which is applied to systems with continuous decision variables. This method gives insight into the system response whenever small changes are applied to the input variables. The gradient descent technique tries to find solutions with a better fitness value by moving in small steps with the opposite sign of the derivative. An optimum or a saddle point is attained whenever the derivative in a certain point is equal to zero. A disadvantage of this technique is that the technique may fail to find a global optimum. The method finds solutions with better fitness values based on its starting point. Depending on this point, a global or a local optimum is found. This phenomena is illustrated in Figure 4.2. The same principle can be deployed for discrete optimization problems, i.e. hill climbing. Furthermore, the second derivative can be determined. This expression gives insight into how the first derivative changes as the input values are changed and provides insight into the magnitude of a step. Other techniques are available that are based on the gradient-based principle, like Perturbation Analysis, Likelihood ratios, Finite difference, Frequency domain analysis or Harmonic analysis [26] [27] [28] [29] [30].

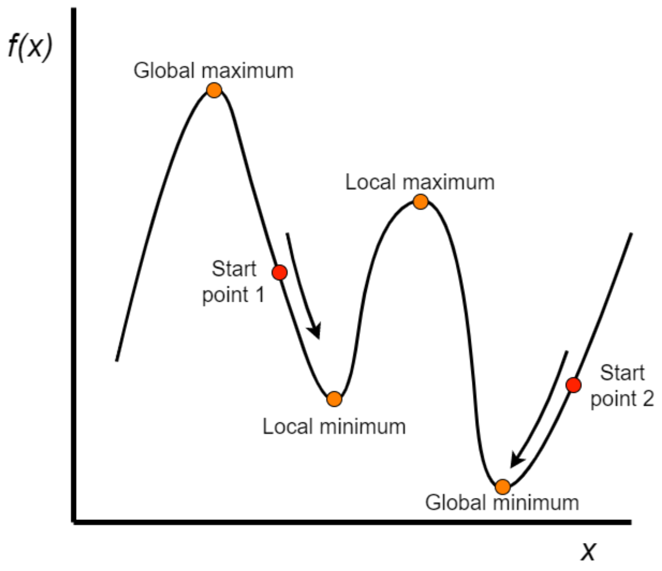


Figure 4.2: Illustrative example of gradient descent method for continuous objective functions.

4.2.3 Gradient-free techniques

Whenever a mathematical expression of the closed-form is missing, other approaches should be found to find a suitable set of decision variables that result in a satisfactory solution. For this purpose, only discrete gradient-free optimization methods are discussed. To apply gradient-free techniques, some important assumptions shall be made. First of all, it is assumed that the solution space is finite. Secondly, it is possible to estimate the value of the fitness function at any given point with the help of the simulator. As mentioned before, if the solution space is small, full enumeration or ranking and selection method could be applied. However, the assumption is made that the solution space is profoundly large. Therefore, it becomes unavoidable to find algorithms that can find good solutions without exploring the solutions space entirely. Within the simulation based optimization domain, so-called metaheuristics and stochastic adaptive search techniques are deployed. These techniques can solve discrete optimization problems where the system response can only be determined via simulation.

Metaheuristics

In general, metaheuristics are based on a random search. The decision variables are changed based on a stochastic search. This search method does not have adequate convergence properties. Therefore, metaheuristics do not guarantee finding the global optimum. However, they work well in practice for discrete problems with a large solution space. Moreover, this technique can find good solutions in a reasonable amount of time. The effectiveness depends on the neighbour selecting strategy, i.e. how is the next set of input variables chosen? One of the most used neighbour selecting strategies is called the hit-and-run strategy. This strategy updates the decision variable by randomly selecting the next or previous variable in the set of this decision variable. This is conducted for all variables and a new neighbour is created [8].

One of the most popular metaheuristics is the genetic algorithm (GA), which is inspired by Darwin's theory of natural evolution. In general, the algorithm chooses the fittest individuals among its population and uses these to reproduce and create next-generation individuals, i.e. children. This metaheuristic is used in the industry extensively with success. A second widely used metaheuristic is the tabu search algorithm. The distinctive feature of the tabu search algorithm is that this algorithm makes use of a tabu list. This list keeps track of the performed mutations and these mutations are restricted in future decision variable updates. A mutation is a process of changing the value of a decision variable from the current value to its updated value (i.e. neighbour selecting). This algorithm prohibits updates of decision variables that are already conducted to use the computational power as efficiently as possible. In the literature, different metaheuristics exist, such as scatter search, particle swarm optimization and ant clonal selection algorithm [31].

Stochastic adaptive search

The stochastic adaptive search technique has the advantage that convergence properties are proven and that it utilizes an adaptive search technique [32]. The stochastic adaptive search technique uses two characteristic features. First of all, this technique uses a stochastic mechanism to generate new neighbours. Additionally, it employs an adaptive feature to generate new neighbours based on past experiences. Algorithms of this domain are based on pure random search techniques. Pure random search is used as a benchmark to determine the effectiveness of metaheuristics or stochastic adaptive search techniques. If a more sophisticated search technique takes more effort (in terms of evaluations) than pure random search, the more advanced search technique's usability is questionable.

A widely used stochastic adaptive search technique is simulated annealing. The algorithm is based on the annealing process of metals. This technique involves the controlled cooling process of heated metals that alters the physical properties of the material. In general, this algorithm searches for better solutions and updates the best solution when a solution with a better fitness value is found. Additionally, the algorithm can accept worse solutions to escape from local optima. This process is equivalent to the cooling down process. The cooling down process of this algorithm can be interpreted as a slow decrease in the acceptance probability of worse solutions [33]. This algorithm is also widely used in the simulation based optimization domain. Different stochastic adaptive search techniques are backtracking adaptive search, stochastic ruler or nested partitions. Information concerning these algorithms can be found in [8].

4.3 Suitable SBO algorithms

As discussed in this chapter, there are multiple techniques and algorithms available to solve optimization problems. Which technique to use depends on the properties of the problem at stake. Whenever the optimization problem is continuous and a closed-form of the fitness function is available, gradient-based optimization algorithms are favourable. However, the current problem can be classified as a discrete optimization problem due to its discrete decision variables and constraints. Moreover, the closed-form of the fitness function is not available and is therefore evaluated with the help of a simulation. Hence, gradient-based optimization methods are not suitable and are not considered. The solution space of the optimization problem is large. For that reason, full enumeration nor ranking and selection techniques are deployed, as they are considered time-inefficient. Hence, metaheuristics and stochastic adaptive search techniques are investigated. These techniques produce satisfactory solutions for SBO problems having a considerably large solution space. The potentially suitable algorithms for the current optimization problem are elaborated in detail here below.

4.3.1 Monte Carlo simulation

The Monte Carlo simulation technique selects a solution in the solutions space with a certain probability and evaluates its fitness. Whenever the selected solution produces better results compared to the best solution, the best solution is updated with the selected solution. This process is repeated up and until a specified number of evaluations are conducted. This method is selected as a benchmark. Whenever the performance of the alternative methods performs worse compared to this benchmark method, the usability of the alternative methods is questionable.

4.3.2 Hybrid Genetic Algorithm

The GA is one of the proposed techniques to find satisfactory solutions for a SBO problem. A GA is an iterative optimization process that constantly changes its population by applying GA actions, like ranking and selection, crossover and mutation. This iterative process is conducted up and until some stop criterion is met concerning the fitness function or the number of evaluations. In a GA, a set of decision variables is called a chromosome. A chromosome has the length of the number of decision variables (k) and is captured in a vector $\vec{x} = [x_1, x_2, ..., x_k]$. Each decision variable in a chromosome is referred to as a gene. Within the GA multiple chromosomes are defined per generation. A set of N_p chromosomes is referred to as a population. An example population is presented in Figure 4.3. Each iterative step where the population is changed is called a generation. In this figure, a single chromosome is framed by the red box and a single gene is framed by the green box [31].

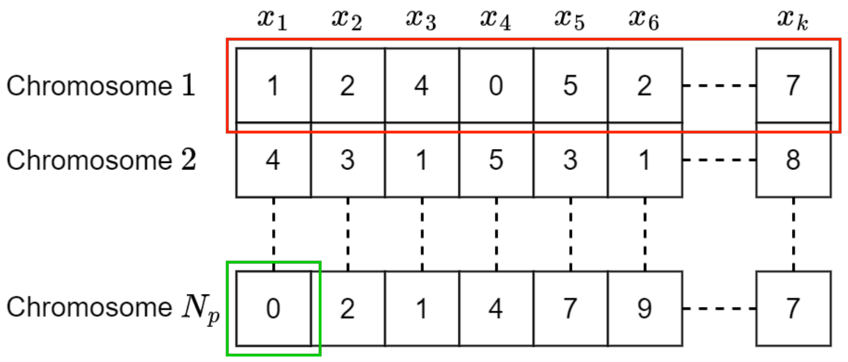


Figure 4.3: Illustration of a population of the GA.

Normally, a GA is classified as a global search technique because the method considers the entire solution space. Different operators are incorporated to enhance the performance of this optimization technique. Because a traditional GA does not consider the neighbourhood of a certain solution directly, a local search operator is included. This extension combines global and local search to boost the performance. These types of GAs are referred to as Hybrid genetic algorithms (HGA) [34]. The proposed HGA is based on the HGA presented in the journal article of Gharsalli and Guerin (2019). They concluded that a HGA with strong cooperation between GA and an iterative local search technique (in sequence) results in a good performance. Herein, local search is applied to the best chromosomes of the previous generation to intensify research only around promising solutions. Applying local search to all chromosomes is a waste of computational resources. They also studied a framework where local search is applied in parallel with the GA. However, they concluded that the results of the framework where the local search is applied in sequence results faster in satisfactory solutions [35]. The flowchart of the proposed HGA for the current project is presented by Figure 4.4. The operators of the HGA are discussed in the text here below.

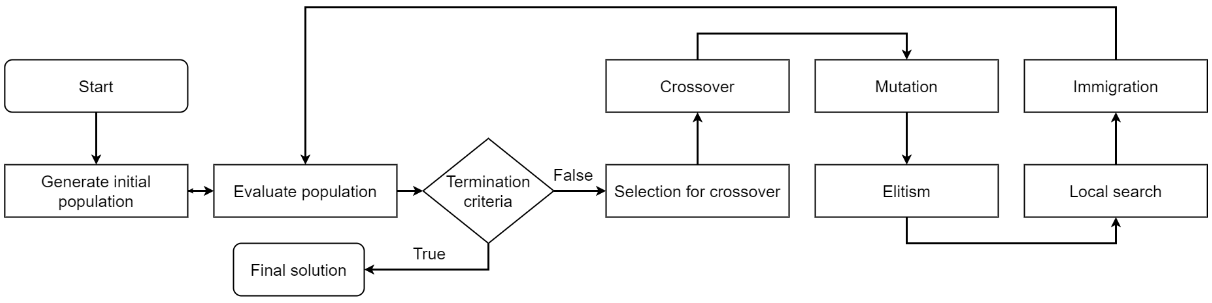


Figure 4.4: Flow diagram of the proposed HGA.

Generate initialize population

The first step of the HGA is to initialize a population. The size of the population is equal to N_p chromosomes. The size of a chromosome is equal to the number of decision variables. During the initialization, random values are assigned to these variables. Hereafter, the first generation is evaluated and these results and data are used to define the next population's generation.

Selection for crossover

The first GA operation consists of the ranking and selection procedure. During this procedure, the evaluated population of the previous generation is partly ranked and the best scoring chromosome is selected to be used in the crossover operation. The selection procedure can be performed via various methods. In the current HGA, the K-way tournament selection method is used. The K-way tournament selection procedure selects K chromosomes of the previous generation randomly and chooses the best performing chromosome of this set to become a parent. The selection procedure is conducted up and until all parents are selected. The selected parents are, hereafter, used to create offspring chromosomes via crossover and mutation.

Crossover

The offsprings of the current generation are generated with the help of the crossover operator, by recombining two parent chromosomes into two offspring chromosomes. This operator uses the genetic information of two good parents to create promising children. Crossover is applied to a pair of parent chromosomes with a high probability. This probability is referred to as the crossover probability. This probability is one of the HGA parameters. Keep in mind, that crossover between two good solutions does not guarantee better solutions. The idea behind this approach is that whenever the parents have good traits, the children could have good traits as well with a high probability.

In the literature, different crossover methods are presented. The uniform crossover method is one of the most used methods for bounded parameter sets. This method considers the limits of the parameters individually and does not change the values of these parameters to values that are not in the parameter set, like crossover methods that change the order of a chromosome (i.e. order crossover techniques (used on scheduling problems)) [36]. The uniform crossover method considers all genes of a chromosome. The values of the genes are swapped with a certain probability, i.e. crossover probability. This method is illustrated in Figure 4.5.

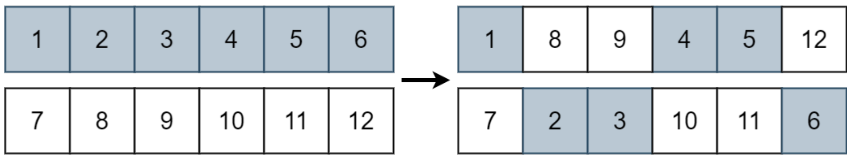


Figure 4.5: Illustrative example to demonstrate the uniform crossover operator.

Mutation

The mutation operator is applied to maintain diversity in the genetic population. This diversity is required to ensure that the chromosomes of the population will not be trapped in a local region. The mutation operator is applied to every offspring generated by the crossover operator. The mutation operator considers every gene individually. The mutation action is applied to a gene with a low probability, i.e. mutation probability. Whenever a gene mutates, the value of the gene is replaced by a randomly chosen value belonging to the parameter set of the corresponding decision variable. Figure 4.6 presents an example of the mutation operator. In this example, the second gene is mutated. The parameter set of the second decision variable is equal to $S_2 = [1\ 2\ 3\ 4\ 5\ 6]$. The mutation operator replaces the old value with a new value, randomly selected out of this set.

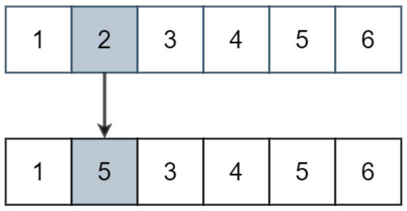


Figure 4.6: Illustrative example to demonstrate the mutation operator.

Elitism

The fundamental trait of the GA is the selection of the best individuals, based on their fitness value, to be used for the next generation. To maintain this trait, the elitism operator is used. This operator ranks all chromosomes of the previous generation based on their fitness value. Thereafter, the top n individuals are selected and copied into the new population. These chromosomes are, thereafter, used by the local search operator. The elitism operator ensures that the best found solutions remain in the population and are even further explored [37].

Local search

After the top n chromosomes are selected and copied to the new generation, local search is applied to these chromosomes. The idea behind this operator is to further explore the neighbourhood of the best found solutions for these elite chromosomes. For this purpose, local search is applied to these elite chromosomes. The local search operator considers these solutions one by one. A neighbouring solution of a selected elite chromosome is defined. Hereafter, this solution is simulated and evaluated. Whenever the neighbouring solution performs better, the elite chromosome is replaced by the neighbouring chromosome. Then, this process continues for the new elite chromosome. This process is applied up and until no fitter chromosome is found for max_{LS} iterations. Whenever this stop criterion is met, the same search is conducted for the remaining elite chromosomes. This search procedure is illustrated by Figure 4.7. Where num_{LS} is the number of local searches that have not resulted in improvement, C_{elite} is the current elite chromosome, $C_{\text{neighbour}}$ is the created neighbour, f_{best} is the fitness value of the elite chromosome and $f_{\text{neighbour}}$ is the fitness value of the neighbouring chromosome.

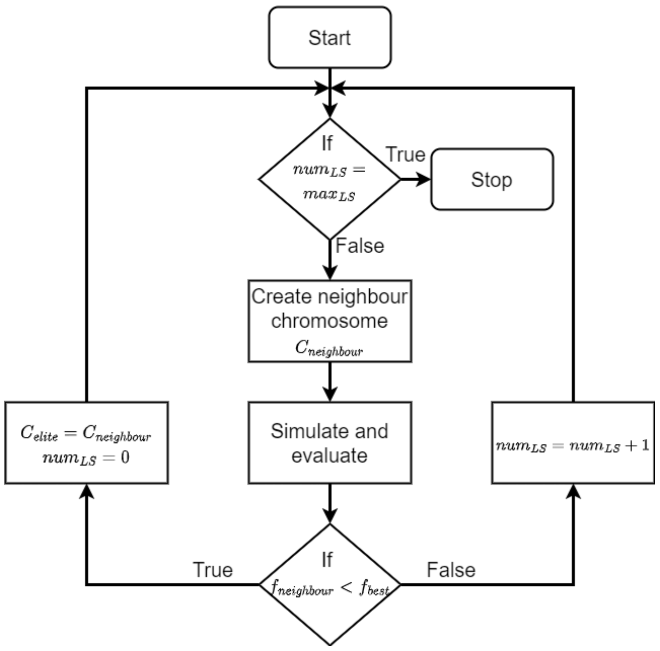

Start
If $num_{LS} = max_{LS}$
True
Stop
False
Create neighbour chromosome $C_{neighbour}$
Simulate and evaluate
If $f_{neighbour} < f_{best}$
True
False
$C_{elite} = C_{neighbour}$ $num_{LS} = 0$
$num_{LS} = num_{LS} + 1$

Figure 4.7: Flow diagram of the local search procedure.

The purpose of the local search is to define a chromosome that lies in the neighbourhood of the elite solution, implying that the decision variables should be changed by only a small amount. Therefore, a neighbour of the elite chromosome is defined via the following procedure. The decision variables of the system that is under investigation are discrete, suggesting that the new value of the decision variable depends on its own parameter set. For this purpose, the new value of the decision variable is only allowed to become the old value or one of the neighbouring values. This principle is illustrated via an example presented in Figure 4.8. The current value of the decision variable is equal to three. Therefore, the new value of the decision variable is only allowed to become two, three or four. This process is repeated for every gene. Via this procedure, a new neighbouring solution is defined.

$$S = [1\ 2\ 3\ 4\ 5\ 6] \longrightarrow S = [1\ 2\ 3\ 4\ 5\ 6]$$

Figure 4.8: Illustrative example of the selection of a new value of the discrete decision variable.

Immigration

Immigration is applied as a last GA operator and replaces all duplicate chromosomes in the population with randomly generated chromosomes. This operator is applied to explore the solution space as efficient as possible. There is no need to evaluate duplicate chromosomes in the same population. The elitism and immigration operators provide the possibility to explore the solution space as efficient as possible while maintaining the same level of exploitation for the used population size.

Evaluation

The fitness of each chromosome is evaluated whenever a population is generated. The fitness is based on the objectives and constraints of the optimization problem. A lower fitness value is equivalent to a better solution. Therefore, the high-level goal of the optimization problem is to minimize the fitness function. Additionally, a solution database is consulted before a solution is simulated. The results for an already simulated solution are copied out of this database. This feature ensures that the computational power is used as efficiently as possible. It is a waste of resources to simulate an already evaluated solution.

Termination

The HGA is terminated whenever a stop criterion is reached. Whenever these are not met, a new generation of chromosomes is constructed via the above-described operators. In general, a HGA is terminated whenever a specified number of generations have been evaluated, a satisfactory solution is found or when a certain convergence term is met concerning the improvement of the solution. Then, the chromosome with the lowest fitness value in the population is selected. This chromosome is selected as the final solution of the HGA.

Hybrid genetic algorithm parameters

The HGA has some algorithm-specific parameters influencing the performance of the optimization method. Choosing the values of these parameters is key to finding satisfactory solutions. Whenever this algorithm is adapted to the optimization framework, a study has to be conducted to select proper values. The HGA parameters are the population size, crossover rate, mutation rate, tournament rate, elitism rate and the maximum number of local search iterations.

4.3.3 Simulated annealing

The simulated annealing (SA) algorithm is the second proposed algorithm to find satisfactory solutions for a SBO problem. A SA is also an iterative optimization problem that constantly evaluates a neighbouring solution to find a more promising solution. As mentioned before, SA searches for better solutions and updates the best solution whenever a solution with a better fitness value is found. Additionally, the algorithm can accept worse solutions to escape from a local optimum. The probability that the algorithm accepts solutions that perform worse depends on the acceptance function (that is a function of a decreasing temperature value). A solution generated via the SA algorithm is equivalent to a single chromosome in the HGA. Only one solution at a time is considered. The proposed SA algorithm is based on the algorithm presented in [33]. The proposed algorithm utilizes two loops. The algorithm first evaluates different neighbouring solutions for a certain temperature value. The temperature value is updated after a specified number of iterations. The temperature value influences the acceptance function of the algorithm whenever a neighbouring solution does not outperform the current solution, as mentioned before. The flow diagram of the proposed SA algorithm is presented in Figure 4.9. The operators of the algorithm are discussed in the text here below.

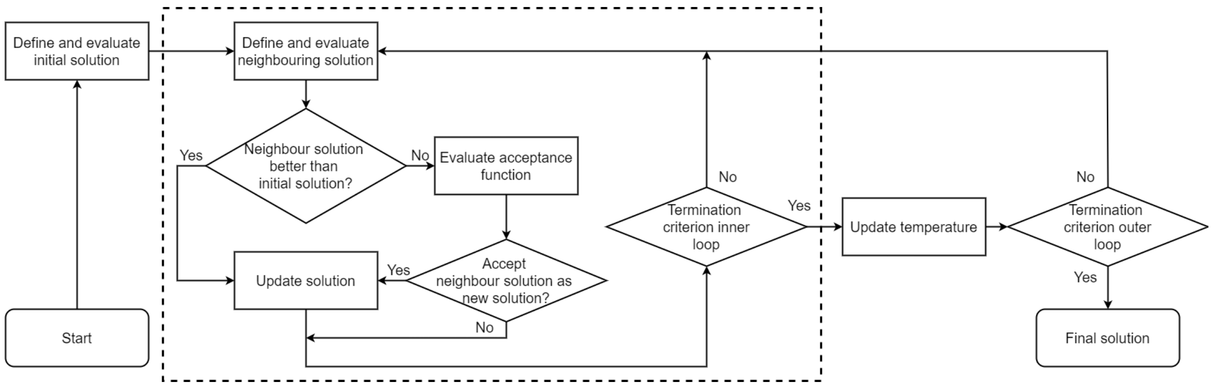

Define and evaluate initial solution
Start
Define and evaluate neighbouring solution
Neighbour solution better than initial solution?
Yes
No
Evaluate acceptance function
Update solution
Yes
Accept neighbour solution as new solution?
No
Termination criterion inner loop
No
Yes
Update temperature
Termination criterion outer loop
No
Yes
Final solution

Figure 4.9: Flow diagram of the proposed SA algorithm.

Define an initial solution

The first step of the SA algorithm is to initialize the first solution. The size of this solution is equal to the number of decision variables. During this initialization process, random values are assigned to the parameters. Hereafter, the solution is evaluated and the defined initial solution is selected as the current best solution.

Acceptance function

Thereafter, a new neighbouring solution is defined. The new solution lies in the neighbourhood of the current best solution. The neighbouring solution is defined via the same approach as discussed in the subsection concerning the local search operator of the HGA. Hereafter, the neighbouring solution is evaluated. The current best solution is replaced by the neighbouring solution whenever the new solution outperforms the current best solution concerning its fitness value. Whenever this is not the case, the acceptance function is evaluated. As mentioned before, the acceptance function provides the possibility to escape local minima. The probability that the algorithm accepts a solution is based on the Metropolis acceptance criterion. This criterion portrays how a system changes its state in which the energy is minimized. This criterion is used to decide whether to accept the neighbouring solution or not [38]. In the context of the SA algorithm, the acceptance probability depends on the difference between the fitness value of the neighbour solution, the current best solution and the current temperature value. This probability is presented by Equation 4.2. Where p is the acceptance probability, Δf is the difference between the fitness value of the created neighbour and the current best solution and T_{current} is the current temperature value.

$$p = \begin{cases} 1, & \text{if } \Delta f < 0. \\ e^{-\Delta f / T_{\text{current}}}, & \text{otherwise.} \end{cases} \tag{4.2}$$

Termination criteria

After determining if the current best solution shall be replaced by the defined neighbouring solution, the algorithm reaches the termination criterion of the inner loop (see Figure 4.9). This termination criterion is responsible for the update process of the temperature of the system. Each temperature value is used for a specified number of iterations. Whenever the number of iterations is exceeded, the temperature is updated. Additionally, the termination criterion of the outer loop is responsible for the termination of the SA algorithm. Whenever a specified final temperature value is reached, the system is "cooled down" enough and the search is terminated. Then, the current best solution is defined as the final solution.

Update temperature

The update temperature operator belongs to the outer loop of the algorithm. The algorithm starts with an initial temperature value. After termination of the inner loop, the temperature value is decreased towards the final temperature value which results in termination of the algorithm. The temperature value is updated with the help of a cooling schedule. This schedule is a critical part of the algorithm. A very slow cooling schedule results in many iterations until a final solution is found. However, a very fast cooling schedule can result in an algorithm that gets trapped in a local optimum, due to the temperature value updates with large steps. Therefore, a fast cooling schedule results in a system that accepts little worse solutions due to the fast decrease of the acceptance probability. In the literature, different cooling schedules are introduced, e.g. logarithmic, geometric and exponential schedules. The geometric schedule is more often used in practice [33]. This relation is given by Equation 4.3. Where T_{update} is the updated temperature value and β is a factor that decreases the temperature value with a specific rate. The value of β lies typically between 0.8 and 0.99 [39]. A smaller value results in faster cooling.

$$T_{\text{update}} = \beta \, T_{\text{current}} \tag{4.3}$$

Simulated annealing algorithm parameters

The SA algorithm also has some algorithm-specific parameters influencing the performance of the optimization method. Choosing the values of these parameters is key to finding satisfactory results. Whenever this algorithm is adapted into the optimization framework, a study has to be conducted to select proper values. The SA algorithm parameters are the number of iterations per temperature step, the initial temperature value, the final temperature value and the cooling factor.

Chapter 5

Tubtrax optimization framework

This chapter presents the optimization framework used in this project to solve Tubtrax optimization problems. Herein, the framework is elaborated in detail, the algorithm used in the optimization framework is tuned, the performance of the HGA is analyzed and two case studies are worked out to demonstrate the capabilities of the developed optimization framework.

5.1 Proposed optimization framework

First of all, the proposed optimization framework is manifested. Herein, the fitness functions, the constraints, the penalty function and the framework's structure are presented. Moreover, the used algorithm to update the decision variables of the system is selected.

5.1.1 Optimization fitness function

Two optimization objectives are identified in this project. Therefore, two unique fitness functions are particularized. The first fitness function is composed of an objective and a penalty function. This objective function is related to the requirement and the capacity of a given configuration. Moreover, the penalty function is related to the violation of the constraints. The penalty value is increased by a finite value whenever a relaxable constraint is violated, implying that this solution is still feasible but has some less favourable characteristics. Moreover, the penalty value is increased by an infinite value whenever an unrelaxable constraint is violated and this solution is labeled infeasible. The goal is to find a satisfactory set of system parameters for a given configuration that results in the best capacity, meets the requirements and does not violate the constraints. The first fitness function is given by the relation presented in Equation 5.1. Where $f_1(\vec{x})$ is the fitness function related to the requirement and capacity of a given configuration, $K(\vec{x})$ is a function related to the requirements and the capacity and $P(\vec{x})$ is the penalty function. $K(\vec{x})$ depends on the goal of the optimization problem. Herein, the requirements and the capacity are captured. Which requirements are considered depends on the use case. Moreover, the capacity term in this function is provided as seconds per carrier. Therefore, a lower value corresponds to a higher capacity value and is more favourable, whereas a higher value corresponds to a lower capacity value and is less favourable.

$$f_1(\vec{x}) = K(\vec{x}) + P(\vec{x}) \tag{5.1}$$

The second optimization objective is linked to the cost of the system. The Tubtrax system can be defined via various configurations, i.e. different combinations of conveyor types and lengths. But why is a certain variant preferred over another? This is related to the total cost of a system. Therefore, this fitness function depends on the number of used conveyors and the footprint. This fitness function does not include a penalty function and is given by the relation presented in Equation 5.2. Where $f_2(\vec{x})$ is the fitness function related to the cost of the configuration, c_{conveyor} is the cost per used conveyor, $c_{\text{footprint}}$ is the cost per used length, $S_{\text{lengths, i}}$ is the length of conveyor i and n_s the total number of used conveyors in the configuration.

$$f_2(\vec{x}) = c_{\text{conveyor}}\, n_s + c_{\text{footprint}} \sum_{i=1}^{n_s} S_{\text{lengths, i}} \tag{5.2}$$

The goal of the optimization problem is to find a system that is as cheap as possible, has the highest capacity, satisfies the requirements and does not violate the constraints, implying that both fitness functions have to be minimized. A final solution is defined as a solution where both fitness functions are as low as possible.

5.1.2 Optimization constraints

The optimization framework considers multiple constraints that are explicit and implicit. Moreover, it is possible to enable and disable each constraint, i.e. consider it in the optimization study or not. Furthermore, it is possible to specify if a constraint is relaxable or unrelaxable. The constraints that are identified in the system are discussed here below.

- **Constrained velocity delta:**
  This constraint is related to the velocity delta per time step. The step in velocity of a carrier or baggage item is only allowed to change based on the acceleration value, implying that abrupt stops and accelerations are not allowed. This behaviour is only possible when the conveyors are controlled inappropriately. For example, the first conveyor has a velocity value of $v_1 = 2$ m/s and the second conveyor has a velocity value of $v_2 = 0$ m/s. Implying that whenever a carrier proceeds from the first conveyor to the second conveyor, the velocity delta becomes $\delta_v = 2$ m/s. This value is too large and, therefore, this constraint is violated (under the assumption that the time step is equal to $T = 0.01$ s and the acceleration value is equal to $a = 4$ m/s^2). This constraint is a *QUSK* constraint (taxonomy presented in subsection 4.1.2). The constraint is quantifiable and unrelaxable. Behaviour induced by the violation of this constraint is undesired and should be avoided at all costs. Additionally, this constraint is simulation based and can only be checked after the simulation is run. Finally, the constraint is specified beforehand and therefore known.

- **Collisions:**
  This constraint is related to the collision of carriers or baggage items. Collisions between these objects in the system are undesired behaviour and therefore prohibited and should be avoided at all costs. This constraint is also a *QUSK* constraint.

- **Slip:**
  The slip constraint is related to the constrained velocity delta constraint. Within the model, a carrier is controlled by the conveyor that contains the physical majority of the carrier. Slip occurs whenever a carrier is physically on two conveyors and the velocity of these conveyors is not equal to each other. The velocity delta constraint is violated whenever the velocity difference is still present whenever the control of the carrier is handed over to the next conveyor. However, when that is not the case and the velocity difference between the conveyors is zero whenever the control of the carrier is handed over, only slip occurs. Slip is allowed depending on the problem and could potentially result in position inaccuracy. This constraint is classed as a *QRSK* or *QUSK* constraint.

- **Physical control:**
  The physical control constraint is related to one of the design rules of the Tubtrax system (as discussed in subsection 3.1.1). This constraint checks if the carrier is not controlled too early. A carrier is only allowed to be controlled (i.e. change its velocity value) when a carrier is physically for a specified fraction on the conveyor that wants to control the carrier. Otherwise, undesired behaviour could occur, like slip. This constraint is a *QRAK* or *QUAK* constraint. The constraint can be quantified and it is possible to specify if the constraint is relaxable or not. Additionally, the constraint can be checked before the simulation is run. Finally, the constraint can be specified beforehand and therefore known.

- **Simulation error:**
  The simulation error constraint is a constraint that is hidden and erupts whenever the simulation is terminated due to an error. These errors are the result of invalid inputs or behaviour that is restricted by the programmer. Therefore, this constraint is a *NUSH* constraint.

5.1.3 Optimization penalty function

A penalty function is included to penalize solutions that violate constraints. subsection 5.1.2 presents the constraints that can be considered in the optimization framework. How this penalty function is defined depends on the constraints that are included and violated. As mentioned before, it is possible to specify if a constraint is considered and if the constraint is relaxable or not. Moreover as discussed, a finite penalty value is added whenever a relaxable constraint is violated and an infinite penalty value is added whenever an unrelaxable constraint is violated.

The physical control and slip constraint are constraints that can be relaxable or unrelaxable, implying that whenever the constraint is unrelaxable and is violated the penalty value is infinite. Moreover, whenever the constraint is relaxable and violated the penalty value depends on the amount of violation, implying that solutions that only violate the physical control constraint by a couple of centimeters are more favourable than solutions that violate this constraint by a meter. Furthermore, the amount of violation of the slip constraint is expressed in the time that slip occurs. The constrained velocity delta, collisions and simulation error constraints are constraints that are never allowed to be violated, implying that whenever these constraints are violated, an infinite value to the penalty function is assigned and the solution is labeled as infeasible. Moreover, a penalty factor is added to the penalty function to influence the impact of the penalty on the fitness function. This relation is given by Equation 5.3. Where P_{factor} is the penalty factor and P_{total} is the total penalty value of the simulated system. The question may arise what an appropriate value for this factor is. This effect is studied in subsection 5.3.2.

$$P(\vec{x}) = P_{\text{factor}}\ P_{\text{total}} \tag{5.3}$$

5.1.4 Optimization framework design

An optimization framework is designed to provide the possibility to find feasible and satisfactory solutions in an automated procedure. The framework is responsible for defining potential solutions via an agile strategy. Moreover, the framework evaluates the defined solutions and explores the solution space to find better ones. The optimization framework used for this project exists of four modules. This framework is presented in Figure 5.1. The modules are elaborated in the text here below.

Initialize system

The initialize system module is only run once and initializes the framework. First of all, all different layout possibilities are defined. Beforehand is defined what conveyor lengths and the number of consecutive conveyors behind each other are considered. This principle is illustrated with the help of an example. The set of available conveyor lengths is equal to $S_{lengths} = [1,\ 2,\ 3,\ 4]$ meter. Additionally, the number of conveyors that are allowed to be used is specified. In this example, the lower bound is equal to $lb = 3$ conveyors and the upper bound is equal to $ub = 4$ conveyors, implying that the total number of conveyor configurations is equal to $4^3 + 4^4 = 320$ possibilities. The number of possibilities grows exponentially whenever the number of consecutive conveyors increases. Therefore, unfavourable conveyor configurations are omitted to reduce the number of possibilities. In general, the length of the last conveyor does not influence the response of the system. This conveyor is in most cases a runout conveyor of the system. Moreover, special functionalities often require a minimum conveyor length. For example, some use cases require multiple dieback points on a single conveyor. If it is not possible to define these given the imposed constraints, the algorithm omits configurations that are not capable of offering this functionality on a particular conveyor length. Hence, it is recommended to omit configurations that do not produce feasible or satisfactory results.

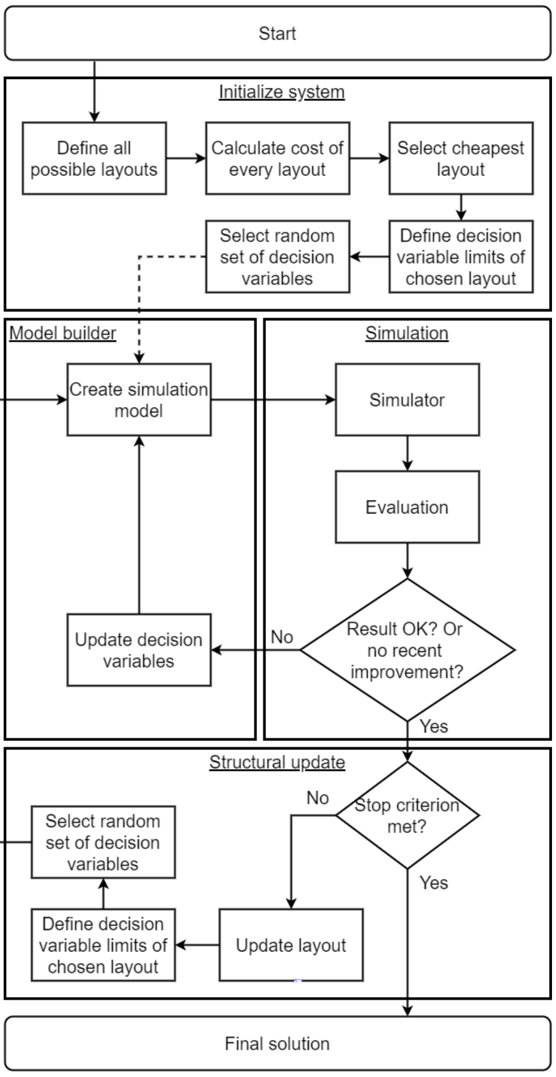

Start
Initialize system
Define all possible layouts
Calculate cost of every layout
Select cheapest layout
Define decision variable limits of chosen layout
Select random set of decision variables
Model builder
Create simulation model
Update decision variables
Simulation
Simulator
Evaluation
Result OK? Or no recent improvement?
No
Yes
Structural update
Stop criterion met?
No
Yes
Select random set of decision variables
Define decision variable limits of chosen layout
Update layout
Final solution

Figure 5.1: Flow diagram of the proposed optimization framework.

Hereafter, all possible configurations are ranked based on the configuration's cost. This cost term corresponds to the second fitness function as presented by Equation 5.2. Thereafter, the cheapest layout configuration is chosen and the limits of the decision variables and the velocity settings are set. These limits are chosen thoughtfully. Considering decision variables that result in infeasible or unsatisfactory solutions has no added value. As a consequence, this results in a waste of computational resources. Precautionary measures are introduced to bound these limits. First of all, defining the position of the dieback point not on the conveyor makes no sense. Moreover, defining the position of the dieback point too close to the conveyor's begin also makes no sense. Whenever the position of the dieback point is defined too close to the begin of the conveyor, the decision point lies on its upstream conveyor, implying that deceleration takes place on the upstream conveyor. Moreover, the physical control constraint cannot be violated. Therefore, this behaviour is undesired and restricted. Secondly, the position of the speed change point can be bounded depending on the length of a conveyor. This point should be defined on the conveyor and the point should not be defined too close to the conveyor's begin. Thirdly, the downstream release gap belonging to a dieback point is only defined whenever an interface does not have a speed change action enabled. In this project, the downstream release gap is not considered whenever a speed change action is defined on that particular conveyor. Additionally, the bounds of the downstream release gap should be chosen sensibly whenever this parameter is enabled. The lower bound of the downstream release gap is quite straightforward. In practice, a rule of thumb is used that the gap between the carriers should have a minimum distance. Therefore, this minimum value is used as the lower bound. The upper bound could in theory be infinite. However, this would be cumbersome and not realistic. Therefore, the upper bound is equal to the distance from the dieback point up and until the end of the downstream conveyor. The velocity settings are determined based on the begin and end velocity value of the system and where optional speed change takes place. After the limits of the decision variables and the velocity settings are defined for the selected configuration, a random set of decision variables is selected. These values lie within the set limits and are chosen randomly. After this action, the initialize system module has executed all its tasks and the system can be built via the model builder module.

Model builder module

The model builder module is responsible for creating the simulation model and after evaluation to update the decision variables. A simulation model is created based on the information obtained from the previous actions. With the selected layout, velocity settings and the selected decision variable values, the model builder module defines an executable simulation model.

Simulation module

The system can be simulated after an executable model is defined and is performed in the simulation module. Herein, the model is run via the created simulation framework as presented in chapter 3. The optimization framework evaluates the performance of the created system with the output of the simulation framework. This output is evaluated with the help of the first fitness function as presented in Equation 5.1. A solution with a lower fitness value corresponds to better performance compared to a solution with a higher fitness value. After the evaluation, the first termination decision point is reached. This termination criterion depends on the used decision variables update algorithm. The termination criteria of the HGA and the SA algorithms are discussed in subsection 4.3.2 and subsection 4.3.3, respectively. Whenever the stop criterion is not met, the framework executes the update decision variables action. This action creates a new solution with the same system layout by updating the decision variables. Hereafter, a new simulation model is created, simulated, evaluated and the first termination criterion is considered again. Whenever the termination criterion is met, a final solution for the selected system layout is found. This could imply two things. First of all, a solution is found that has not improved for a specified number of iterations. Secondly, the maximum number of iterations is attained. Nevertheless, the framework proceeds to the structural update module.

Structural update module

The structural update module is responsible for updating the system's layout whenever the second stop criterion is not met. The second stop criterion checks two conditions. First of all, the stop criterion checks if the maximum number of iterations is attained, i.e. iterations concerning updating the system's layout. Secondly, the stop criterion checks if the fitness value of the system (Equation 5.1) has improved concerning the previously found solutions. Whenever no improvement is booked for a specified number, the framework is terminated and a final solution is found. Whenever the second stop criterion is not met, the optimization framework continues its search. As a result, the used system layout is updated and the next promising layout is selected concerning its costs. Again, the velocity settings and appropriate decision variables limits are set. Hereafter, the values of the decision variables are chosen randomly and this information is sent to the model builder module. Thereafter, the same procedure is executed to find a satisfactory solution for the selected system layout.

5.1.5 Update decision variable algorithm

Some special attention is paid to the algorithm that updates the decision variables. Three algorithms are potentially suitable (as discussed in section 4.3) to update the decision variables, i.e. Monte Carlo simulation, hybrid genetic algorithm and simulated annealing algorithm. These algorithms are evaluated concerning their performance. It is not only critical that an algorithm produces satisfactory results, but the algorithm should also not take too much time to produce these results. As mentioned before, the efficiency of the used algorithm is a key factor in determining the suitability of an algorithm. Whenever an algorithm does not produce quicker satisfactory results as via the Monte Carlo simulation approach, the practicality is questionable. Therefore, an analysis is conducted that studies the performance of these algorithms.

The optimization problem is restricted to study the performance of these algorithms. Only one layout is used during these experiments. The structural update part of the optimization framework is, therefore, omitted. The goal of the restricted problem equals the goal of the system used during the sensitivity analysis of the simulation framework. The requirement is to create a gap between the carriers on the last conveyor of a specified value. This optimization framework aims to find a set of decision variables that satisfies this requirement, do not violate the unrelaxable constraints and do violate the relaxable constraints as little as possible. During this study, only the first fitness function is utilized. Herein, the function related to the requirements and the capacity of a given configuration is associated with the requirement difference. The requirement difference equals the absolute distance between the attained and the desired gap between the consecutive carriers. Moreover, the term associated with the capacity is omitted, due to the capacity is fixed (i.e. the capacity is not required to be minimized) because the distance between every carrier has to be the same and the velocity value at the last conveyor is constant. The fitness function used for this analysis is presented by Equation 5.4. Herein, d_{attained} is the actual attained gap and d_{desired} is the desired gap. Moreover, some assumptions are made concerning the algorithm that updates the decision variables before the experiment is conducted. The HGA is terminated whenever no recent improvement is booked or whenever the fitness value is equal to zero (i.e. hit the target). Therefore, the value of the no recent improvement parameter is chosen rather large. This is done to study the full potential of the algorithm. Moreover, the SA algorithm is terminated whenever a specified final temperature is reached. Additionally, the algorithm parameters of the HGA and SA are chosen based on previous run simulations. Some special remarks are made concerning the SA algorithm parameters. The start and final temperature values are chosen such that the random search at the beginning of the loop is limited to a couple of hundred evaluations. The random search at the beginning provides the possibility to search the entire solution space and focus on a local minimum thereafter.

$$f_1(\vec{x}) = K(\vec{x}) + P(\vec{x}) = |d_{\text{attained}} - d_{\text{desired}}| + P(\vec{x}) \tag{5.4}$$

The results of this study are presented in the figures here below. Figure 5.2 depicts the fitness value against the number of evaluations for the HGA, SA algorithm and the Monte Carlo simulation approach. Moreover, Figure 5.3 depicts the requirement difference against the number of evaluations for the HGA, SA algorithm and the Monte Carlo simulation approach. As can be observed, the HGA finds solutions with a lower fitness value quicker compared to the other algorithms. Furthermore, the SA algorithm and Monte Carlo approach get stuck in a local minimum that does not satisfy the requirement, whereas the HGA finds a solution that meets the requirement. The positive value of the fitness value is due to the effect of the penalty function, i.e. a relaxable constraint is violated. The SA algorithm does not satisfy the requirement during this experiment. Multiple experiments have to be conducted to circumvent the effect of getting stuck in a local optimum. However, the HGA algorithm always finds a solution where

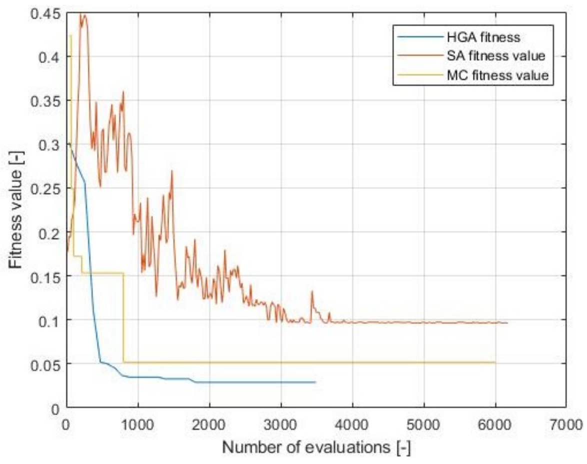


Figure 5.2: Fitness value of the HGA, SA algorithm and Monte Carlo plotted against the number of evaluations (layout index 7).

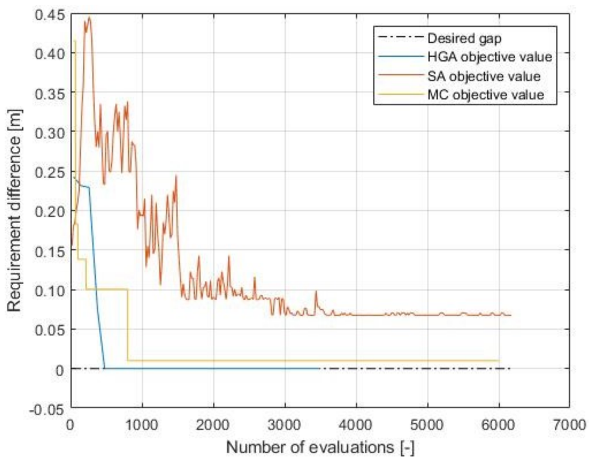


Figure 5.3: Objective value of the HGA, SA algorithm and Monte Carlo plotted against the number of evaluations (layout index 7).

the requirement is met (conclusion based on multiple runs), implying that the HGA finds more consistent and faster satisfactory results compared to the SA algorithm for the chosen layout. Moreover, the Monte Carlo approach finds a solution in the neighbourhood of the final solution of the HGA. Applying the HGA to update the decision variables resulted in a solution with a fitness value of 0.029 after less than 2000 evaluations, whereas the SA algorithm only finds a solution with a fitness value of 0.097 after more than 4000 evaluations. Moreover, the Monte Carlo approach has processed 6000 evaluations, which resulted in 279 feasible solutions (i.e. only 4.65% of the evaluations resulted in a feasible solution) and the best fitness value of 0.052. To study if the same conclusion can be drawn for different layouts, the same experiment is conducted with a different set of conveyor lengths. These results are presented in section C.1. During this experiment, the SA algorithm and Monte Carlo approach find solutions where the requirement is met. However, the HGA finds a feasible and satisfactory solution with fewer evaluations.

The HGA produces consistent and with fewer evaluations satisfactory results compared to the other methods. Moreover, the HGA finds fast (after a couple of hounded evaluations) solutions that lie in the neighbourhood of the final solution. The number of evaluations of the HGA during this study is large and can be reduced during normal application. Moreover, the advantage of the HGA is that the algorithm explores the entire solution space and that due to the elitism and local search operator the most promising solutions are maintained and further explored. Additionally, the HGA can escape a local optimum fairly easily. Nevertheless, a disadvantage of the HGA is that the algorithm is complex and has a lot of algorithm parameters that require tuning. On the contrary, the advantage of the SA algorithm is that the algorithm is simple, quickly implemented and has a small number of algorithm parameters. However, the disadvantage of the SA algorithm is that the algorithm can be stuck in a local optimum. Therefore, the algorithm requires multiple runs to obtain satisfactory results, which is computationally expensive. Moreover, the HGA outperforms the Monte Carlo approach in both experiments, where the SA algorithm does not always produce better solutions. As a result, the HGA is identified as the most promising algorithm and is used to update the decision variables. Only one algorithm is used to update the decision variables in the remainder of the report due to the time required to tune the algorithm's parameters and to execute the optimization framework.

5.2 Algorithm parameter tuning

As concluded, the HGA is used to update the decision variables in this optimization framework. This algorithm has multiple algorithm specific-parameters that have a significant influence on the performance. Therefore, it is important to find appropriate values for these parameters. The HGA possesses the following algorithm parameters: crossover rate, tournament rate, mutation rate, population size, elitism rate and the maximum number of not improved local search iterations. For example, using a small population size results in a less diverse population but is less computationally expensive. However, increasing the population size results in a more diverse population but requires more time to evaluate. Therefore, finding a good balance in the values of these is of importance. In practice, many algorithms are applied where these parameters are chosen either arbitrarily or based on previously conducted studies. To use the full potential of the HGA, a proper scheme is proposed to tune the algorithm's parameters. Design of experiments (DOE) is a common approach in combination with (hybrid) genetic algorithms to tune these parameters. This approach is based on statistical data that utilizes the empirical results of the algorithm. This approach is used in practice to test heuristic methods and give direction concerning the choice of the design parameters [40].

The parameters within the DOE scheme that are tuned are referred to as factors, whereas the output variables (i.e. results of the simulation) are referred to as the response. The levels correspond to the number of possibilities per factor that are considered. In general, two levels are used, i.e. high and low level. The high level corresponds to the upper limit of a factor and the low level corresponds to the lower limit of a factor. The DOE scheme is used to study the effect of factors on the response and to discover relations between the factors. In this project, the scheme is used to determine which parameters have a significant impact on the response. Additionally, the DOE scheme is used to choose an appropriate value for these factors.

The tuning procedure consists of multiple iterations. During each iteration, an experiment is conducted to determine which factors have a significant influence on the response. Hereafter, the most significant factors are equalized to a constant value and the remaining parameters are varied. This procedure is repeated up and until suitable values for all factors are determined [41]. A full factorial experiment is conducted in this DOE, implying that the number of experiments equals $n_r \cdot 2^{n_f}$ experiments. Where n_r is the number of replications and n_f is the number of varied factors. The lower and upper limits of the factors are based on manual experiments and user experience with the HGA and the system under investigation. Moreover, the optimization problem is restricted in this DOE. For this purpose, the structural update module is omitted. For convenience, the already presented restricted problem in the update decision variable algorithm section is used. Therefore, the fitness function presented by Equation 5.4 is selected as the response during the tuning procedure. Moreover, two layout configurations are tuned to select suitable parameter values that accelerate the performance of the update algorithm.

Table 5.1: Factors and levels of the DOE framework.

Factor		**Symbol**	**Low level (-)**	**High level (+)**
Population size		N_p	10	40
Crossover rate		p_c	0.50	1.00
Mutation rate		p_m	0.01	0.20
Elitism rate		p_e	0.05	0.20
Max. no improvement local search		max_{LS}	1	10

Only five HGA parameters are inspected during the DOE analysis. The tournament rate is equalized to a fixed value to reduce the number of parameter combinations. This value is chosen based on exploration simulations. As mentioned before, two levels are used. Using more levels provides more insights into the response of the system. However, more experiments are required to obtain these insights. Running a single experiment is costly. Therefore, the number of experiments is limited. Hence during the first DOE iteration, there are in total $2^5 = 32$ different parameter set combinations and thus 96 individual experiments are conducted (i.e. every parameter set is evaluated three times). To reduce the total computation time, the parallel toolbox in Matlab is used. This toolbox provides the possibility to run multiple simulations at a time on digital servers. The values of the levels used in the DOE scheme are presented in Table 5.1.

Figure 5.4 presents the effect of each factor on the response. This plot illustrates the effect of each level of every factor. The effect is equal to the difference of the fitness value whenever a high and low level are applied. The effect is labeled as significant whenever the effect exceeds a certain threshold value. This value is called the Lenth's pseudo standard error (PSE). The PSE value applies methods to estimate the standard error of effects estimations. These methods assume that all effects have the same variance and are based on the concept of sparse effects, i.e. only a few effects are statistically significant [42]. Moreover, Figure 5.5 presents a Pareto chart of the first DOE iteration. The Pareto chart includes the effects of each factor on the response and the PSE value. A factor is considered statistically significant whenever an effect passes this threshold. The plots illustrate the results for both layout configurations. With these plots, it is possible to identify the first significant parameter and choose an appropriate value. The Pareto chart indicates that the population size is significant for both layout configurations. Additionally, Figure 5.4 indicates that a higher population size value results in a lower fitness value. Therefore, the population size is equalized to its upper limit, i.e. $N_p = 40$. Moreover, the remaining factors are labeled as non-significant. Therefore, only the population size is equalized to a constant value. The remaining parameters are studied further in the next iterations.

One significant factor is identified after the first DOE iteration. Hereafter, the interactions between the remaining factors and the chosen value for the population size are analyzed. The same experiment is conducted where four factors are variable, implying that $2^4 = 16$ different parameter sets combinations are identified. The same plots are created and the same analysis is conducted as during the first DOE iteration. These plots are presented in section C.2. With these results, the mutation rate is identified as a significant factor and is equalized to its lower

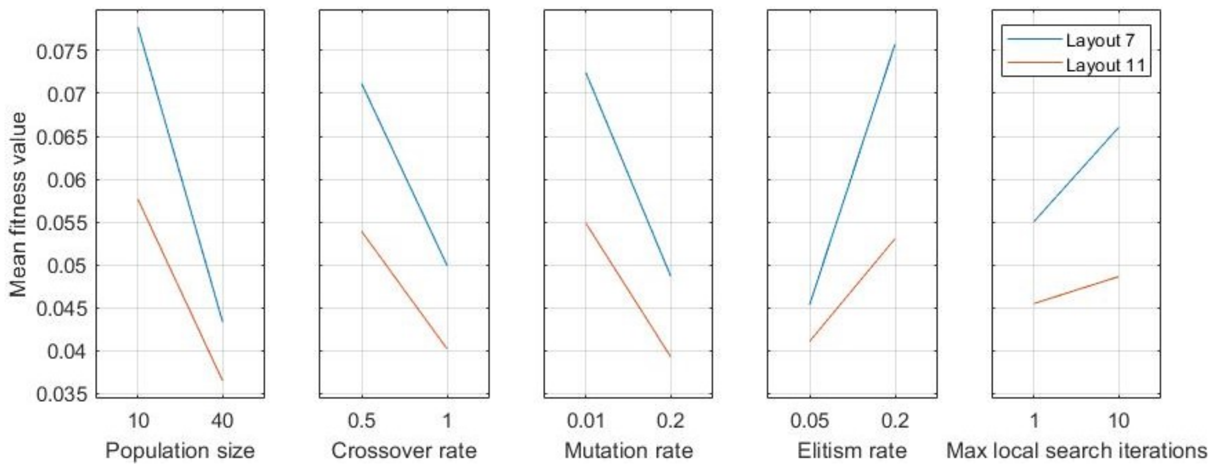


Figure 5.4: Effect of the factors on the response of the system (DOE iteration 1).

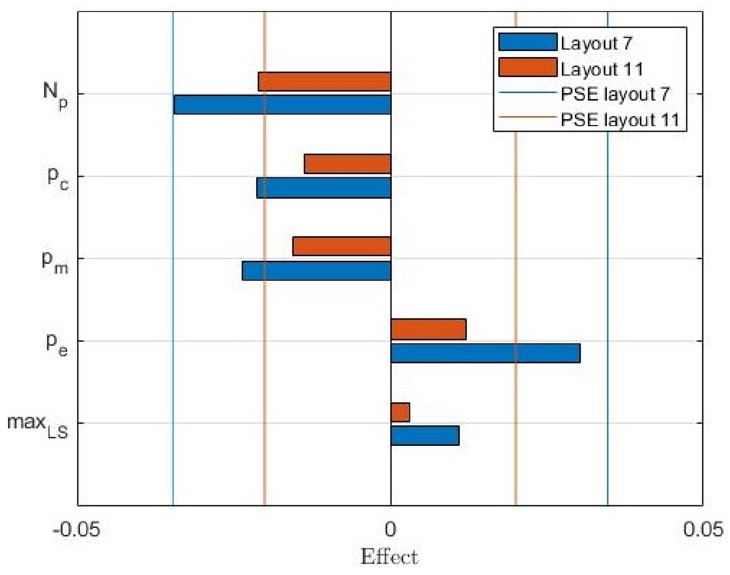


Figure 5.5: Pareto chart plotting the effects of the factors on the response (DOE iteration 1).

limit, i.e. $p_m = 0.10$. Hereafter, the interactions between the remaining factors and the chosen values for the population size and the mutation rate are analyzed, implying that during the third DOE iteration $2^3 = 8$ different parameter sets combinations are identified. Again, the same plots illustrating the main effects on the response and the Pareto chart are presented in section C.2. With these results, the remaining parameter values are determined. The results indicate that the crossover rate could be significant but in both directions (lower and upper limit). Therefore, the crossover rate is chosen in the middle, i.e. $p_c = 0.50$. The remaining parameters are labeled as non-significant and these values are chosen sensibly. The value of the elitism rate is chosen in the middle of the possibilities, i.e. $p_e = 0.10$. Lastly, increasing the maximum number of no improvements of the local search value results in more evaluations. However, decreasing this number could result in a waste of the potential of the local search operator. Therefore, this value is also chosen in the middle, i.e. $max_{LS} = 5$. The tuned parameter values are summarized in Table 5.2.

Table 5.2: Tuned HGA parameters.

Factor	**Symbol**	**Tuned value**
Population size	N_p	40
Crossover rate	p_c	0.50
Mutation rate	p_m	0.10
Elitism rate	p_e	0.10
Max. no improvement local search	max_{LS}	5

5.3 Performance of HGA characteristics

This section analyzes the characteristics of the HGA concerning its performance. Therefore, the HGA is compared to the GA to embody the added value of the local search operator. Additionally, the penalty multiplication factor is analyzed and an appropriate value is selected.

5.3.1 Influence local search operator

This subsection is involved with the local search operator of the HGA. In the report is claimed that this operator has added value in determining solutions with a lower fitness value. This subsection presents an analysis that shows if this operator has added value and outperforms the regular GA. An experiment is conducted to demonstrate this. This experiment is again a simplified version of the final optimization framework. Herein, the structural update part is omitted. For this purpose, the same problem is considered as during the other analysis in this chapter. Additionally, this problem is solved for two layout configurations.

The results of this experiment are summarized by Figure 5.6. This plot presents a bar chart with the mean fitness values for the HGA and the GA. Moreover, the experiment is conducted six times to obtain more confident results. The height of the bar depicts the mean fitness value and the error bars portray the 99.7% significance level. The results between the HGA and the GA for layout 7 are different as can be observed. Hence, the local search operator decreased the mean fitness value. Additionally, a student t-test is conducted to show if the results are statistically different. The p-value for layout 7 is equal to 0.09, implying that the probability is rather low and doubts the validity of the null hypothesis. To be specific, the results are seemingly significantly different and demonstrate the usefulness of the local search operator. The mean fitness value of the HGA and the GA is equal to 0.0307 and 0.0377, respectively, implying that the HGA reduces the fitness value by 19%. The results between the HGA and the GA for layout 11 are less different. The mean fitness value obtained by the HGA is a fraction lower compared to the mean fitness value obtained by the GA. Again, a student t-test is conducted and results in a p-value of 0.84, implying that the null hypothesis is valid with a high probability and that the results are not statistically different. The mean fitness value of the HGA and the GA is equal to 0.0338 and 0.0343, respectively. This implies that the HGA reduces the fitness value by 1%. Therefore, the conclusion can be drawn that the local search operator decreases the fitness value for some layouts. In general, the mean fitness value is not increased due to the application of local search and, therefore, an added value in the current framework.

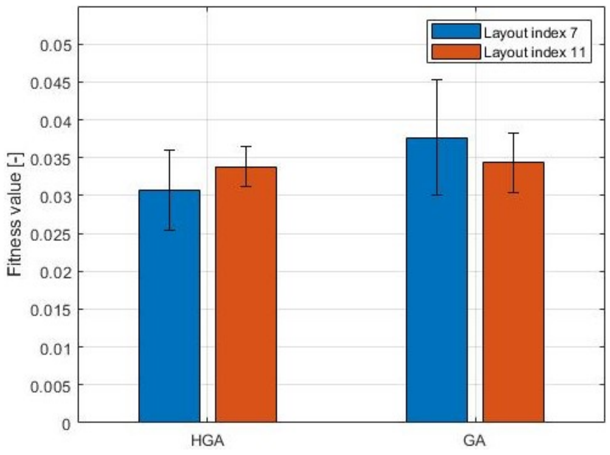


Figure 5.6: Compare fitness value HGA and GA of two configurations.

5.3.2 Influence penalty value

This subsection is involved with the penalty factor used in the first fitness function. This factor influences how severe a violation of a relaxable constraint is punished. As mentioned before, the question may arise what a proper value for this factor would be. Therefore, the effect is analyzed with an experiment. This experiment utilizes a simplified version of the optimization framework where the structural update module is omitted. For this purpose, the same simplified problem is considered as during the other analyses in this chapter. Additionally, this experiment is conducted for two different layout configurations. During the experiment, the penalty factor is varied by the following set of factors $P_{\text{possibilities}} = [0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 1,\ 2]$.

The results of this experiment for layout 7 are presented below. Figure 5.7 presents the fitness value against the number of evaluations, Figure 5.8 presents the difference between the attained and desired gap against the number of evaluations and Figure 5.9 presents the normalized fitness value against the number of evaluations. The normalized fitness value is an expression that demonstrates the effect of the absolute penalty value and is given by Equation 5.5. This expression normalizes the penalty that is incorporated into the fitness value. This expression is obtained by dividing the penalty value by the penalty factor. Hereafter, it is possible to determine which penalty factor has resulted in the lowest absolute penalty value.

$$f_{1,\text{ normalized}}(\vec{x}) = |d_{\text{attained}} - d_{\text{desired}}| + \frac{P(\vec{x})}{P_{\text{factor}}} \tag{5.5}$$

With Figure 5.8 is determined which penalty factor results in a solution where the requirement is met. As can be concluded, solutions that have not attained the requirement are punished more severe due to the high penalty factor, implying that the framework prefers solutions that do attain the requirement and therefore have a lower penalty value. Consequently, a penalty factor that results in attaining the requirement is preferred. This is accomplished whenever the penalty factor is not higher than 0.5. For obvious reasons, the fitness value of a solution applying a lower penalty factor is lower than for solutions applying a higher penalty factor. A naive conclusion is that solutions that apply a lower penalty factor are favourable compared to solutions that apply a higher penalty factor. Therefore, the normalized fitness plot is created and offers the ability to compare the different experiments to each other. The results in this plot contradict the conclusion based on the fitness plot. Herein, a solution that applies the lowest possible penalty factor finds a solution with the highest amount of absolute penalty value. This plot claims that solutions with a higher penalty factor find solutions with a lower absolute penalty value. As mentioned before, the penalty factor is not allowed to be higher than 0.5 otherwise, the requirement is not met. Therefore, an appropriate value for the penalty factor for the current layout configuration is 0.3. Furthermore, the same experiment is conducted for a different layout configuration. These plots are presented in section C.3. Herein, the same conclusion is drawn that solutions that utilize a penalty factor of higher than 0.5 produce solutions that do not attain the requirement. Moreover, the conclusion is drawn that an appropriate value for the penalty factor for the other layout configuration is 0.2. Therefore, the mean value between these results is considered as the final penalty factor, i.e. $P_{\text{factor}} = 0.25$.

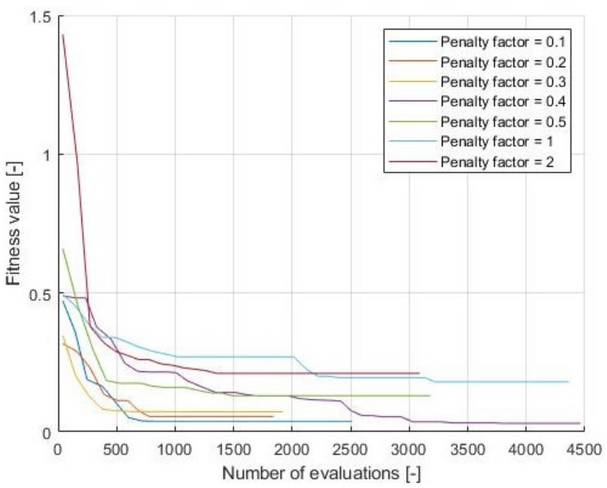


Figure 5.7: Fitness value vs. the number of evaluations for different penalty multiplication factors (layout index 7).

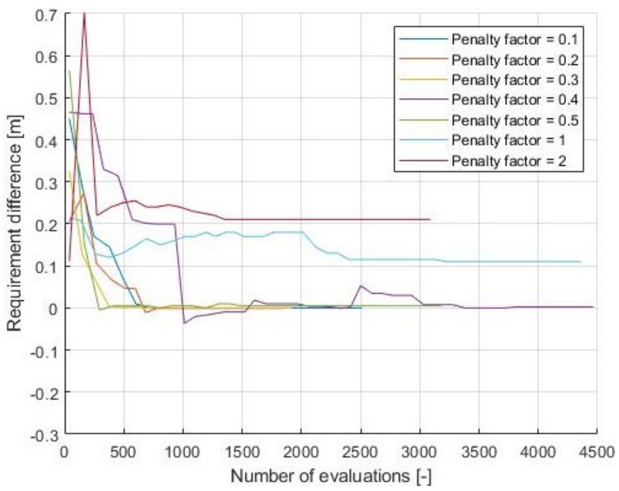


Figure 5.8: Requirement difference vs. the number of evaluations for difference penalty multiplication factors (layout index 7).

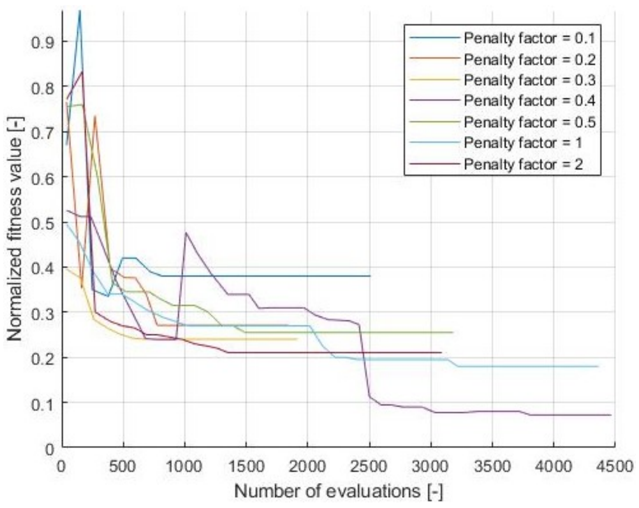


Figure 5.9: Normalized fitness value vs. the number of evaluations for different penalty multiplication factors (layout index 7).

5.4 Case study

The optimization framework is defined to find satisfactory solutions via an automated procedure that meets the imposed requirements, does not violate the unrelaxable constraints and violates the relaxable constraints as little as possible. Therefore, this section presents two use cases to demonstrate the capabilities. Herein, the first use case is involved in the problem, as discussed before, in this report, i.e. creating a gap of a specified distance between consecutive carriers. Additionally, the second use case is involved in the train builder functionality.

The fitness function's parameters related to the cost of the system's configuration are discussed (Equation 5.2) before these use cases are considered. This fitness function has two parameters that are not yet debated, i.e. the cost per conveyor (c_{conveyor}) and the cost related to the footprint ($c_{\text{footprint}}$). The value of the cost per conveyor depends on the number of used conveyors. However, only twin belt sections are used during the current application, implying that the value of the cost per conveyor is a scalar and only depicts the cost of a twin belt section. It is a cumbersome task to find an appropriate value for this variable. Moreover, this value depends on multiple variables of which the cost associated with each variable could not be identified at the current moment in time. Therefore, the decision is made to equalize this value to 1 and to only demonstrate the possibilities of the optimization framework. The same reasoning is considered for the cost value related to the footprint. In general, it is assumed that shorter system configurations are preferred over longer system configurations. Therefore, the total length of a system configuration is multiplied by a cost value. Once again, it is cumbersome to determine an appropriate value for this term. Therefore, this value is equalized to 1 to demonstrate the possibilities of the optimization framework. A follow-up study shall be conducted to find appropriate values for these parameters.

5.4.1 Create gap use case

The first use case considers the problem that has already been discussed in this project, i.e. creating a specified gap between consecutive carriers at the last conveyor. Herein, three fundamental actions take place, i.e. stop in dieback, start-up and speed change action. Carriers arrive with a begin velocity and leave with an end velocity. Herein, the begin velocity is higher than the end velocity. These values are specified by the requirements. A general overview of the (not yet optimized) layout is illustrated in Figure 5.10. To recapitulate, the purple line, the red line and the turquoise line indicate the locations of the decision point, dieback point and speed change point, respectively. This example uses three conveyors to solve the problem. With this optimization framework is studied if this design can be further improved. Therefore, the design of Vanderlande is compared to the configuration found via the optimization framework.

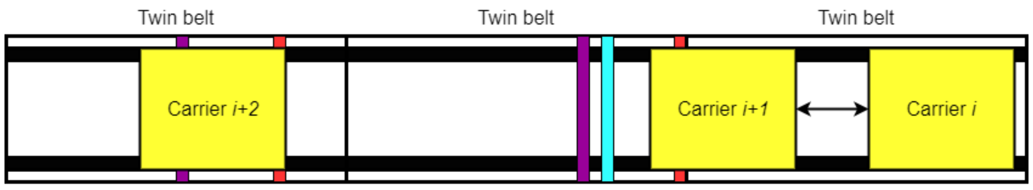


Figure 5.10: General overview of the create gap use case.

Optimization constraints

This use case utilizes all imposed constraints as defined in subsection 5.1.2. Herein, the physical control, constrained velocity delta, collision and simulation error constraints are unrelaxable. Only the slip constraint is relaxable. Some slip occurs during simulations of the configuration defined by Vanderlande. However, this is not the case during the analysis of the design mode (the deterministic system). The difference in behaviour is linked to the fact that during the simulation the plant and controller have different cycle times, resulting in latency. Therefore, this constraint is labeled as relaxable and its goal is to minimize slip.

Decision variables

Some additional measures are considered to limit the number of decision variable possibilities even more. These variables can be further bounded depending on the applied use case, implying that the decision variable limits and velocity settings are examined for each use case. Bounding the conveyors where speed change is allowed to be applied limits the number of possibilities drastically. Speed change cannot be applied at the first or last conveyor of the system during this use case. A carrier always goes in dieback on the first conveyor to ensure that occupancy of the system is maximal. Therefore, applying speed change on that conveyor is unnecessary. Moreover, applying speed change on the last conveyor never results in the completion of the functionality. The system is still performing its task and the gap cannot correctly be mapped. Therefore, defining speed change on the last conveyor is undesired.

Moreover, the limits of the decision variables are also further bounded depending on the use case. The slip constraint is a relaxable constraint that should be minimized as much as possible. Therefore, the position of the dieback point for a conveyor that does not apply speed change can be further confined. In general, a carrier on this conveyor goes in dieback. Therefore, the dieback point is placed more inwards to ensure that the carrier is on speed at the handover, implying that the lower bound of the decision variable is adjusted. This position depends on the geometry of the carrier and the conveyor; and the velocity profile. As mentioned before, the upper bound of this decision variable depends on the physical control constraint. These bounds are illustrated in Figure 5.11. Herein, the invalid locations for the dieback point are colored grey, implying that conveyors that cannot define a dieback point given the imposed constraints are omitted, i.e. a conveyor could be too short. Furthermore, the dieback position of the conveyor where speed change is applied is of less importance. In general, a carrier goes not in dieback on this conveyor. Moreover, the position of the speed change point is bounded via the same approach. The upper bound depends on the physical control constraint. Furthermore, the lower bound depends on the geometry of the carrier and the conveyor; and the velocity profile. Therefore, this position is chosen such that no slip occurs during the handover.

Additionally, the limit of the speed change reset factor is bounded as well. This variable determines when a conveyor is allowed to reset its velocity value to its begin velocity value again. The lower bound can be restricted due to the slip constraint. This bound depends on the geometry of the carrier and the conveyor. The velocity value is only allowed to reset whenever the carrier and the upstream conveyor are no longer in contact with each other (i.e. its contact surface depends on the geometry of the carrier and the conveyor). This principle is illustrated by Figure 5.12. This figure depicts the side view of two conveyors and one carrier. Moreover, distance x_1 and x_2 depend on the geometry of the carrier and the conveyor (these distances are the contact-free areas of the objects), respectively. This implies that the velocity values of both conveyors shall be equal to each other whenever the carrier is in contact with both conveyors. Whenever this is not the case, the conveyor where the carrier is physically located can change its velocity value without generating slip.

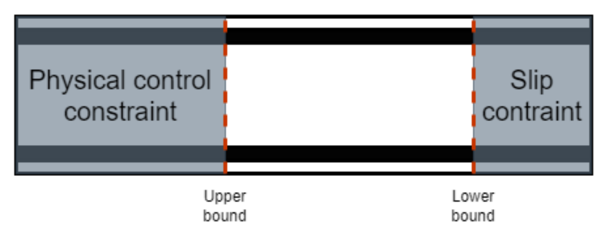


Figure 5.11: Lower and upper bounds of the dieback position for the current use case.

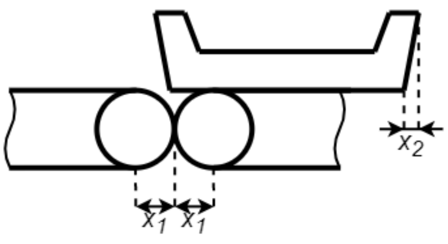


Figure 5.12: Contact surface depending on the geometry of the carrier and the conveyor.

Optimization settings

The optimization framework requires some initialization before it can be solved. The HGA parameters are tuned in section 5.2 and are used in this optimization framework. Additionally, the minimum and maximum numbers of conveyors that are considered are initially equal to three. Hence, this use case is solved by Vanderlande with three conveyors. Therefore, at least three conveyors are required. The number of conveyors is increased if the framework is not able to find a satisfactory solution with this number of conveyors. This use case is similar to the problem used in all analyses during this chapter and implies that the fitness function related to the requirement of the system is equal to Equation 5.4. Herein, the function related to the requirements and the capacity of a given configuration is associated with the requirement difference. Moreover, the term associated with the capacity is omitted, due to the capacity is fixed (i.e. the capacity is not required to be minimized) because the distance between every carrier has to be the same. Moreover, the simulator used in the optimization framework is run via the performance mode. This is done to create a representative view of the system. This approach represents the behaviour of the actual system better compared to a simulation run under the design mode. During the sensitivity analysis is concluded that at least 10 carriers shall be mapped to determine a confident representation of the simulation's output. The final gap value is determined as the average value between all obtained gaps between the carriers. Additionally, the simulation could cost too much time in some cases. Therefore, a maximum simulation time is specified. If the system is not able to process the specified number of carriers before the end of the simulation time, the simulation is terminated. As a result, the fitness value is equalized to infinity and the solution is infeasible.

Optimization results

After the framework is initialized, the optimization problem is solved. It is possible to execute the initialize system module with the help of the imposed settings. Herein, all possible layouts are defined and ranked based on their costs. Hereafter, the cheapest layout is chosen, the decision variable limits are defined and an initial value for each decision variable is selected. Hereafter, the remainder of the optimization framework is executed.

The results of this study are summarized in Table 5.3. This table presents the top 10 layout possibilities based on their costs. Moreover, each experiment is conducted three times to obtain more confident results. Furthermore, these solutions are classed into three categories. In each category, the last two conveyors have the same length. Only the length of the first conveyor is different. To be specific, the length of the second and third conveyors of alternatives 1, 2, 4 and 10 are the same. However, the length of the first conveyor is different. Alternative 1 has the shortest length, alternative 2 has the second shortest length, etc.. This implies, that only one solution has to be found per category. The only thing that has to be ensured is that it is feasible that a carrier can go in dieback and start-up on the first conveyor. This feasibility depends on the imposed constraints as mentioned in the decision variables paragraph of this use case. As can be observed, the cheapest possibility of each category cannot define a feasible solution due to the length of the first conveyor. Only the alternatives thereafter are able to define such systems. Therefore, the fitness value concerning the requirement of the system of the same category is the same (if the solution is feasible). Alternative 2 can define a system with the best possible results as can be observed in this table. The system meets the requirement and violates the slip constraint minimally. Additionally, alternative 5 also meets the requirement (with some tolerance) but violates the slip requirement more compared to alternative 2. This is the result of the increased length of the second conveyor, implying that the system is not able to produce less slip due to this larger length. The same conclusion is drawn concerning alternative 8. Herein, the length of the second section is increased even more. Moreover, alternative 2 is chosen as the final solution. This solution meets the requirement and the relaxable constraints are violated the least. Furthermore, this solution is preferred due to its cost, i.e. less footprint. The conveyor lengths of each alternative, the motion profile settings and the decision variable values of each experiment of the final solution can be found in Appendix A.

The search results of alternatives 2, 4 and 10 by the optimization framework are presented by Figure 5.13, Figure 5.14 and Figure 5.15. These figures present the fitness value, requirement difference value and penalty value, respectively. Herein, the described metric is plotted against

Table 5.3: Create gap use case optimization results.

Alternative		$\mathbf{f_1}$	Requirement difference	Penalty value	$\mathbf{f_2}$
		[-]	[m]	[-]	[Cost]
1		NaN	NaN	NaN	7.8
2		0.017	0.000	0.017	8.2
3		NaN	NaN	NaN	8.2
4		0.017	0.000	0.017	8.6
5		0.049	0.011	0.038	8.6
6		NaN	NaN	NaN	8.6
7		0.049	0.011	0.038	9.0
8		0.089	0.021	0.068	9.0
9		0.089	0.021	0.068	9.4
10		0.017	0.000	0.017	9.8

the number of evaluations. These figures present the results of three conducted experiments. The experiment is replicated due to the stochasticity of the simulation. Every run could potentially result in a different output. Hence, the average output is taken as the final result. The values displayed in Table 5.3 are averaged values of the last evaluation. As can be observed, the framework is able to find a solution that is in the neighbourhood of the final solution rather quickly. Hereafter, the penalty value is reduced by the remaining evaluations. The same plots of the other two categories are presented in subsection C.4.1. With the help of these experiments, the conclusion is drawn that increasing the length of the second conveyor results in worse performance. Systems with a longer second conveyor have more trouble finding a solution where the requirement is met and where the penalty value is minimized. Keep in mind that multiple feasible solutions can be defined for every configuration. Multiple combinations of parameter sets could result in feasible and satisfactory solutions. Therefore, it is recommended to run the optimization framework multiple times and select the best option among them.

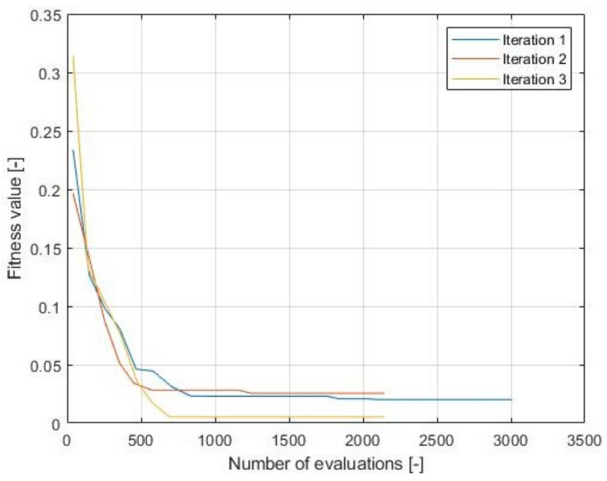


Figure 5.13: Fitness value vs. number of evaluations (alternatives 2, 4 and 10).

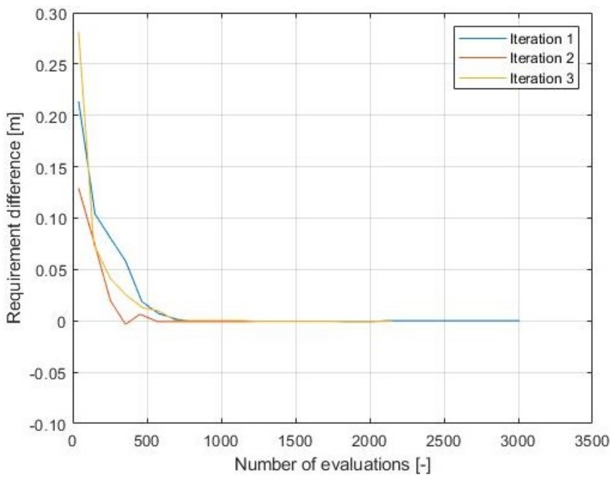


Figure 5.14: Requirement difference vs. number of evaluations (alternatives 2, 4 and 10).

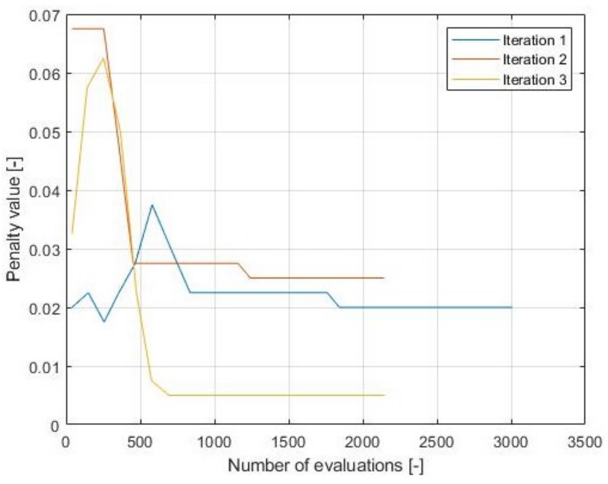


Figure 5.15: Penalty value vs. number of evaluations (alternatives 2, 4 and 10).

Use case comparison

The found solution is compared to Vanderlande's solution. The length of the second and third conveyors of both systems are the same. Only the length of the first conveyor differs and is chosen larger by Vanderlande due to the identification of the carrier has to be completed before other actions are allowed to take place. However, this is not a problem when comparing the results. In reality, the configuration of alternative 4 corresponds to Vanderlande's configuration, implying that only the position of the dieback points, speed change point and downstream release gap differs. Vanderlande's design has an average requirement difference of 0.010 meters, whereas the optimization framework's design has an average requirement difference of 0 meters. Moreover, the variance of both systems is the same. The dissimilarity is the result of the design process. The system designed by Vanderlande is based on a deterministic system. Herein, the plant and controller are not uncoupled and the reaction time is equalized to a constant value. In reality, this is not the case as explained in section 3.2. The reaction time depends on the sample moments of the plant and the controller. Moreover, the average penalty value of the manually created design is equal to 0.025, whereas the average penalty value of the solution found via the optimization framework is equal to 0.014. This example illustrates the usefulness of not only the optimization framework but also the performance mode. Whenever both are combined, solutions are found that outperform manually created designs. Herein, not only the requirement difference is smaller, but also solutions are found with lower penalty values.

Use case improvement

The controller sample time plays a crucial role during the design of an use case. A system is designed based on a specified design sample time. This sample time is used to determine the position of the decision point, the speed change point and the value of the downstream release gap (a lower sample time value results in less reaction time). However, the controller sample time is not always the same at the customer's site, this value lies within a certain range. Therefore, the robustness of this use case is studied. For this purpose, the optimized solution is run for two controller sample times, i.e. $T_c = T_d$ and $T_c = 1/4 \cdot T_d$. Herein, T_c is the controller sample time and T_d is the design sample time. These results are presented in Figure 5.16. This figure presents bar charts of the gap between carriers at the last conveyor of both experiments. As can be observed, the mean gap value of the system applying the controller sample time equal to the design sample time results in the desired gap. However, the mean gap value increases if the controller sample time is decreased. As mentioned before, three actions take place that influences the system's response, i.e. stop in dieback, start-up and speed change. The system behaves as desired whenever the controller sample time is equal to the sample time used to design the system. Herein, all actions start executing as intended. However, all actions start too early whenever the controller sample time is decreased, implying that the carriers go in

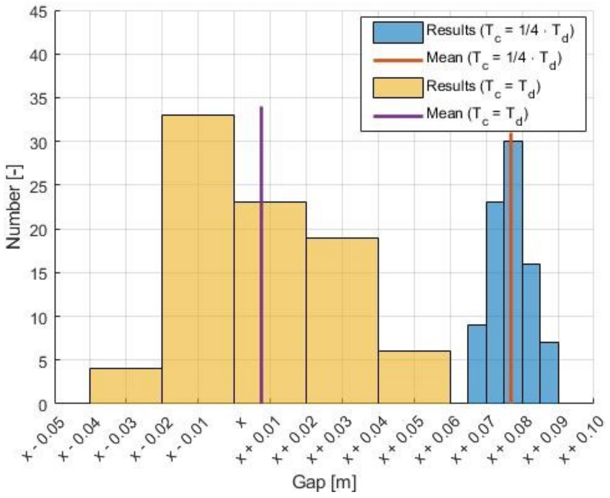


Figure 5.16: Simulation results for the create gap use case (no quick features applied).

dieback in front of the intended dieback point, that the carriers are released too early during the start-up and that the carriers perform the speed change action too early. The accumulation of the influences of these actions results in the increased gap value. Moreover, it is observed that the variance of the measured data is larger for the system that utilizes the design sample time as controller sample time compared to the system that utilizes the reduced controller sample time. This is linked to the reaction time. As mentioned before, the reaction time is influenced by the controller sample time. Moreover, a system that utilizes a larger controller sample time has a larger range concerning the reaction time. The influence of the controller sample time on the reaction time is presented in subsection 3.2.3.

An improvement of the system is introduced to make the system more robust for varying controller sample times by reducing the dependency on the controller. Therefore, quick speed change is introduced. The quick speed change functionality circumvents the centralized controller and initiates speed change whenever a carrier triggers a sensor. Consequently, only the dieback and the start-up action are governed by the controller. A general overview of the improved system is illustrated in Figure 5.17. A solution is found with the optimization framework. Herein, the same system and HGA settings are applied as discussed at the beginning of the subsection, only the speed change action is decoupled from the controller. The system parameters of this solution are presented in Appendix A and the intermediate results in subsection C.4.2.

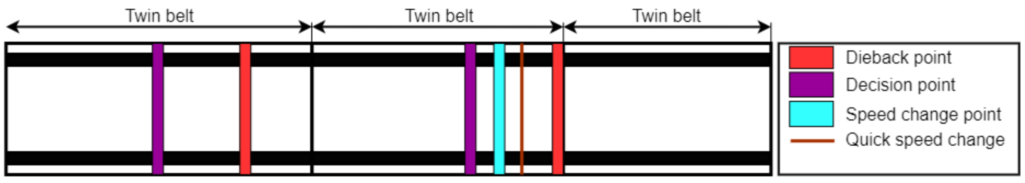


Figure 5.17: General overview of the first proposed improvement (add quick speed change).

The results of this experiment are presented in Figure 5.18. This figure presents bar charts of the gap between the carriers for both controller sample times. The system operates correctly if it is run for the designed cycle time. Moreover, the results of the system run with the decreased cycle time are better compared to the same results of the system where no quick speed change is applied. Decoupling the initiation of the speed change action from the centralized controller is beneficial. Moreover, the observed gap difference is linked to the dieback and start-up action. Furthermore, the variance is reduced as well, implying that the results have a smaller range.

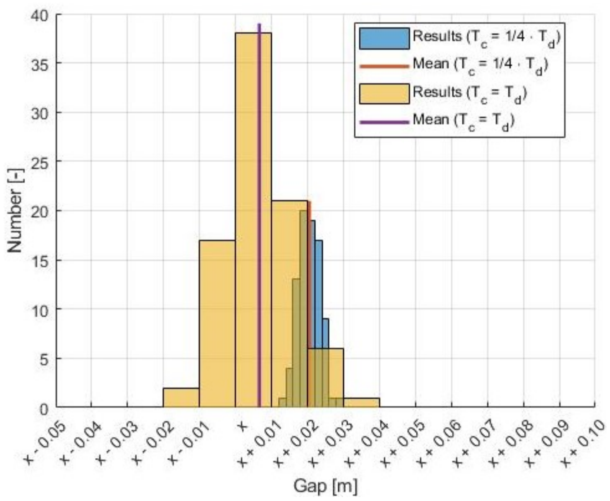


Figure 5.18: Simulation results for the first proposed improvement (apply quick speed change).

A second improvement is introduced to study if the system can be more resilient for varying controller sample times. This improvement contains, besides quick speed change, also quick stop at the first conveyor, implying that only the start-up is governed by the centralized controller. A general overview of this system is presented by Figure 5.19. The system parameters of this solution are presented in Appendix A and the intermediate results in subsection C.4.2.

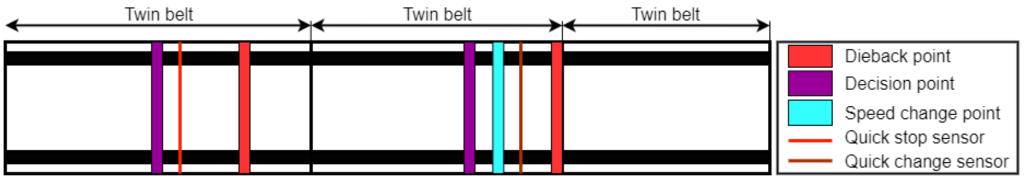


Figure 5.19: General overview of the second proposed improvement (apply quick stop/change).

The results of this experiment are presented in Figure 5.20. This figure presents bar charts of the gap between the carriers for both controller sample times. Again, the system operates correctly if it is run for the designed cycle time. Moreover, the variance of these results is smaller due to only the start-up action is governed by the centralized controller. However, the gap value becomes smaller than the desired gap value whenever the controller sample time is decreased. This reduction is governed by the start-up action. In this system, only the start-up action is regulated by the centralized controller, implying that the dieback and speed change actions are executed on time. However, the start-up action is applied too early, due to the actual reaction time being decreased. Therefore, the gap between the carrier is always smaller than intended.

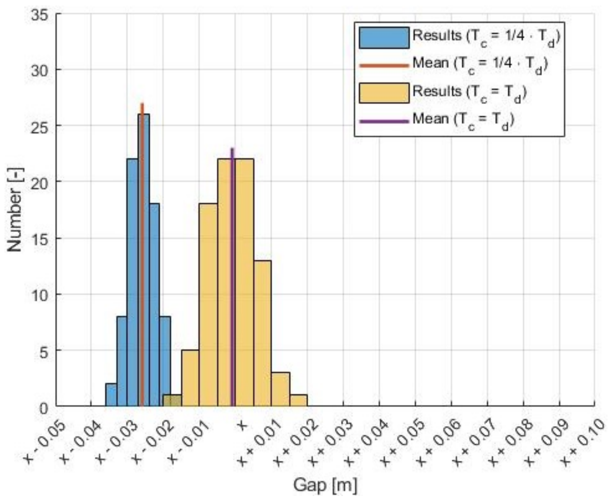


Figure 5.20: Simulation results for the second proposed improvement (apply quick stop/change).

The results of the improvement experiments are summarized in Table 5.4. In practice, the controller sample time varies. Therefore, a design has to be robust for these varying controller sample times. During experiments is concluded that the results of the original system vary significantly whenever different controller sample times are applied. Attaining the desired gap between consecutive carriers is a strict requirement and a gap difference of 7.7 centimeters is attained whenever the system is run with the reduced controller sample time. This gap difference is too much. Therefore, two improvements are suggested, i.e. apply quick speed change and apply quick stop and speed change. Whenever only quick speed change is applied, the average gap difference is 2.1 centimeters when the reduced controller sample time is used. Moreover, the results for both sample times have a smaller range compared to the results of the original system. Whenever quick stop and speed change are applied, the average gap difference is equal to -2.6 centimeters when the reduced controller sample time is used. Moreover, these results have a smaller range compared to the results of the original system and the system that only applies quick speed change. Concluding which design is better, depends on the requirements and preferences of the system. Whenever the mean gap value is not allowed to be smaller than the desired gap value, the system that only applies quick speed change is preferred. The average requirement difference will be in the range of 0.7 and 2.1 centimeters. However, the results have a larger variance and range. Whenever the gap requirement is softer, the system that applies quick stop and quick speed change is preferable. The average requirement difference will be in the range of -0.1 and -2.6 centimeters. Moreover, the range and variance of the results are smaller compared to the other systems, due to only the start-up action being governed by the centralized controller. Hence, this conclusion depends on the characteristics of the problem.

Table 5.4: Statistical summary of the three discussed systems.

	Controller cycle time [s]	**Mean** [m]	**Standard deviation** [m]	**Range** [m]
No quick features	$1/4 \cdot T_d$	$(x + 0.077) \pm 0.002$	0.006	0.022
	T_d	$(x + 0.008) \pm 0.005$	0.020	0.075
Only quick change	$1/4 \cdot T_d$	$(x + 0.021) \pm 0.001$	0.003	0.016
	T_d	$(x + 0.007) \pm 0.003$	0.010	0.044
Quick stop and change	$1/4 \cdot T_d$	$(x - 0.026) \pm 0.001$	0.003	0.016
	T_d	$(x - 0.001) \pm 0.002$	0.007	0.035

5.4.2 Train builder use case

The second use case considers a problem that Vanderlande would like to be improved. This functionality creates trains of carriers with a specified distance between the carriers in the same train and is released whenever a full train is created. Herein, the same fundamental actions take place as in the first use case, i.e. stop in dieback, start-up and speed change. Again, carriers arrive with a begin velocity and leave with an end velocity. Herein, the begin velocity is higher compared to the end velocity. Moreover, the acceleration value of the conveyor that creates the train is different compared to the acceleration values of the other conveyors. A lower acceleration value is applied because trains are created for stacks of carriers. These stacks are heavier than normal and for safety reasons, a lower acceleration value is utilized. These values are specified by the requirements. A general overview of the (not yet optimized) layout is illustrated by Figure 5.21. This example uses three conveyors to solve the problem. Herein, the last conveyor is responsible for creating trains of two carriers. Whenever this train is created, the train is released and carrier *j+2* becomes the first carrier of the new train. The goal of this use case is to create a configuration that satisfies the requirement that releases trains with a specific distance between the carriers in that train, does not violate the unrelaxable constraint and violates the relaxable constraints as little as possible. Moreover, the capacity of this functionality should be as high as possible. In theory, multiple configurations can create and release trains. However, which solution satisfies the requirements, does not violate the constraints and has the highest possible throughput?

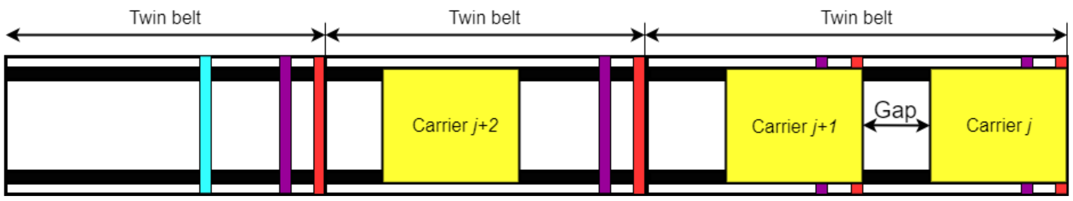


Figure 5.21: General overview of the train builder use case.

Optimization constraints

This use case does utilize all defined constraints as defined in subsection 5.1.2. Herein, the physical control, constrained velocity delta, collision and simulation error constraints are unrelaxable. Only the slip constraint is relaxable. A solution shall be defined where slip is minimized. In this use case, only slip at the handover at the last conveyor is considered (when the train leaves the system). At this conveyor, the gap between the carriers is already created and shall not be changed due to any losses. This is not important at the upstream conveyors, because the carriers are not yet aligned and any potential slip is counteracted later in the system.

Decision variables

Some additional measures are considered to limit the number of decision variables even more. As mentioned before, these variables can be further bounded depending on the applied use case. Therefore, the decision variable limits and velocity settings shall be examined for each use case. Again, bounding the conveyors where speed change is applied reduces the number of possibilities drastically. Speed change cannot be applied at the last conveyor of the configuration of this use case. This conveyor has the designated task to build the trains of carriers. Moreover, lower velocity values produce smaller position errors. Therefore, the speed change action shall take place upstream of this conveyor. Moreover, applying speed change on the first conveyor is not convenient. To ensure the highest possible throughput, speed change shall be applied as late as possible. Therefore, the decision is made to allow only speed change on the second conveyor of the configuration.

Additionally, the limits of the decision variables can be further bounded depending on the use case. The dieback coordinates of the train builder conveyor are fixed and depend on the length of the train, the length of the carriers and the gap value between the carriers in a train. Moreover, the remaining dieback points are fixed at the end of the conveyor. The slip constraint is here not of importance (only at the end of the system). Therefore, the most convenient dieback points are chosen. Furthermore, this is also required to ensure that the train builder functionality operates as designed (i.e. automatically creates the desired gap between carriers). As mentioned before, speed change shall not be applied too early. Applying this too early results in a lower throughput value. Moreover, the position of the speed change point is bounded due to the physical control constraint (as discussed in the create gap use case section). Moreover, the train builder release gap of the conveyor upstream of the train builder conveyor requires some special attention. The lower bound shall be specified such that the complete train has left the train builder conveyor before an upstream carrier goes in dieback on this conveyor.

Optimization settings

The optimization framework requires some initialization steps before the use case can be solved. The HGA parameters are tuned in section 5.2 and are used in this optimization framework. Additionally, the minimum and maximum numbers of conveyors that are considered are initially equal to three. First is considered if solving this problem is possible with this amount of conveyors. If this is not the case, the number of used conveyors is increased. Moreover, the carrier length and the gap between the carriers are also specified to solve this use case. This information can be found in Appendix A.

This use case is different to the problem used in all analyses in this chapter. During this problem, it is of importance to find a solution that satisfies the requirements, does not violate the unrelaxable constraints, violates the relaxable constraints as little as possible and finds a solution with the highest capacity. The fitness function related to the requirement of the system, therefore, includes a term concerning the requirement difference, the capacity and the penalty. For that reason, this fitness function is defined by Equation 5.6. Where μ is the system's capacity in seconds per carrier. This fitness function includes the capacity term which enables the possibility to find a configuration that has the highest possible capacity. The reader could imagine that a lot of different configurations are possible that satisfies the requirements and do not violate the constraints, but what configuration has also the highest possible capacity?

$$f_1(\vec{x}) = K(\vec{x}) + P(\vec{x}) = |d_{\text{attained}} - d_{\text{desired}}| + \mu + P(\vec{x}) \tag{5.6}$$

Moreover, the simulator used in the optimization framework is run via the performance mode. This is done to create a representative view of the system. This approach represents the behaviour of the actual system better compared to a simulation run under the design mode. During the sensitivity analysis is concluded that at least 10 carriers shall be mapped to determine a confident representation of the simulation's output. The requirement and capacity are therefore determined as the average value between the results between all processed carriers. Additionally, the simulation will cost too much time in some cases. Therefore, a maximum simulation time is specified. If the system is not able to process the specified number of carriers before the end of the simulation time, the simulation is terminated. As a result, the fitness value is equalized to infinity and the solution is infeasible.

Optimization results

After the framework is initialized, the optimization problem is solved. It is possible to execute the initialize system module with the help of the imposed settings. Herein, all possible layouts are defined and ranked based on their costs. Hereafter, the cheapest layout is chosen, the decision variable limits are defined and an initial value for each decision variable is selected. Thenceforth, the remainder of the optimization framework is executed.

The results of this study are summarized in Table 5.5. This table presents the top 10 layout possibilities based on their costs. Moreover, each experiment is conducted three times to obtain more confident results. As can be concluded, almost every alternative is able to produce results that meet the requirement, only alternatives 5, 6 and 7 are not able to meet the requirement. These solutions cannot produce satisfactory results due to an unsatisfactory decision variable set. Some adjustments to the decision variable limits shall be made to solve this hurdle. However, it is not necessary to do so. The framework is able to find satisfactory solutions that are cheaper compared to the unsatisfactory solutions. The satisfactory solutions have approximately

Table 5.5: Train builder use case optimization results.

Alternative		$\mathbf{f_1}$	**Capacity**	**Requirement difference**	**Penalty value**	$\mathbf{f_2}$
		[-]	[cph]	[m]	[-]	[Cost]
1		3.670	986	0.005	0	15.0
2		3.645	991	0.011	0	15.4
3		3.651	991	0.016	0	15.4
4		3.629	998	0.003	0	15.8
5		3.662	1011	0.096	0	15.8
6		3.549	1047	0.111	0	15.8
7		3.663	1036	0.186	0	16.2
8		3.656	1007	0.063	0	16.2
9		3.736	967	0.012	0	16.6
10		3.700	976	0.001	0	17.0

the same capacity value, this value varies a bit due to the application of the performance mode and the value of the requirement difference. Moreover, an increase in the requirement difference increases the capacity value. This observation is motivated due to an increasing requirement difference means that the gap between the carriers in a train decreases, which obviously results in a higher capacity value. Moreover, all solutions have a penalty value of zero, implying that no slip occurs at the handover at the last conveyor. Alternative 1 is chosen as the final solution due to the low requirement difference and the lowest configuration cost. The conveyor lengths of each alternative, the motion profile settings and the decision variable values of each experiment of the final solution can be found in Appendix A.

The search results of alternative 1 by the optimization framework are presented in the figures below. Figure 5.22 presents the fitness value and Figure 5.23 presents the requirement difference against the number of evaluations for all experiments for the train builder use case. As mentioned before, the experiments are conducted three times due to the stochasticity of the simulation to obtain more confident results. Every run could potentially result in a different output. Therefore, the average output is taken as the final result. The values displayed in Table 5.5 are averaged values of the last evaluation. The penalty plot is not displayed due to the penalty value is always zero.

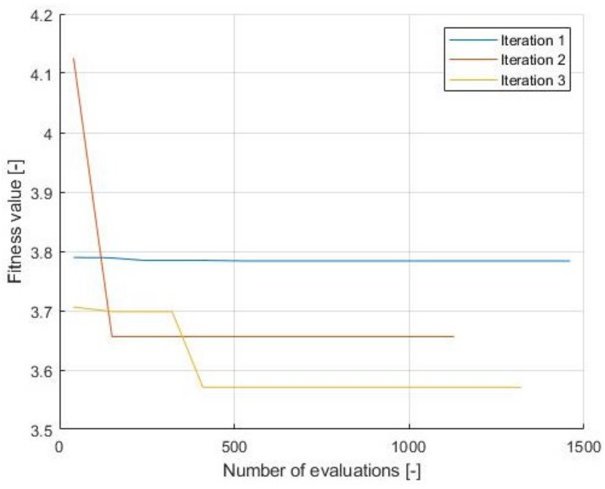


Figure 5.22: Fitness value vs. number of evaluations of the train builder use case (alternative 1).

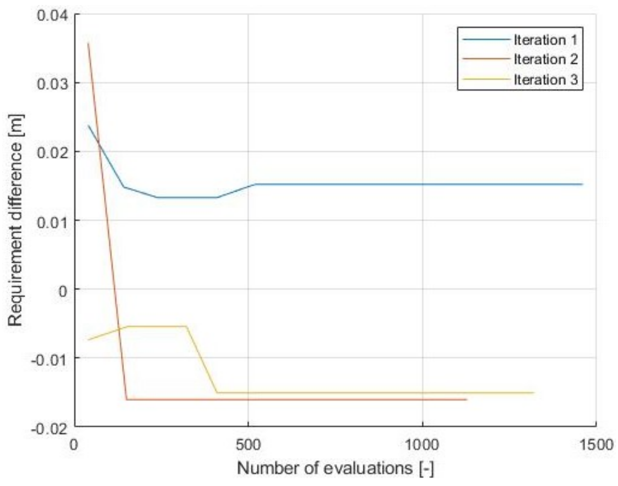


Figure 5.23: Requirement difference vs. number of evaluations of the train builder use case (alternative 1).

Use case comparison

The design regarding the train builder functionality created by Vanderlande had some flaws and was not designed properly for carriers of a specific length. Therefore, the train builder use case is redesigned via the optimization framework. The new design is similar to the designs used for carriers of different lengths. Herein, the second conveyor is used to properly transit the acceleration and deceleration values without conflicting actions. Moreover, speed change is applied on the first conveyor to ensure a feasible flow of carriers regarding the acceleration and deceleration values. The system found via the optimization framework equals in a solution that performs the requirements and does not violate any constraints. Some tuning concerning the downstream release gap could be conducted to even further increase the capacity value. However, a satisfying solution is suggested by the optimization framework.

Use case improvement

Again, this system is designed for a specific controller sample time. Therefore, the robustness of this use case is studied as well. Again, the optimized solution is run for two controller sample times, i.e. $T_c = T_d$ and $T_c = 1/4 \cdot T_d$. The results are presented in Figure 5.24. This figure presents bar charts of the gap between carriers in the leaving train of both experiments. As can be observed, the mean gap value of the system applying the controller sample time equal to the design sample time results in the desired gap. Furthermore, the mean gap value decreases slightly if the controller sample time is decreased. As mentioned before, three actions take place that influences the response of this system, i.e. stop in dieback, start-up and speed change. Herein, speed change does not influence the response of the system. Every carrier goes in dieback before the speed change action is executed, implying that the velocity is equal to the end velocity whenever the conveyor executes the start-up action. Hence, only the dieback and start-up action influence the response of the system. This means that whenever the reduced controller sample time is applied, the carrier goes in dieback too early, and executes the start-up action too early as well. This approximately counteracts both effects and minimizes the gap difference compared to the system that applies the designed controller sample time. Moreover, due to the reduction of the controller sample time, the variance and range are reduced. These results are summarized in Table 5.6.

This use case produces results for varying controller sample times that are acceptable. The results are acceptable due to the average gap value is only decreased by 1.9 centimeters compared to the designed system. Moreover, this difference is not a big deal for this functionality due to where this functionality is applied in the system. Therefore, no improvements are proposed for the train builder use case.

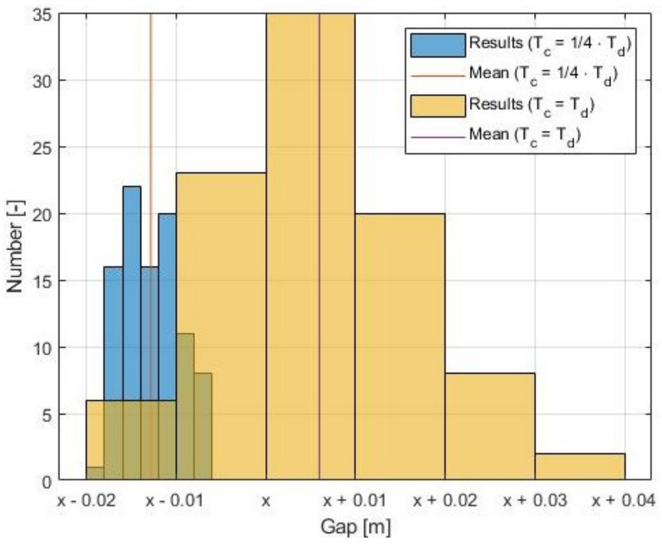


Figure 5.24: Simulation results for the train builder use case (no improvements).

Table 5.6: Statistical summary of the not improved train builder use case.

	Controller cycle time [s]		**Mean** [m]	**Standard deviation** [m]	**Range** [m]
No improvements	$1/4 \cdot T_d$		$(x - 0.013) \pm 0.001$	0.003	0.012
	T_d		$(x + 0.006) \pm 0.003$	0.010	0.046

Chapter 6
Conclusion and recommendations

6.1 Conclusion

The objective of this study aims to develop a numerical tool that improves the design procedure of the Tubtrax baggage handling solutions. Currently, the design procedure is conducted manually, implying that the entire solution definition is conducted by hand, i.e. all calculations, feasibility checks and requirement validation are conducted by hand. The current design process has some disadvantages which are mainly the result of the manual iterative design process. As a result, this design process is time-consuming; prone to human calculation errors and validation errors; and not flexible. In order to find a solution for the manual iterative design procedure problem, the first research question is formulated as:

> *How can model based design improve the design procedure of a task of the Tubtrax baggage handling system compared to the current manual design process?*

This research question investigates the possibility to replace the manual iterative design process with an alternative design process. For this purpose, the model based design methodology is proposed to circumvent the difficulties induced by the traditional design procedure. Model based design is a mathematically based method for designing complex control systems by creating a mathematical representation of the problem. Applying the model based design methodology has the following advantages, i.e. it is flexible, efficient, accurate and modular. In practice, different modeling techniques can be applied to define a mathematical representation of the problem. Choosing the correct modeling technique and program depends on the characteristics and the purpose of the problem and the system. The report concluded that the goal of the simulation framework is to design a satisfactory solution where some level of detail shall be incorporated. Moreover, the conclusion is drawn that the response of the system shall be simulated via the numerical model. Therefore, Matlab is used to define and simulate Tubtrax baggage handling solutions. Herein, modular designs can be created and simulated with the help of handle classes.

After a suitable modeling technique and software tool are selected, the Tubtrax simulation framework can be defined. This framework utilizes an uncoupled plant and controller. The plant includes all conveyors and carriers in the system. Moreover, the controller determines the velocity profile of the conveyors depending on the properties of the carriers and conveyors. Furthermore, the response of the carriers is governed by the properties of the conveyors. This framework provides the possibility to quickly define configurations and set all system parameters. Furthermore, the simulation framework provides the ability to simulate the created system, implying that all calculations are no longer performed by hand. This saves time and prevents that human calculation errors are made. Moreover, the simulation framework provides the ability to analyze the results via different post-process features. These features help to make decisions on how the system shall be updated or if the system satisfies the requirements.

Moreover, an additional disadvantage of the manual design procedure is that there is no confirmation if the by hand defined solution is optimal. Therefore, a feature that finds a satisfactory set of design parameters for the design solution is missing. Hence, the definition of the simulation framework provides the possibility to define such an optimization framework. An optimization framework can be applied to find automatically feasible solutions that are restricted by constraints and should meet imposed requirements. Using simulation results for the optimization framework implies that the optimization framework is designed via simulation based optimization principles. In the literature, numerous optimization algorithms are available. Therefore, the second research question is formulated as:

> *Which optimization algorithms are most suitable to solve a simulation based optimization problem and how do they compare to each other (i.e. the number of evaluations and performance)?*

Simulation based optimization algorithms can be classed into two main subsets, i.e. problems with discrete or continuous decision variables. This project considers only problems with discrete decision variables. Moreover, a distinction can be made between small or large solution spaces. Within this project, the solution space is considerably large. Therefore, metaheuristics, stochastic adaptive search and random search techniques are identified as suitable optimization algorithms to solve the current simulation based optimization problem. Random search or Monte Carlo simulation is applied as a benchmark. If an alternative technique performs worse, then the usability is questionable. Moreover, a HGA is applied as a metaheuristic. The HGA applies all GA operators and applies local search on the elite chromosomes of the previous generation. Furthermore, SA is applied as a stochastic adaptive search technique. Hereafter, the performance of these algorithms is compared to each other and is concluded that the HGA is identified as the most promising algorithm and is used to update the decision variables. The HGA produces consistent and quicker satisfactory results and finds fast solutions that lie in the neighbourhood of the final solution. Moreover, the advantage of the HGA is that the algorithm explores the entire solution space and that due to the elitism and local search operator the most promising solutions are maintained and further explored. Additionally, the HGA can escape a local optimum fairly easily.

After a suitable optimization algorithm is identified to update the decision variables, the question may arise of how the optimization framework shall be constructed to find satisfactory solutions. Therefore, the third research question is formulated as follows:

> *How can model based design aid in the search for a satisfactory set of design parameters by optimizing the capacity for an arbitrary task that is subjected to imposed requirements and constraints; and how should the optimization framework be constructed?*

The model based design methodology offers the possibility to construct an optimization framework to solve an optimization problem to find a set of design parameters that meet the imposed requirements and do not violate the imposed constraints. For this purpose, the simulation framework is utilized as an evaluation mechanism of defined solutions. Herein, two fitness functions are defined that shall guide solving the optimization problem. The first fitness function considers the requirement, capacity and constraints of the system. The second fitness function considers the cost of the configurations. In this optimization framework, a resourceful procedure is defined that starts searching for solutions for the most potential suitable configuration and whenever no satisfactory solution is found, it proceeds to the next promising configuration. The ranking of the configurations is based on the cost of the configurations. The created optimization framework is hereafter applied to two use cases to demonstrate its usability. With the help of the framework, two functionalities of the Tubtrax baggage handling system are improved. Moreover, with the help of the simulation framework, it is possible to study the robustness of a found solution. With the help of these results, a found solution is analyzed and can be improved.

6.2 Recommendations

This study has focused on the design process of configurations that fulfill specified functionalities and how this design process can be improved. Moreover, the defined simulation framework is extended with an optimization framework to find satisfactory solutions via an automated procedure. During the optimization study is noticed that the simulation speed is a bottleneck.

The speed of the simulation framework is reduced due to how the simulation data is stored. This data is stored via structure arrays. The advantage of these arrays is that these arrays improve the readability of the script. Readability is required to ensure that everyone can understand the scripts and is able to extract required data easily. Moreover, these arrays can be easily extended whenever additional functionalities are implemented. The application of matrices can be considered to improve the simulation speed. However, the disadvantage of using matrices is that the readability is reduced, implying that extracting data and adding new functionalities becomes harder. Implying that a better understanding of the simulation framework is required. Therefore, a trade-off shall be made between simulation speed and how the script is organized. Moreover, this decision is influenced by the application purpose. If Vanderlande only desires the simulation framework to define new concepts and to add new functionalities via a modular procedure, then the decreased simulation speed is not a problem. However, if they would like to find suitable configurations via simulation based optimization techniques, the simulation framework shall be constructed differently to accelerate the simulation speed.

Moreover, the defined simulation framework is a proof of concept, implying that only some fundamental conveyor types and functionalities are implemented. The definition of the simulation framework mainly focused on the flow of carriers that are only influenced by dieback points, speed change points and the interaction between carriers. However, in practice, additional factors are of importance to construct feasible configurations. In reality, the positions of objects and sensors are considered. For example, size checks of a loaded carrier. These checks have to be executed and finished before the decision point is reached by the carrier. This affects the conveyor's length and the position of the dieback point, implying that these objects influence the configuration of the functionalities. Therefore, it is recommended to extend the simulation framework with these objects in order to facilitate a more complete solution.

Thereafter, the fitness function related to the cost of the system shall be further investigated. During this project, it is assumed that the cost per conveyor and the cost associated with the footprint are equalized to 1 to demonstrate the possibilities of the optimization framework. Finding appropriate values for these parameters is a cumbersome task, due to the identification of the variables that govern this cost value at the current moment in time is not possible.

The tuning procedure of the HGA could be conducted in a more advanced manner. During this DOE only a first-order model is used to draw conclusions, implying that direct interrelationships between factors are not studied directly. Moreover, a full factorial design is used without using center points. Center points could be added to study if the response has curvature and to conclude if a point between the low and high levels is more suitable as an HGA parameter value. A more advanced tuning procedure could be executed to study if the obtained HGA parameters are chosen correctly or should be adjusted. During this project, the decision is made to omit these additions to reduce the total simulation time. However, a more advanced tuning procedure can be conducted if a more efficient simulation model is created.

The developed optimization framework is only able to find satisfactory solutions for systems designed for a specific controller sample time. However, this sample time varies in practice. The influence of the controller sample time on the system's response is demonstrated in the use case improvement section of the create gap use case problem. Therefore, the optimization framework could be extended with the functionality to find satisfactory solutions for systems with varying controller sample times. This addition could be an added value to define robust solutions for complex systems where multiple actions are executed (i.e. stop in dieback, start-up and speed change).

Bibliography

[1] Vanderlande Industries B.V., “Company profile,” 2022. [Online]. Available: https://www.vanderlande.com/about-vanderlande/company-profile/

[2] ——, “History,” 2022. [Online]. Available: https://www.vanderlande.com/about-vanderlande/history/

[3] ——, “Warehousing,” 2022. [Online]. Available: https://www.vanderlande.com/warehousing/

[4] ——, “Parcel,” 2022. [Online]. Available: https://www.vanderlande.com/parcel/

[5] ——, “Airports,” 2022. [Online]. Available: https://www.vanderlande.com/airports/

[6] ——, “Vanderlande,” 2022. [Online]. Available: https://www.vanderlande.com/

[7] D. Du, P. Pardalos, and W. Wu, “History of optimization,” *Encyclopedia of Optimization*, 2008.

[8] A. Gosavi, *Simulation based optimization.* Springer New York, NY, 2015.

[9] P. F. Smith, S. M. Prabhu, and J. Friedman, “Best practices for establishing a model-based design culture,” *SAE Technical Papers*, no. March, 2007.

[10] M. Ahmadian, Z. Nazari, N. Nakhaee, and K. Zoran, “Model based Design and SDR,” *2nd IEE/EURASIP Conference on DSPenabledRadio*, 2005.

[11] M. Yaryna, “Why Model-Based Design Makes Traditional Automotive Software Development Obsolete,” 2020. [Online]. Available: https://medium.com/the-research-nest/why-model-based-design-makes-traditional-automotive-software-development-obsolete-b3abb2cbe794

[12] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems.* Springer Science+Business Media, LLC, 2008.

[13] G. Wainer, “Modeling and simulation concepts,” in *Discrete-Event Modeling and Simulation*, 2009, pp. 3–31.

[14] I. Grigorvey, *AnyLogic in Three Days.* AnyLogic, 2015.

[15] M. Harchol-Balter, “Networks of Queues and Jackson Product Form,” in *Performance Modeling and Design of Computer Systems : Queueing Theory in Action.* Cambridge : Cambridge University Press, 2013.

[16] D. van Beek and D. Hendriks, *Model-Based Systems Engineering Lecture Notes 4TC00.* Eindhoven University of Technology, 2014.

[17] K. Hoad, S. Robinson, and R. Davies, “Automated selection of the number of replications for a discrete-event simulation,” *Journal of the Operational Research Society*, vol. 61, pp. 1632–1644, 2010.

[18] W. Hachicha, A. Ammeri, F. Masmoudi, and H. Chachoub, “A comprehensive literature classification of simulation optimisation methods,” *MPRA Paper 27652*, no. June 2014, 2010.

[19] Y. Carson and A. Maria, “Simulation optimization: Methods and applications,” *Winter Simulation Conference Proceedings*, pp. 118–126, 1997.

[20] M. C. Fu, “Optimization via simulation : A review,” *Annals of Operations Research*, vol. 53, pp. 199–247, 1994.

[21] M. C. Fu, C. C. Price, and J. Zhu, *Handbook of Simulation Optimization.* Springer New York, NY, 2015.

[22] S. L. Digabel and S. M. Wild, “A Taxonomy of Constraints in Simulation-Based Optimization,” Cornell University, Tech. Rep. May, 2015.

[23] O. Yeniay, “Penalty function methods for constrained optimization with genetic algorithms,” *Mathematical and Computational Applications*, vol. 10, no. 1, pp. 45–56, 2005.

[24] H. Jalali and I. V. Nieuwenhuyse, “Simulation optimization in inventory replenishment: A classification,” *IIE Transactions*, vol. 47, no. 11, pp. 1217 – 1235, 2015.

[25] L. J. Hong, W. Fan, and J. Luo, “Review on ranking and selection: A new perspective,” *Frontiers of Engineering Management*, vol. 8, no. 3, pp. 321–343, 2021.

[26] Y. Wardi, C. G. Cassandras, and X. R. Cao, “Perturbation analysis: A framework for data-driven control and optimization of discrete event and hybrid systems,” *Annual Reviews in Control*, vol. 45, pp. 267–280, 2018.

[27] M. Billio, A. Monfort, and C. Robert, “The simulated likelihood ratio method,” *Computational and Statistical Methods for the Analysis of Spatial Data*, 1998.

[28] N. Sad, “Simulation and the Finite-Difference Stochastic Approximation,” *AMO - Advanced Modeling and Optimization*, vol. 14, no. 1, pp. 197–205, 2012.

[29] L. Schruben, “Simulation Optimization Using Frequency Domain Methods,” *Winter Simulation Conference Proceedings*, pp. 366–369, 1986.

[30] S. Jacobson and L. Schruben, “A harmonic analysis approach to simulation sensitivity analysis,” *IIE Transactions*, vol. 31, 1999.

[31] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, “Metaheuristic algorithms: A comprehensive review,” in *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications.* Elsevier Inc., 2018, pp. 185–231.

[32] G. Wood and Z. Zabinsky, “Stochastic Adaptive Search,” in *Handbook of Global Optimization.* Springer, Boston, MA, 2002.

[33] D. Henderson, S. H. Jacobson, and A. W. Johnson, “The Theory and Practice of Simulated Annealing,” in *Handbook of Metaheuristics*, 2006, pp. 287–319.

[34] T. A. El-Mihoub, A. A. Hopgood, L. Nolle, and A. Battersby, “Hybrid Genetic Algorithms : A Review,” *Engineering Letters*, vol. 11, no. August, p. 124–137, 2006.

[35] L. Gharsalli and Y. Guerin, “A hybrid genetic algorithm with local search approach for composite structures optimization,” *8th European conference for aeronautics and space sciences (EUCASS)*, 2019.

[36] A. Lawrynowicz, “Genetic Algorithms for Solving Scheduling Problems in Manufacturing Systems,” *Foundations of Management*, vol. 3, no. 2, pp. 7–26, 2011.

[37] R. C. Purshouse and P. J. Fleming, “Why use Elitism and Sharing in a MultiObjective Genetic Algorithm?” *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 520–527, 2002.

[38] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[39] A. K. Peprah, S. K. Appiah, and S. K. Amponsah, "An Optimal Cooling Schedule Using a Simulated Annealing Based Approach," *Applied Mathematics*, vol. 08, no. 08, pp. 1195–1210, 2017.

[40] A. Gunawan, H. C. Lau, and Lindawati, "Fine-tuning algorithm parameters using the design of experiments approach," *Learning and Intelligent Optimization - 5th International Conference, LION 5*, 2011.

[41] L. Eriksson, E. Johansson, N. Kettaneh-Wold, C. Wikstrom, and S. Wold, *Design of experiments: principles and applications.* Umea: Umetrics, 2008.

[42] R. V. Lenth, "Quick and easy analysis of unreplicated factorials," *Technometrics*, vol. 31, no. 4, pp. 469–473, 1989.

Appendix A
Confidential appendix

This section is confidential and therefore left empty.

Appendix B
Simulation framework validation results

This chapter of the appendix includes additional plots concerning the validation of the Tubtrax simulation framework.

B.1 90 degrees sort section

This section presents bar charts of the result of the experiments conducted to validate the simulation model of the 90 degrees sorting functionality. Figure B.1, Figure B.2 and Figure B.3 present bar charts of the experimental results of the sensor deltas as described in subsection 3.4.3. Moreover, Figure B.4, Figure B.5 and Figure B.6 present bar charts of the simulation results of the sensor deltas as described in subsection 3.4.3.

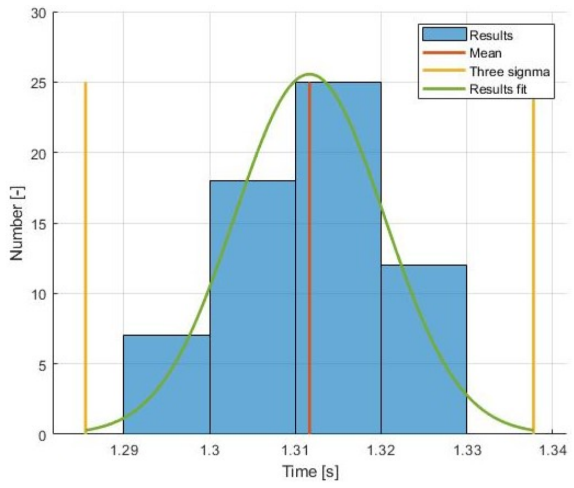


Figure B.1: Experiment results 90 degrees sort functionality (time between triggering sensor 1 and transfer state down sensor).

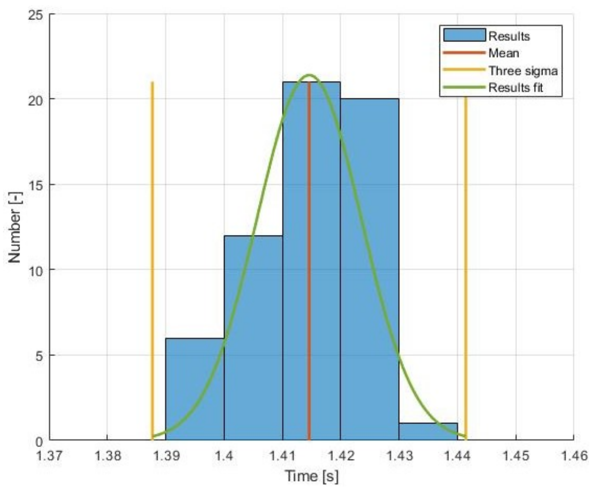


Figure B.2: Experiment results 90 degrees sort functionality (time between triggering sensor 1 and transfer state up sensor).

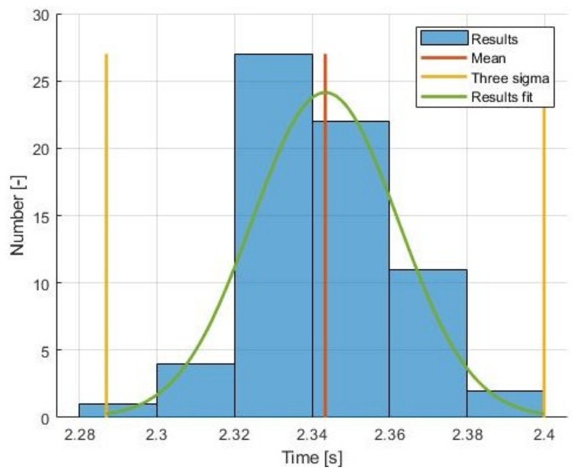


Figure B.3: Experiment results 90 degrees sort functionality (time between triggering sensor 1 and sensor 2).

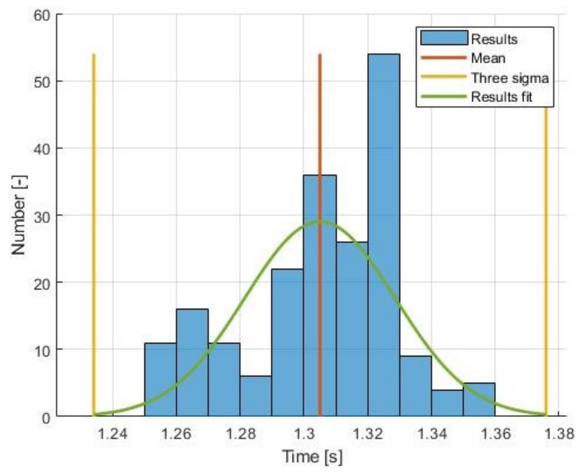


Figure B.4: Simulation results 90 degrees sort functionality (time between triggering sensor 1 and transfer state down sensor).

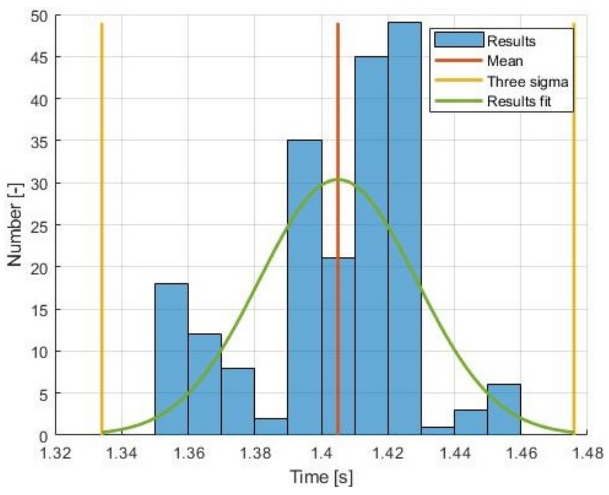


Figure B.5: Simulation results 90 degrees sort functionality (time between triggering sensor 1 and transfer state up sensor).

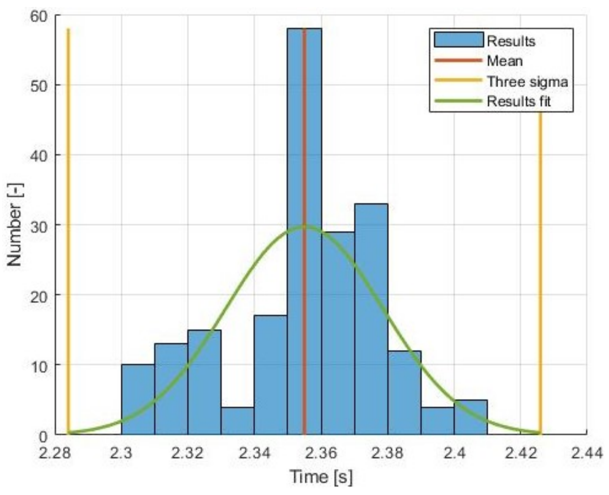


Figure B.6: Simulation results 90 degrees sort functionality (time between triggering sensor 1 and sensor 2).

B.2 90 degrees merge section

This section presents bar charts of the result of the experiments conducted to validate the simulation model of the 90 degrees merging functionality. Figure B.7, Figure B.8 and Figure B.9 present bar charts of the experimental results of the sensor deltas as described in subsection 3.4.4.

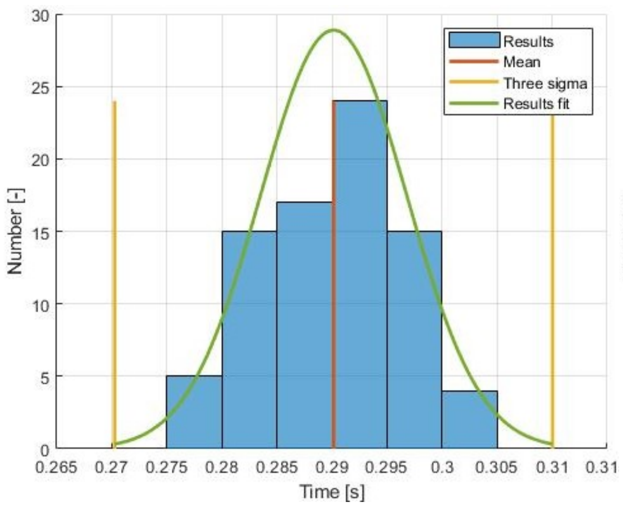


Figure B.7: Experiment results 90 degrees merge functionality (time between triggering sensor 1 and transfer in elevated state).

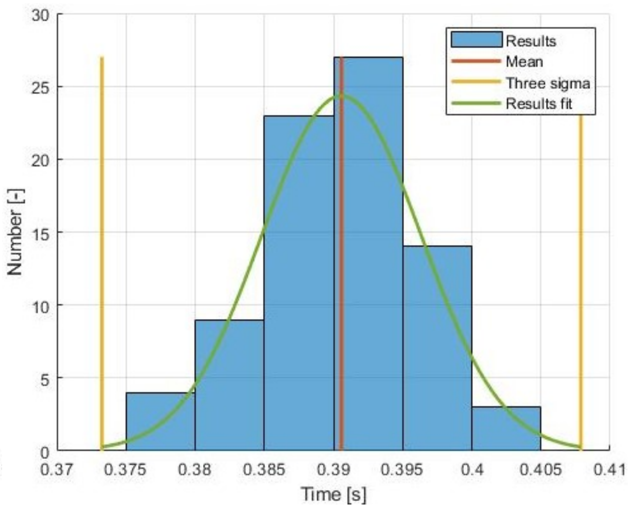


Figure B.8: Experiment results 90 degrees merge functionality (time between triggering sensor 1 and transfer in lowered state).

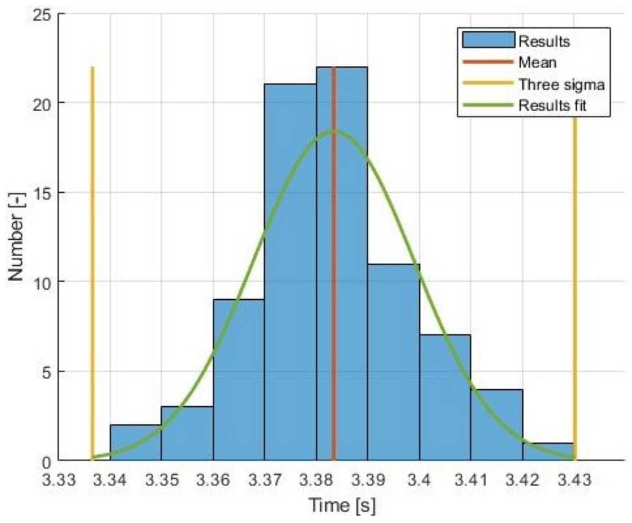


Figure B.9: Experiment results 90 degrees merge functionality (time between triggering sensor 2 and sensor 3).

Moreover, Figure B.10 and Figure B.11 present bar charts of the simulation results of the sensor deltas as described in subsection 3.4.4. No bar chart of sensor delta 3 is presented. This data is not distributed and always equal to the presented value.

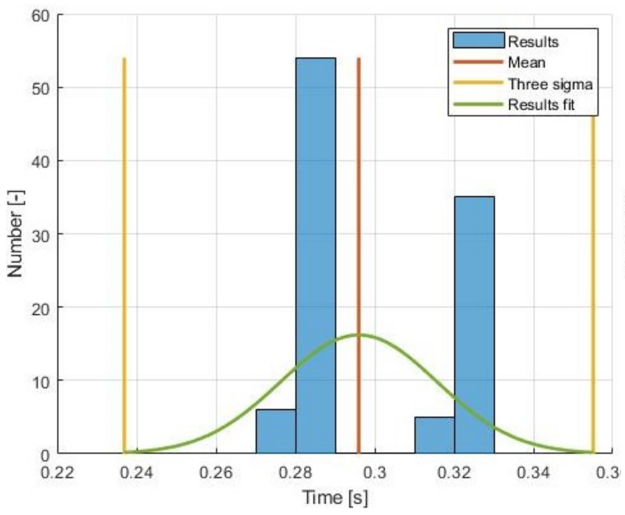


Figure B.10: Simulation results 90 degrees merge functionality (time between triggering sensor 1 and transfer in elevated state).

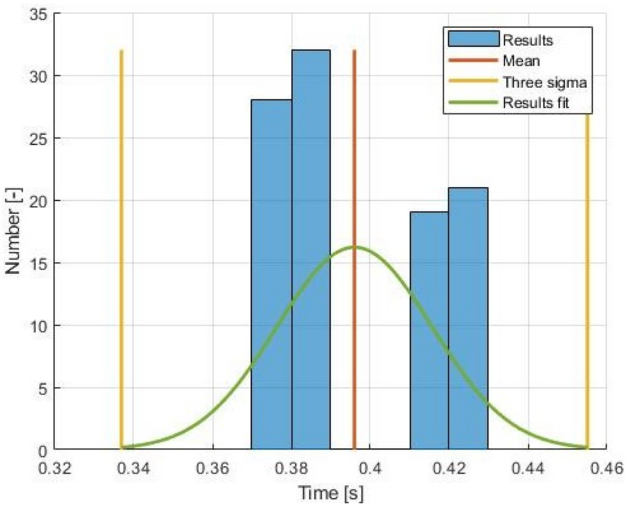


Figure B.11: Simulation results 90 degrees merge functionality (time between triggering sensor 1 and transfer in lowered state).

Appendix C
Optimization framework results

This chapter of the appendix includes additional analysis results to substantiate conclusions that are drawn in chapter 5. This appendix includes results of a second layout configuration regarding the conclusion concerning the decision variable update algorithm; plots and analysis of the second and third DOE iteration; results of a second layout configuration regarding the penalty multiplication factor and additional plots concerning the use cases.

C.1 Additional update decision variable algorithm results

This section presents the experimental results to draw a conclusion regarding the decision variable update algorithm for a different layout configuration. An additional layout is considered to make the conclusion more general and to demonstrate which decision variable update algorithm is most suitable for this optimization framework. The found solution applies to all layout configurations. Figure C.1 presents an illustration where the fitness value of the HGA, SA algorithm and Monte Carlo approach is plotted against the number of evaluations. Moreover, Figure C.2 presents an illustration where the requirement difference of the HGA, SA algorithm and the Monte Carlo simulation approach is plotted against the number of evaluations. In this experiment, the SA algorithm and Monte Carlo approach produce solutions that meet the requirement. However, the HGA finds a satisfactory solution with fewer evaluations. The HGA finds its best solution with a fitness value of 0.042 after just 430 evaluations. The GA finds its best solution with a fitness value of 0.027 only after 4000 evaluations, implying that the SA finds a solution with a slightly lower fitness value. However, the SA finds a solution with a fitness value of 0.32 after 400 evaluations. Hence, the SA requires 10 times more evaluations to achieve this. The Monte Carlo approach has processed 6000 evaluations, which resulted in 240 feasible solutions (i.e. only 4% of the evaluations resulted in a feasible solution) and the best fitness value of 0.043. To conclude, the HGA, SA and Monte Carlo approach finds solutions with an approximate same fitness value. The SA algorithm finds a solution with a slightly lower fitness value compared to the HGA and Monte Carlo approach. However, the HGA finds satisfactory solutions quicker compared to the other approaches. As mentioned before, the time required to find a satisfactory solution is of importance as well. Therefore, the HGA is the preferred decision variable update algorithm for this layout configuration as well.

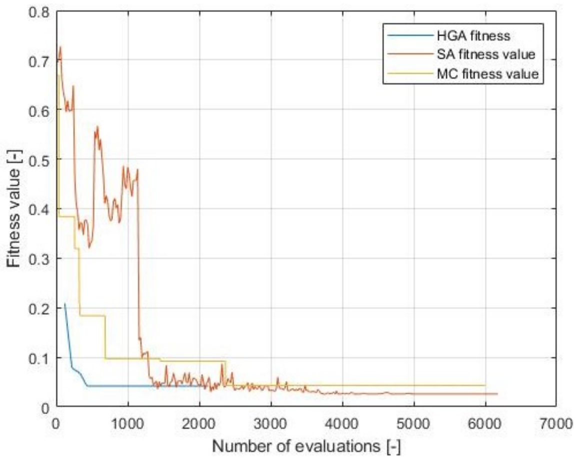


Figure C.1: Fitness value of the HGA, SA algorithm and Monte Carlo plotted against the number of evaluations (layout index 11).

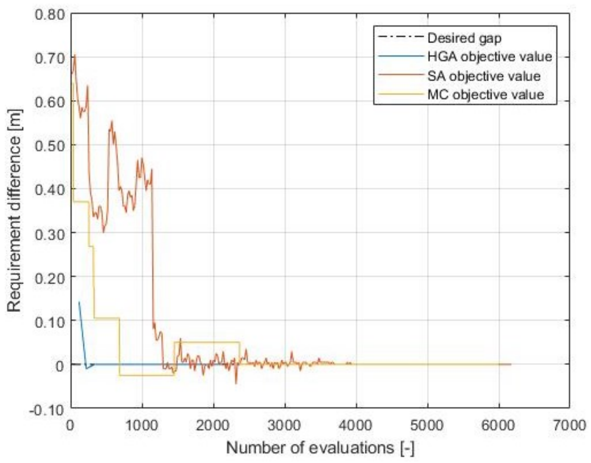


Figure C.2: Objective value of the HGA, SA algorithm and Monte Carlo plotted against the number of evaluations (layout index 11).

C.2 Additional HGA tuning results

This section presents the plots of the remaining DOE iterations, i.e. iterations 2 and 3. The main effect plot and the Pareto chart of the second DOE iteration are presented by Figure C.3 and Figure C.4, respectively. These plots indicate that the mutation rate can be labeled as a significant factor for both layouts. Therefore, the lower limit of the mutation rate is selected as the tuned value. The remaining factors can be labeled as non-significant. Moreover, the crossover rate and the maximum number of local search iterations are in both layouts conflicting. In layout 7, the upper limit would result in a lower fitness value. And in layout 11, the lower limit would result in a lower fitness value. However, these factors are labeled as non-significant and therefore this is not a problem. The mutation rate, elitism rate and the maximum number of local search iterations in combination with the selected values for the population size and the mutation rate are further studied in a third DOE iteration.

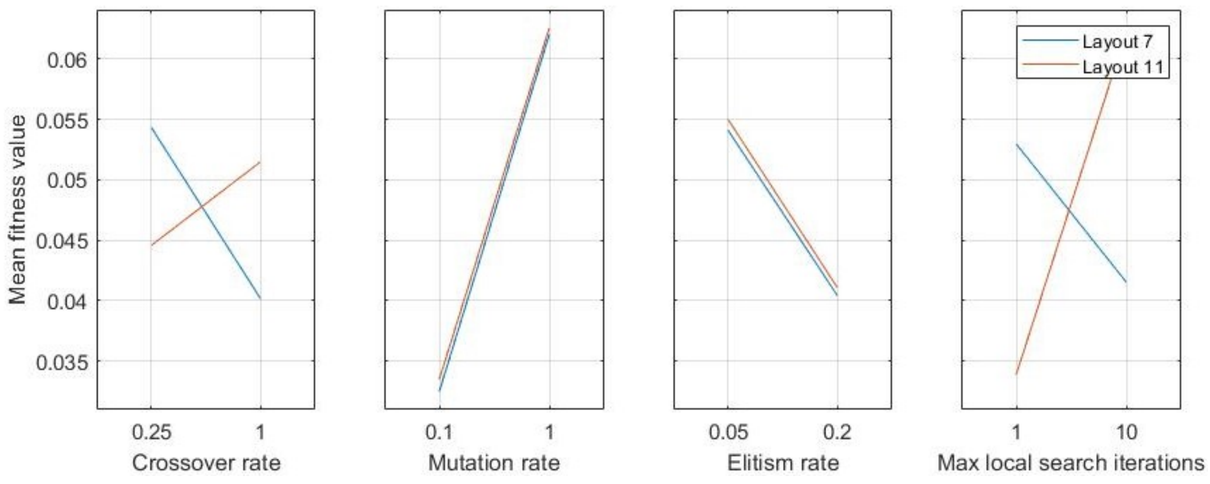


Figure C.3: Main effects of the factors on the response of the system (DOE iteration 2).

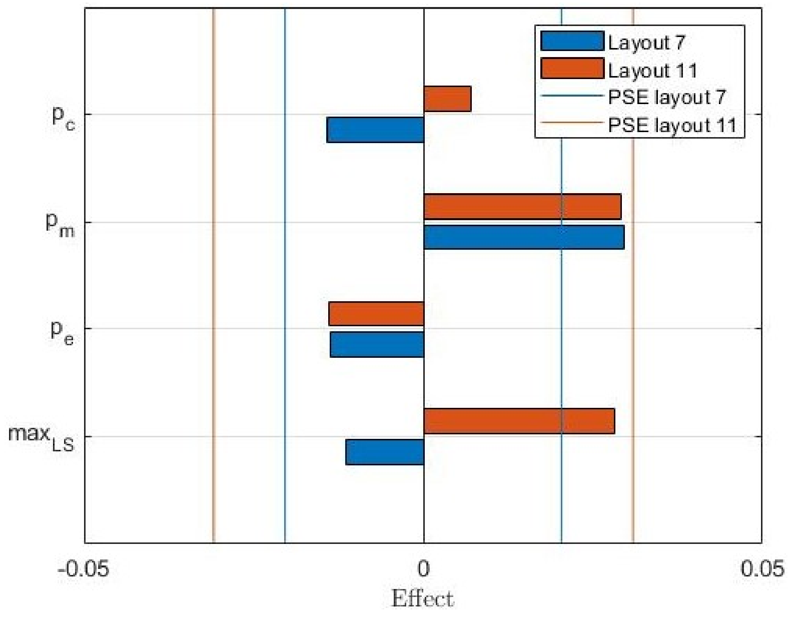


Figure C.4: Pareto chart plotting the effects of the factors on the response (DOE iteration 2).

The main effect plot and the Pareto chart of the third DOE iteration are presented by Figure C.5 and Figure C.6, respectively. These plots indicate that the crossover rate is significant but in both directions (lower and upper limit). Therefore, the crossover rate is chosen to be equal to $p_c = 0.50$. This value lies somewhere in the middle. Additionally, the elitism rate is labeled as non-significant. Therefore, the value is chosen in the middle of the possibilities, i.e. $p_e = 0.10$. Lastly, the maximum number of local search iterations is also labeled as non-significant. Therefore, this value is chosen sensibly. Increasing this number results in more evaluations. However, decreasing this number results in a waste of the potential of the local search operator. Therefore, this value is also chosen in the middle, i.e. $max_{LS} = 5$.

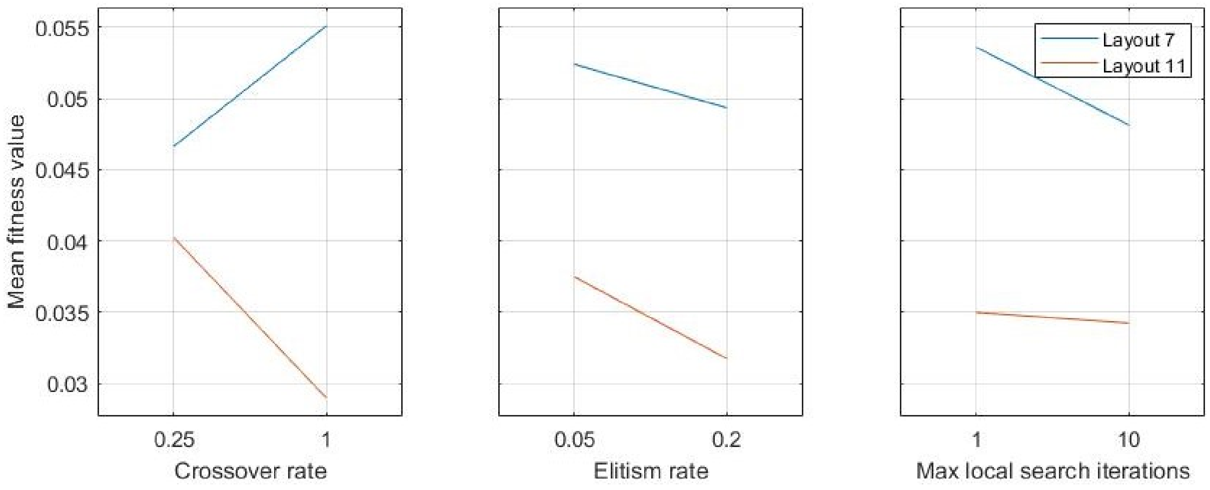


Figure C.5: Main effects of the factors on the response of the system (DOE iteration 3).

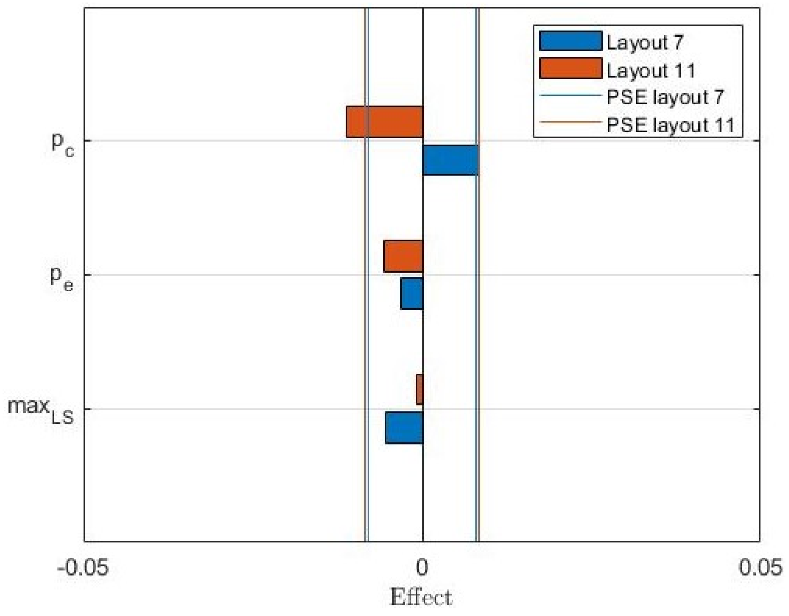


Figure C.6: Pareto chart plotting the effects of the factors on the response (DOE iteration 3).

C.3 Additional penalty factor results

This section presents the results of the penalty multiplication factor analysis for the second layout configuration. Figure C.7 presents the fitness value against the number of evaluations, Figure C.8 presents the difference between attained and the desired gap against the number of evaluations and Figure C.9 presents the normalized fitness value against the number of evaluations. The same steps are conducted as in the report. First of all, the requirement difference value is observed. Herein, the conclusion is drawn that the multiplication factor shall not be larger than 0.5 otherwise, the requirement is not met. Thereafter, the normalized fitness value plot is considered. Herein is observed that a multiplication factor of 0.2 results in a solution with the least amount of absolute penalty value.

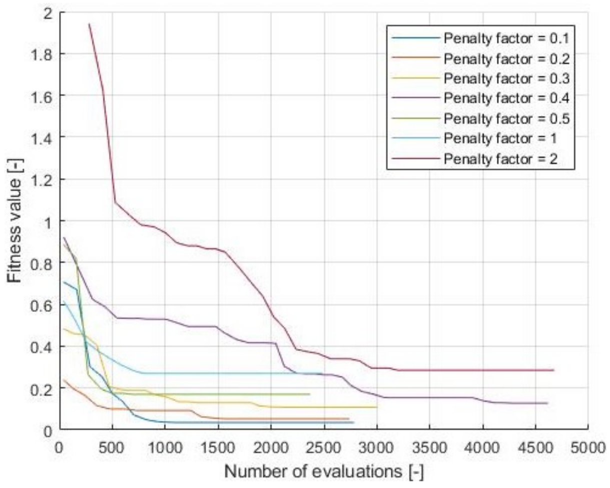


Figure C.7: Fitness value vs. number of evaluations for different penalty multiplication factors (layout index 11).

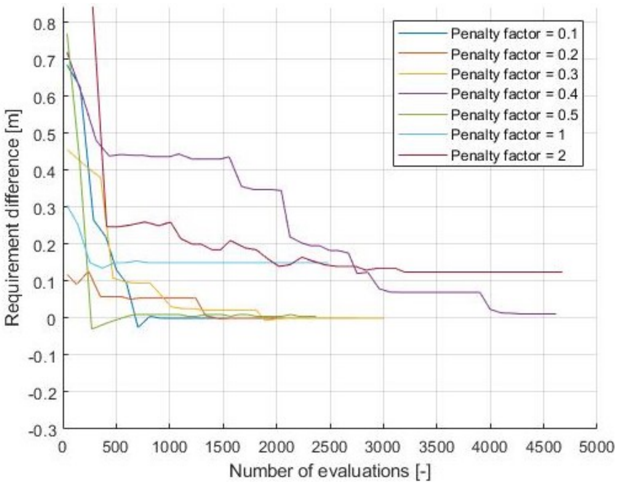


Figure C.8: Requirement difference vs. number of evaluations for different penalty multiplication factors (layout index 11).

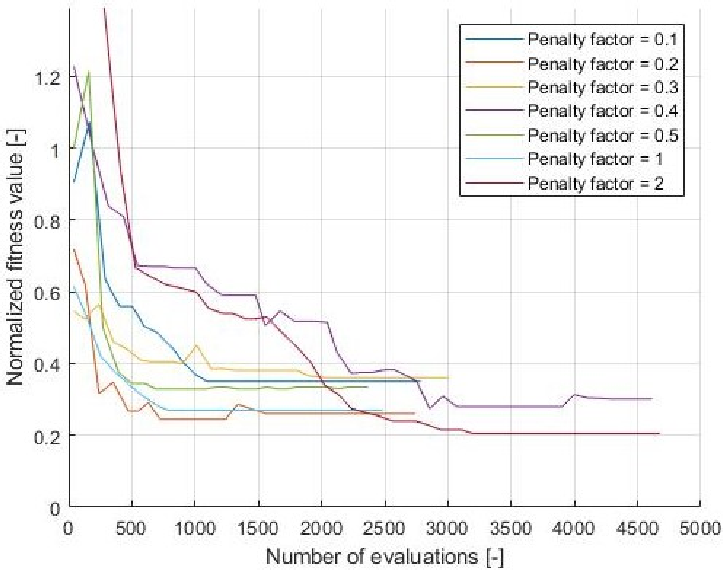


Figure C.9: Normalized fitness value vs. number of evaluations for different penalty multiplication factors (layout index 11).

C.4 Additional use case results

This section presents additional results of the optimization study concerning the create gap use case.

C.4.1 Create gap use case

This subsection presents additional results of the optimization study concerning the create gap use case. Figure C.10, Figure C.11 and Figure C.12 present the fitness value against the number of evaluations, the requirement difference against the number of evaluations and the penalty value against the number of evaluations, respectively. These plots present the results of alternatives 5 and 7. As observed, the framework requires more effort to find a solution where the requirement is met and where the penalty value is as low as possible. Moreover, the penalty value increases whenever the requirement difference decreases, implying that the second conveyor is probably too long to find a satisfactory solution.

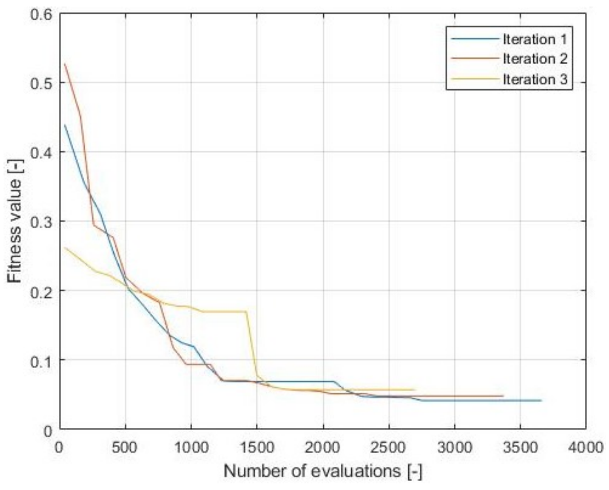


Figure C.10: Fitness value vs. the number of evaluations for alternative 5 and 7.

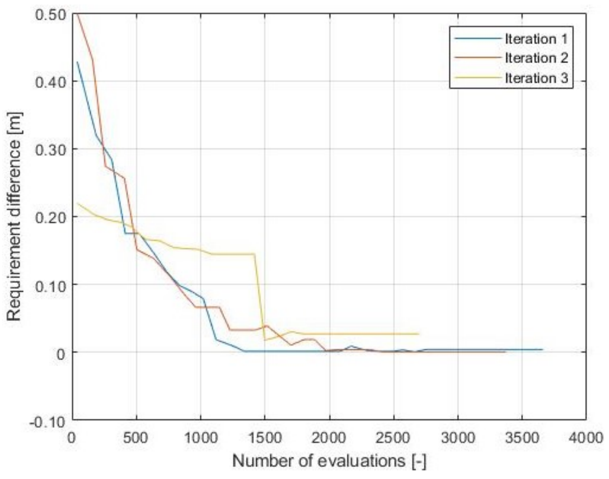


Figure C.11: Requirement difference value vs. the number of evaluations for alternative 5 and 7.

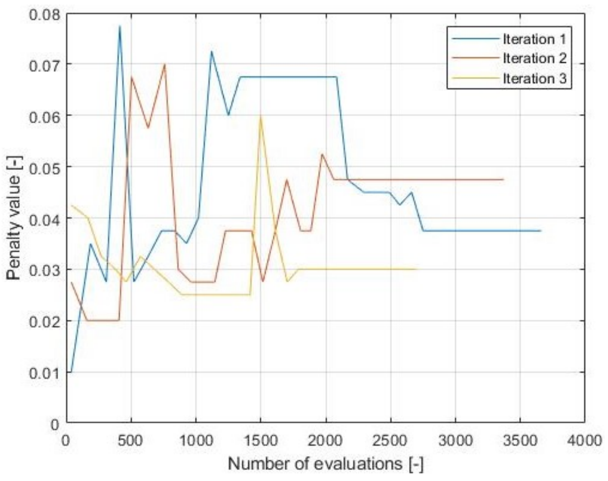


Figure C.12: Penalty value vs. the number of evaluations for alternative 5 and 7.

Figure C.13, Figure C.14 and Figure C.15 present the fitness value against the number of evaluations, the requirement difference against the number of evaluations and the penalty value against the number of evaluations, respectively. The same conclusion can be drawn for the third category as for the second category. Herein, the length of the second conveyor is even longer compared to the length of the second conveyor of categories 1 and 2. As can be observed, it has become more difficult to find a solution that meets the requirement and has an as low as possible penalty value. Additionally, whenever the requirement difference decreases, the penalty value increases, implying that a larger second conveyor length is less desired. Longer conveyor lengths result in more slip.

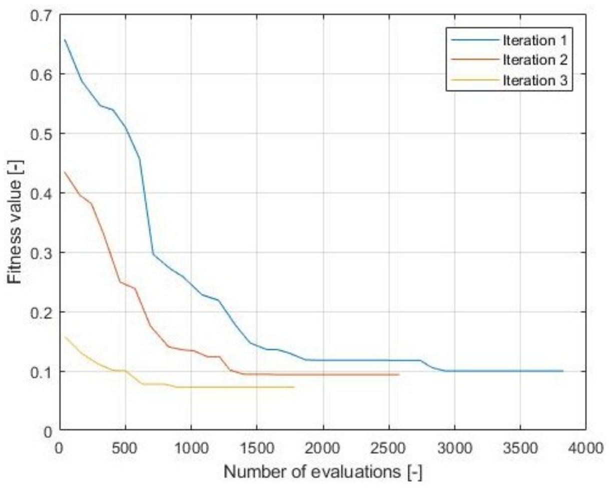


Figure C.13: Fitness value vs. the number of evaluations for alternative 8 and 9.

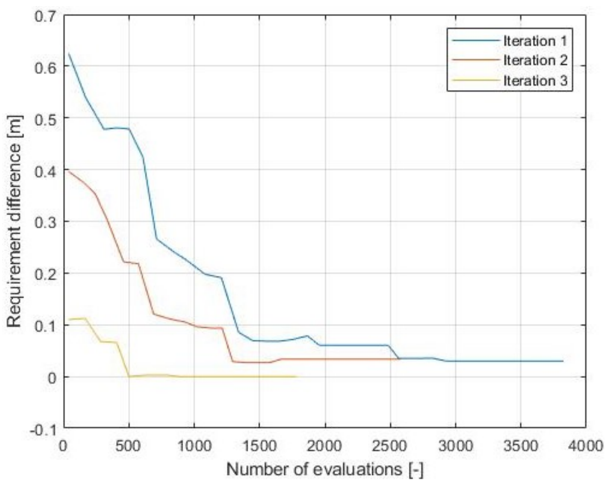


Figure C.14: Requirement difference value vs. the number of evaluations for alternative 8 and 9.

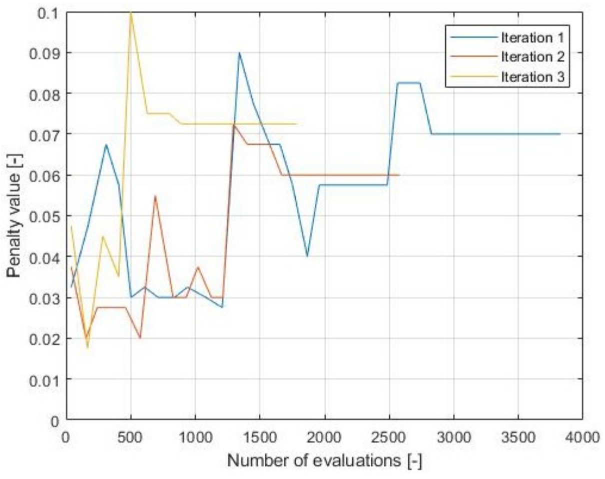


Figure C.15: Penalty value vs. the number of evaluations for alternative 8 and 9.

C.4.2 Create gap use case improvement

This subsection presents intermediate results of the optimization study concerning the two proposed improvements of the create gap use case. Figure C.16, Figure C.17 and Figure C.18 present the fitness value against the number of evaluations, the requirement difference against the number of evaluations and the penalty value against the number of evaluations, respectively. These figures present the intermediate results of the optimization framework to find a final solution for the first proposed improvement.

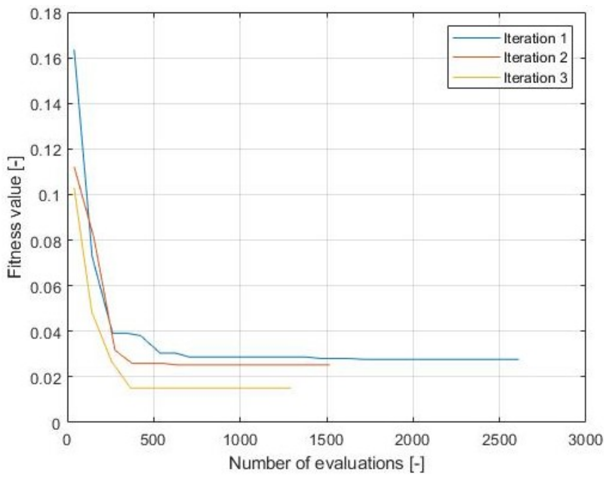


Figure C.16: Fitness value vs. the number of evaluations of the first proposed improvement.

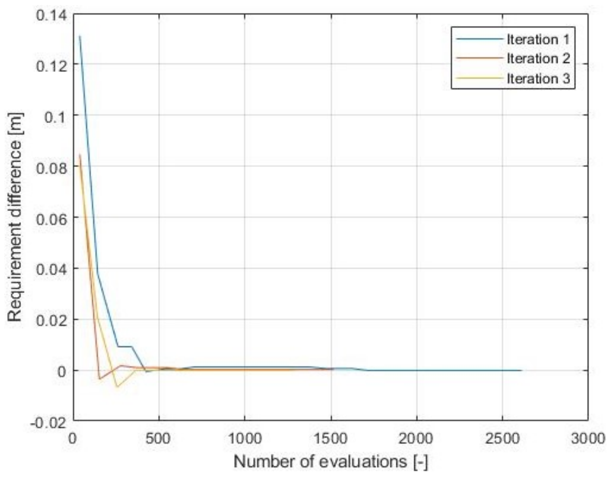


Figure C.17: Requirement difference value vs. the number of evaluations of the first proposed improvement.

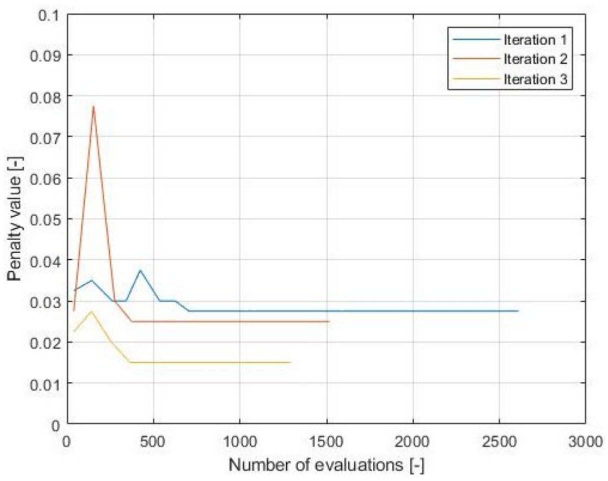


Figure C.18: Penalty value vs. the number of evaluations of the first proposed improvement.

Figure C.19, Figure C.20 and Figure C.21 present the fitness value against the number of evaluations, the requirement difference against the number of evaluations and the penalty value against the number of evaluations, respectively. These figures present the intermediate results of the optimization framework to find a final solution for the second proposed improvement.

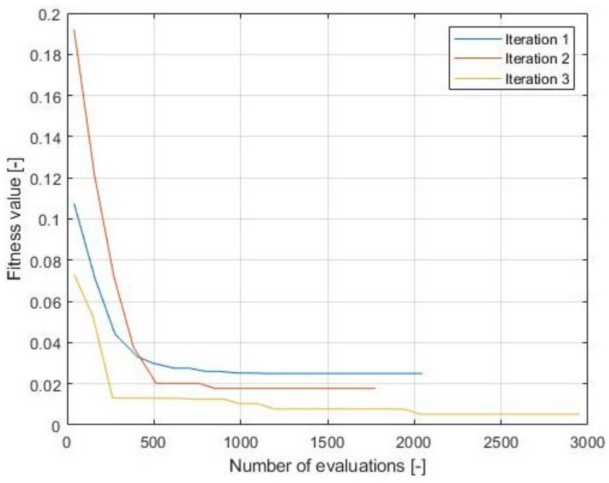


Figure C.19: Fitness value vs. the number of evaluations of the second proposed improvement.

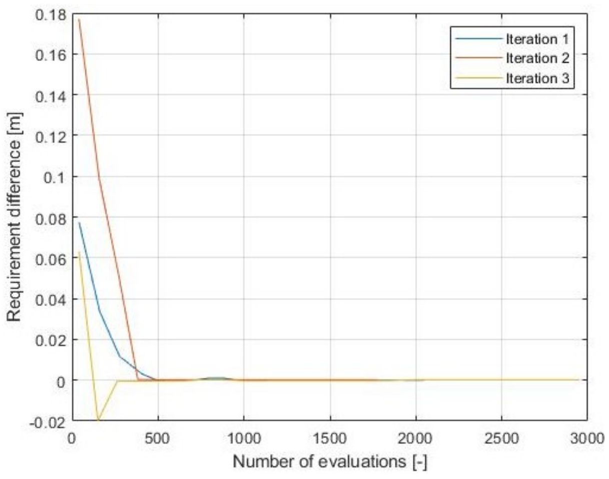


Figure C.20: Requirement difference value vs. the number of evaluations of the second proposed improvement.

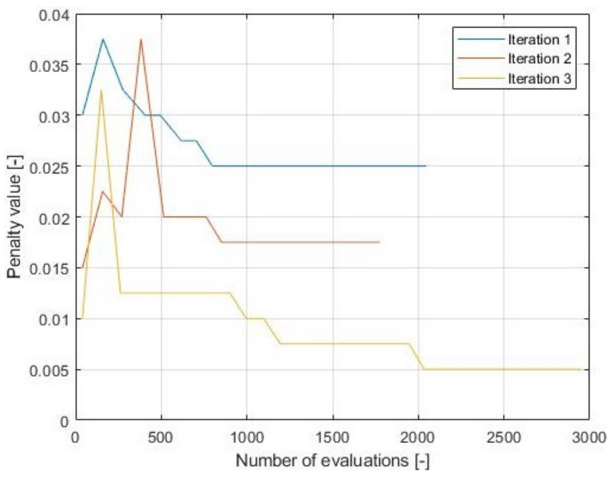


Figure C.21: Penalty value vs. the number of evaluations of the second proposed improvement.