Iñigo Moraza Garcia

Layout planning of Continuous Conveyor Systems Using Modelica

Master's thesis

to achieve the academic degree of

Master of Science

Production Science and Management

submitted to

Graz University of Technology

Supervisors

Dipl.-Ing. Alexander Ortner-Pichler

Assoc.Prof. Dipl.-Ing. Dr.techn. Christian Landschützer

Institute of Logistics Engineering

Univ.-Prof. Dipl.-Ing. habil. Dirk Jodin

Graz, March 2017

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

29.03.2017	I. Moraza
……………………………	…………………………………………………
Date	Signature

ABSTRACT

The Master's Thesis was designed to look into the modelling and layout optimization of conveyor systems using an object-oriented modelling language. For the implementation of the methodology OpenModelica free and open-source software has been used.

First of all, there is a research about conveyor systems analysis and current available methodologies in the calculation of parameters such as throughput and technical availability. Next, the required attributes for the calculation are analyzed with the aim of preparing the methodology and a positioning calculation process is developed. That methodology could be implemented using another object-oriented modelling language than the selected one.

OpenModelica package class is used to build different conveyor modules such as sources, straight or curve conveyors. Inside every class, conveyor modules can be designed to contain defined parameters written by the user or driven parameters in which the attributes are variable depending on the system's layout.

Finally, the methodology to construct continuous conveyor systems, analyze them and optimize its layout is written for the correct use of the software.

Iñigo Moraza Garcia

March 16th, 2017

ACKNOWLEDGEMENTS

I would first like to express my sincere gratitude to everyone who helped me on the Master's Thesis during my studies at Technische Universität Graz.

I specially want to thank my thesis supervisor Dipl.-Ing. Alexander Ortner-Pichler for his help and support since the very beginning. Thanks to his skills and instruction the aim of the project has been properly done. I would also like to acknowledge Assoc.Prof. Dipl.-Ing. Dr.techn. Christian Landschützer for his preparation, supervision and comments on this thesis. This accomplishment would not have been possible without them.

I also feel very grateful to every colleague helped, encouraged and supported me doing this work, either at the university or outside it.

Lastly, I am very much obliged to my parents and sister for supporting me and encouraging me through this years of study and during the process of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PROBLEM STATEMENT

The main of the Master's thesis is to develop a way to design and analyze conveyor system using a modelling language. Before the implementation a theoretical background study has to be done. The structure of the conveyors and the parameters involving the design of the conveyor system's layout have to be studied. That information can be used to choose which attributes are computable and are need for the calculation.

The attribute study has to be used to create a layout and performance calculation methodology that could be implemented in more calculation tools, not only the chosen software for the implementation.

After all the preparation, the methodology will be implemented in the chosen calculation tool, Modelica or another one, that best suits the requirements. At the end it will be a process to design and analyze conveyor systems using the modelling language.

The contribution target of the thesis work is to add another continuous conveyor layout planning tool to the existing ones.

LITERATURE RESEARCH

In the western world, conveyor systems have taken an important place in many industries and in the economy in general. The tendency is to continue with the automation of processes regarding transportation, storage, production lines, etc. Furthermore, in high wage countries this automation focus is gaining importance in order to decrease costs. Consequently, it can be said that the improvement and optimization of conveyor systems is currently interesting and will increase in the near future.

Conveyor systems are mechanical equipment systems to move objects, such as packages, totes, pallets, etc. from one position to another. This method is widely used in many industries. For instance, Automotive, Pharmaceutical, clothing, Computer and electronic, packaging, agricultural, bottling and more.

In order to choose the best conveyor system type and design, some analysis of the system is needed. Depending on the transported material and its requirements, a type of conveyor will suit best.
There are a number of different techniques and methodologies to study conveyor system behaviour and build virtual mock-ups of conveyors. In this Master's Thesis there is a study and methodology development of continuous conveyor systems building and calculation using Modelica programming language, using OpenModelica free and open-source software.

State of the Art of Current Conveyor Systems

Material handling systems

Material handling is the basic theory that is necessary to analyze transport, package, stock and other factors inside a conveyor system or warehouse. Material handling is a system or combination of systems, equipment and operators for transportation, packing and store. It is not limited to transportation.
Material handling means: *[1]*

- The right amount
- The right material
- At the right place
- At the right time
- In the right position
- In the right sequence
- For the right cost

The 20 principles of Material Handling System Design: *[2]*

1. **Orientation Principle**. Study the system relationships prior to specification to determine problems, constraints, goals.
2. **Planning Principle**. Plan to meet requirements efficiently, but maintain flexibility for contingencies.
3. **Systems Principle**. Coordinate and integrate receiving, inspection, storage, product, assembly, packaging, warehousing and distribution systems.
4. **Unit load Principle**. Use large but practical Unit Loads.
5. **Space Utilization Principle**. Effectively use all cubic space.
6. **Standardization Principle**. Use standard equipment and methods if possible.
7. **Ergonomic Principle**. Recognize human limitations in human-mechanical systems.

8. **Energy Principle**. Consider energy consumption in economic analysis.
9. **Ecology Principle**. Design for environmental friendliness.
10. **Mechanization Principle**. Automate if possible to increase efficiency.
11. **Flexibility Principle**. Use methods and equipment that can perform variety of tasks.
12. **Simplification Principle**. simplify and eliminate handling steps if possible.
13. **Gravity Principle**. Gravity is free. Use it.
14. **Safety Principle**. Provide safe methods and equipment. Follow safety codes.
15. **Computerization Principle**. Consider computerization and automation.
16. **System Flow Principle**. Integrate data and material flows.
17. **Layout Principle**. Analyze multiple viable sequencing and layout solutions.
18. **Cost principle**. Compare alternatives in terms of cost per unit handled.
19. **Maintenance Principle**. Use preventive maintenance on equipment.
20. **Obsolescence Principle**. Prepare an economic plan for equipment replacement based on life-cycle costs.

Types of conveyor systems

Conveyors can be categorized in many ways, but in the basic concept of conveyor layout it is possible to divide conveyor systems in two main categories: *[1]*

- Entry and exit point: Exit point can be fixed or with a re-entry
  - Open conveyor system: Fix entry and exit points
  - Re-circulating: The conveyor forms a loop, so the material can stay in the system can travel inside the system until it departs.
- Direction: most conveyors work uni-directionally but some can work in both directions.
  - Uni-directional: Conveyor system with a single direction
  - Bi-directional: Some conveyors can work forward or backwards. They are often designed to have a modular structure.

Some examples of uni-directional conveyor systems:

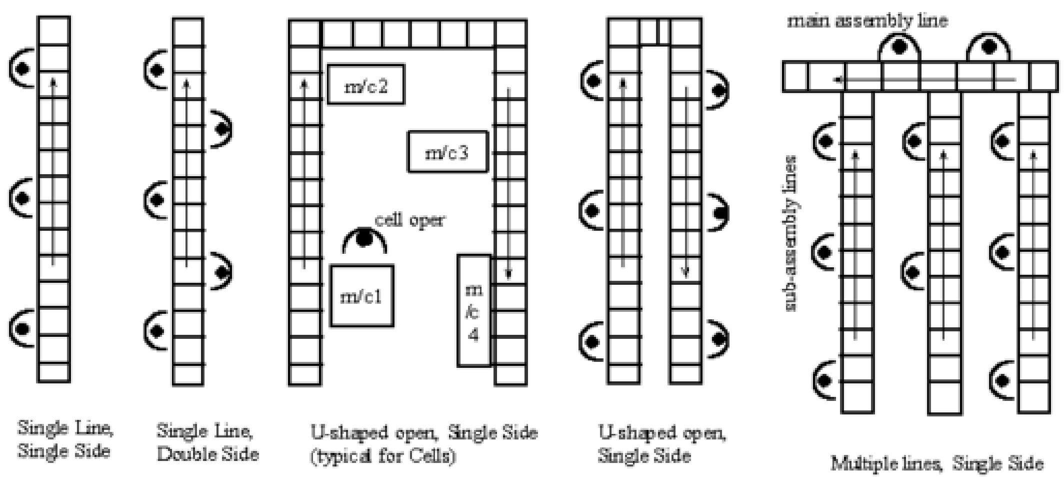


Figure 1: Uni-directional conveyor system *[1]*

Each type have a different performance and features. The system election should be done regarding the system requirements.

Some common applications of uni-directional systems: *[1]*

- Open single lines: assembly lines, chemical plants, etching operations, for example to make lead-frames for electronics chips, etc.
- Double-sided open lines (these can fit more operators per unit length of conveyor): assembly lines
- U-shaped open lines: typical for manufacturing cells. Notice that a cell may have fewer operators than the number of machines.
- U-shaped assembly lines (very compact, single loading and unloading point (often an advantage for material handling): Assembly lines.
- Multiple Lines: typically used for a product with several sub-assembly operations. Each line performs a single module of the product, and the main line assembles the final product.

Closed loop conveyors are used for different requests, such as in machining shops or in some assembly shops. The biggest advantage of closed loop conveyors is that the piece that cannot be work at the moment can wait in the conveyor until it is worked.

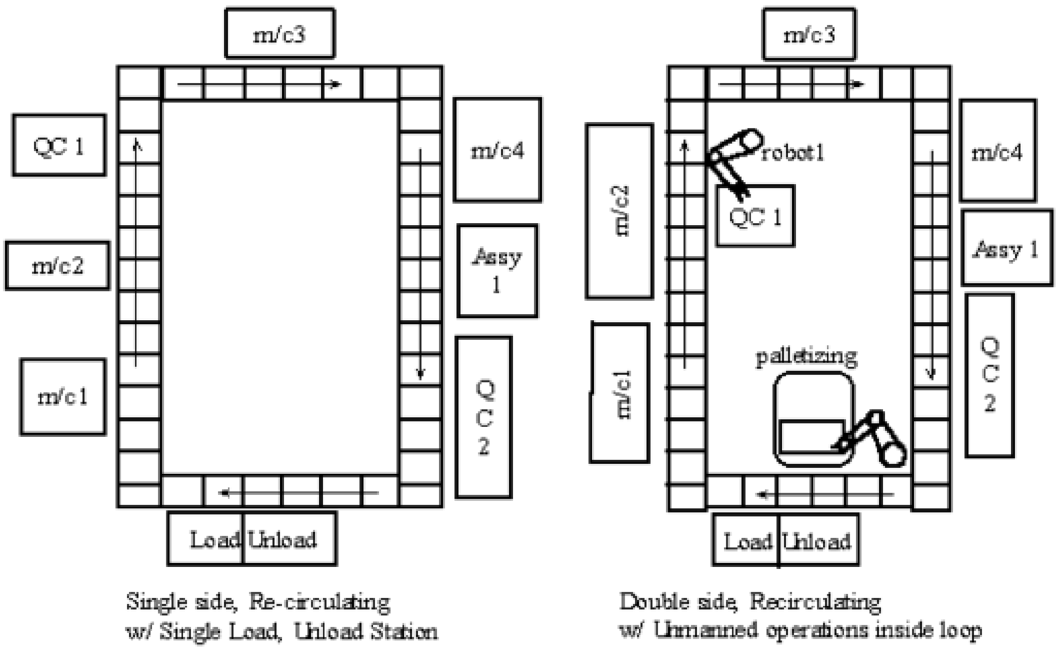

m/c3
QC 1
m/c2
m/c1
m/c4
Assy 1
QC 2
Load
Unload
Single side, Re-circulating w/ Single Load, Unload Station
m/c3
robot1
QC 1
m/c2
m/c1
palletizing
m/c4
Assy 1
QC 2
Load
Unload
Double side, Recirculating w/ Unmanned operations inside loop

Figure 2 : Closed loop conveyor types *[1]*

<u>Layout of the Systems</u>

The Layout of a material handling system is the designing of where the facilities will be placed. Facilities can be machines, pick-up units, conveyor modules, etc. There are different types of defined layouts that can be used for a material handling system, and the right choice is fundamental for the efficient performance of the system. The Single Row Layout Problem (SRLP) consists of finding the most efficient arrangement of a given number of facilities or machines along one side of the material handling path. It is mainly used in manufacturing environments, along with in the organization of rooms in a corridor too, such as in hospitals. *[3]*

Double Row Layout, and Multi Row Layout consist of placing the facilities or machines in both sides of the system. In the case of a Multi Row Layout the system has more than one conveyor lines.

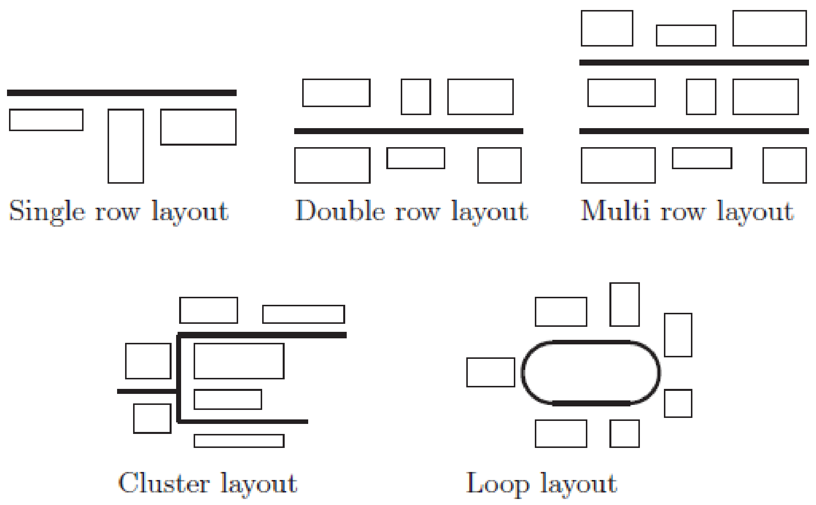


Figure 3: Types of material handling layouts *[3]*

Continuous conveyor

Continuous conveyor is the way to name a conveyor system or material handling system that transports units creating a continuous flow.

This type of conveyor is suitable for conveying large material volumes, or materials that are required continuously, along defined paths. Materials can be loaded and unloaded at multiple points along the conveyance route. This type of conveyor is always (continuously) moving. This is in contrast to intermittent conveyors, which move materials in discrete cycles. Continuous conveyors are classified into floor-bound and non-floor-bound systems. Floor-bound systems can convey the material being handled horizontally, vertically, and at angles of inclination. They often require a lot of floor space and have a defined conveyance route. Meanwhile, non-floor-bound systems are usually rail-mounted. *[4]*

They often feature a simple design, and, thanks to their level of automation, require little supervision. The corresponding investment and maintenance costs depend on the specific conveyor's design and planned routes. The drive mechanisms employed in continuous conveyors are intended for continuous operation and usually consist of a single drive mechanism with a simple design and a low level of power consumption. Long-term savings are achieved with the relatively low operating and labour costs are involved in these systems. However, system expansions can entail difficulties arising from capacity and performance aspects, as well as difficulties arising when a system's task is changed. *[4]*

Continuous conveyors are used, among many other applications, for the delivery and removal of materials and products from the chemical industry, as well as in mining, in the metal industry, in the machining and processing industry, in power plants, in production flows, in storage areas, in the automotive industry, and for joining the individual production processes. *[4]*

The fastest way to find current state of the art at continuous conveyor types is making a market research an checking the Examples of continuous conveyors from a conveyor manufacturer: *[5]*

- Mechanical conveyors
- Gravity roller conveyors with drive mechanisms
- Tubular drag chain conveyors
- Vibratory conveyors (can also be intermittent conveyors, depending on their mode of operation)
- Circular conveyors (vertical carousels)
- Belt conveyors

- Rotary airlocks
- Belt-driven bucket elevators
- Chain conveyors
- Screw conveyors
- Gravity conveyors
- Spiral chutes
- Wheel conveyors, gravity wheel conveyors, ball transfer conveyors, rail conveyors (without drive mechanism)
- Pneumatic and hydraulic conveyors
- Pneumatic conveyance (blower for bulk materials)
- Hydraulic conveyance (pipelines)

Methodologies and Approaches

This section works on the existing theory and techniques to calculate and analyze the performance of a conveyor system. Queuing theory and approaches to technical availability and throughput are explained. It is the basic theory of the performance of a conveyor system.

Queuing Theory

Queuing theory is the mathematical study of queues or waiting lines. In that study a model is built and the waiting times and length can be calculated.

Queuing is used in many fields, such as telephone exchange, supermarket, petrol station, computer systems, production lines, etc. The development of Queuing theory was done by Agner Karup Erlang when dealing with Copenhagen telephone exchange in the beginning of the 20th century.

His works inspired engineers, mathematicians to deal with queueing problems using probabilistic methods. Queueing theory became a field of applied probability and many of its results have been used in operations research, computer science, telecommunication, traffic engineering, reliability theory, just to mention some. It should be emphasized that is a living branch of science where the experts publish a lot of papers and books. *[6]*

Performance measures of queuing systems

To characterize a queuing system we have to identify the probabilistic properties of the incoming flow of requests, service times and service disciplines. The arrival process can be characterized by the distribution of the **inter-arrival times** of the customers, denoted by A(t), that is: *[7]*

$$A(t) = P(\text{ interarrival time} < t)$$

In queuing theory these inter-arrival times are usually assumed to be independent and identically distributed random variables. The other random variable is the **service time**, sometimes it is called service request, work. Its distribution function is denoted by B(x), that is: *[7]*

$$B(x) = P(\text{ service time} < x)$$

The service times, and inter-arrival times are commonly supposed to be independent random variables. *[7]*

The aim of all investigations in queuing theory is to get the main performance measures of the system which are the probabilistic properties (distribution function, density function, mean, variance) of the following random variables: number of customers in the system, number of waiting customers, utilization of the server/s, response time of a customer, waiting time of a customer, idle time of the server, busy time of a server. Of course, the answers heavily depends on the assumptions concerning the distribution of inter-arrival times, service times, number of servers, capacity and service discipline. It is quite rare, except for elementary or Markovian systems, that the distributions can be computed. Usually their mean or transforms can be calculated. *[7]*

For simplicity consider first a single-server system Let ρ, called **traffic intensity**, be defined as: *[8]*

$$\rho = \frac{\text{mean service time}}{\text{mean interarrival time}}$$

Then the **utilization of the server during time** T is defined by: *[8]*

$$\frac{1}{T}\int_0^T x(N(t) \neq 0)dt$$

where T is a long interval of time. As T→ ∞ we get the **utilization of the server** denoted by U_s and the following relations holds with probability 1: *[8]*

$$U_s = \lim_{T\to\infty}\frac{1}{T}\int_0^T x(N(t) \neq 0)dt = 1 - P_0 = \frac{E\delta}{E\delta + Ei'}$$

where P_0 is the steady-state probability that the server is idle $E\delta$, Ei denote the mean busy period, mean idle period of the server, respectively. *[8]*

Characteristics in queuing theory

The most important output factors are: *[9]*

Probability ρ(k): The system performance of a queuing system can be described sufficiently with the probabilities p(k). From p(k) the average values of all other interesting output factors can be derived.

$p(k)$ = P [there are k of orders in the queuing system]

Utilization ρ is the fraction of time where the system is operating from the total time.

$$\boldsymbol{\rho = \frac{\lambda}{m * \mu}}$$

Throughput λ is the average number of customers or product, counted in units per time.

$$\boldsymbol{\lambda = m * \mu * \rho}$$

Response time ts is the total time an order spends in the queuing system.

$$\boldsymbol{t_s = \bar{t} = w + \frac{1}{\mu}}$$

Waiting period w is the time an order wait to the operation starts in the conveyor system. So the response time is the sum of the waiting time and service time.

Queue length $\overline{\boldsymbol{Q}}$ the number of orders in the queue.

Number of orders in the queuing system Ns:

$$\boldsymbol{N_s = \bar{k} = \sum_{k=1}^{\infty} k * \rho(k)}$$

Service disciplines and structure

Disciplines determine how the organization and scheduling of the coming units at the conveyor. The most common ones are:

- First in First out (FIFO): The customer with the longest waiting time is served first.
- Last in First out (LIFO): the customer with the shortest waiting time is served first.
- Service in Random Order (SIRO): The customers are served randomly no matter the waiting time.

- Processor Sharing (PS): Customers are served simultaneously.
- Priority Service (PNPN): Customer with the highest priority is served first.
- Shortest Job First.

Kendall's Notation

The standard way to define and classify queuing nodes is using Kendall's Notation. Kendall used the following notation to describe different queuing system models: *[10]*

A / B / m / K / n / D
A: the arrival process. It describes the arrival process.
B: The service process. The distribution of time of the service of a customer
m: Number of parallel stations. The station are identical.
K: Capacity of the system. Maximum number of customers allowed.
n: Population size. Number of customer sources. This can affect the arrival rate.
D: Service disciplines.

When the system has infinite sources and capacity and the service discipline is First in First Out there are omitted. Often only the first three characteristics are represented.

Arrival and service time specifications

Arrival A and Service B can be defined with various distributions: *[9]*

M Exponential distribution (Markovian)
D Dirac distribution (Cycled processes with constant working times)
G General distribution (distribution is not known)
E Erlang distribution

Throughput

The throughput is the volume of work, information or units that flows though a system. That system could be a computer system, communication system, production line, or many other processes. In the case analyzed in this study, the considered throughput is the one flowing in a conveyor system.

Throughput is probably the most important parameter in a conveyor system or material handling system. With a higher throughput many other performance parameters improves and the costs can decrease. Without throughput the efficiency or conveying speed are not important.

In countries using metric system, throughput in a conveyor system is expressed as "units per hour". In countries using imperial system (miles, feet,...), throughput is defined as "cases per minute".

In many material handling systems the throughput can influence the decision to invest more in automation or increase the speed of the conveyors. *[11]*

Calculation of throughput is easy. Required data is the conveyor velocity and the distance between cases, from edge to edge. Another system to assess throughput is to count the number of products passing by a point during a defined time.

The throughput capacity of a conveyor system is dependent on the slowest speed mainline conveyor and the length of product being transported. While this statement is mathematically true, there are also various operational complexities that impact on throughput. These issues tend to be broad in scope and are for the most part separate from the transportation capacity of the conveyor system.

Aside from the operational issues, conveyor system capacity is based on the speed at which cases are flowing through the system. For example, in a typical picking and sorting system, the metering belt conveyor (located in the mainline feeding the shipping sorter) determines the maximum case feet per minute that the system is capable of delivering. The metering conveyor is usually feed from several production accumulation conveyor lines via a central merge conveyor. *[11]*

Improving Throughput

The increasing of the throughput in conveyor systems is one of the most common issues for the efficiency improvement. The higher is the throughput, the better productivity will be in the system. The conveyor system could manage a higher quantity of products in a shorter time and reduce the total cost of using. *[12]*

"increasing the throughput of a conveyor system usually means increasing equipment speeds. With 1000s of feet of conveyor and 100s of individual power units, this can be a daunting, time consuming and expensive task. An easier and much quicker solution is to update software and controls so that the conveyor equipment can physically handle more product. This approach is gaining favour thanks to advanced software and controls that permit higher product density per lineal foot of conveyor thus increasing the systems throughput capacity without increasing equipment speed".

[12]

Technical Availability

In a conveyor system, the technical availability consist of the percent of the time in relation to the entire time the system is running, that the module or facility is available. In other words, technical availability is the percent of the time that a conveyor is free to be operated, or for a parcel to go by it.

It can also be thought as the probability a system have to be free to be used. For example, a conveyor with 95% of technical availability has that probability of being operational during a scheduled time.

Downtime is the period when a system is not available. Downtime can happen due to some reasons, for example mechanical errors, programmed maintenance stops or lack of stock space when the system is loaded. *[13]*

$$\boldsymbol{A(t)} = \frac{\boldsymbol{t(uptime)}}{\boldsymbol{t(uptime)} + \boldsymbol{t(downtime)}}$$

During mechanical errors or maintenance the conveyor system is completely stopped and there is not any parcel through it. Unavailability instead, is when the module is full and it is not able to accept another parcel. Once sending away a parcel, another one can enter. This way the system is slowed and the total throughput of the system decreases.

The operational availability and maintainability is normally considered during the detail design phase, or after installation of the engineering design. For instance, during operational uses of the design or during process ramp-up and production in process engineering installations. *[13]*

Traditionally technical availability has been considered as a special case of reliability while taking the maintainability of equipment into account. Technical availability was regarded as the parameter that describe system reliability and maintainability characteristics as an index of system efficiency or rating of effectiveness. *[13]*

Availability in engineering design is fundamentally based on the question: what must be considered to ensure that the equipment will be in working condition when needed for a specific period of time? *[13]*

Availability is intrinsically defined as "the probability that a system is operating satisfactorily at any point in time when used under stated conditions, where the time considered includes the operating time and the active repair time" *[13]*

Parallel route planning

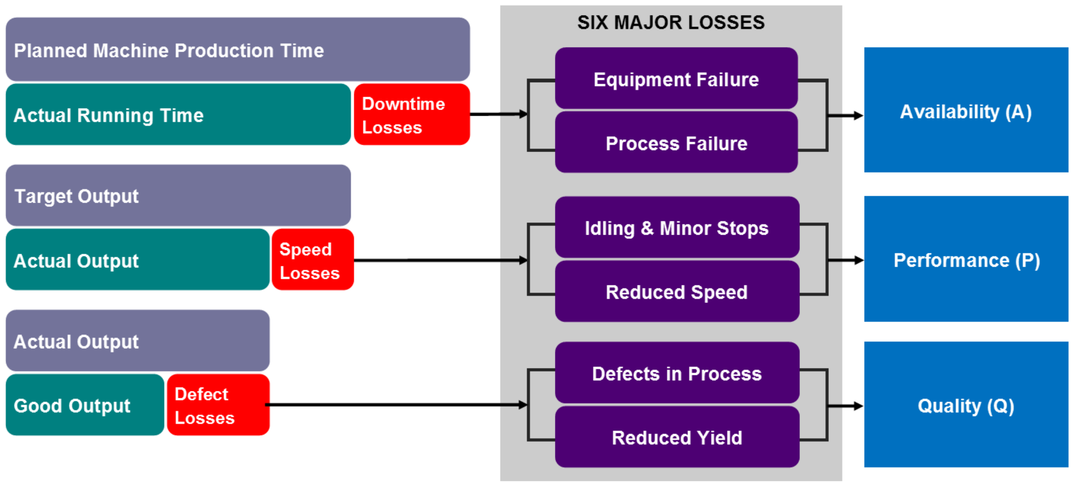
Technical availability calculation in systems that contain parallel routes, that is, two or more conveyors working in parallel, does not have the same calculation parameters than in centralized or single line conveyor systems.

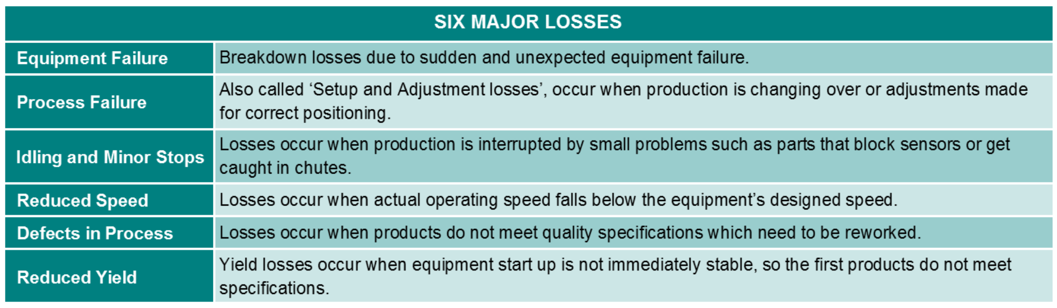
Conveyor systems with parallel routes design can avoid or decrease the effects of downtime, when that happen only in one of the routes. For example, in a 2-routes system with a throughput capacity of 150 parcels/hour each line. but a real throughput of 100 parcels/hour, the effect downtime of one line could be reduced due to increasing the real throughput of the other route to 150 parcels per hour. As a result, the total flow loss is of 50 instead of 100 parcels.

Overall Equipment Efficiency

The OEE, Overall Equipment Efficiency is a term used to analyze the performance efficiency of a manufacturing or transportation system. The system names six causes as the major reasons of performance inefficiency. Those Big Losses are divided into three categories, namely availability, performance and quality.

The factors of six major losses and the structure of OEE are shown below:





SIX MAJOR LOSSES	
Equipment Failure	Breakdown losses due to sudden and unexpected equipment failure.
Process Failure	Also called 'Setup and Adjustment losses', occur when production is changing over or adjustments made for correct positioning.
Idling and Minor Stops	Losses occur when production is interrupted by small problems such as parts that block sensors or get caught in chutes.
Reduced Speed	Losses occur when actual operating speed falls below the equipment's designed speed.
Defects in Process	Losses occur when products do not meet quality specifications which need to be reworked.
Reduced Yield	Yield losses occur when equipment start up is not immediately stable, so the first products do not meet specifications.

Figure 4: Structure of OEE *[14]*

Therefore:

Overall Equipment Effectiveness (OEE) = Availability (A) x Performance (P) x Quality (Q)

Where: *[14]*

- Availability (A) = Actual running time / Planned machine production time
- Performance (P) = (Cycle time/unit x Actual Output) / Actual running time
- Quality (Q) = (Total Production – Defect) / Total production

OEE is a top view metric indicating the gap between the initial and improved performance of a manufacturing unit. It categorizes major reasons for poor performance of equipment, providing the basis for further analysis and improvement in a production process. However, although the concept behind OEE can be applied to multiple sectors and industries, OEE concentrates on the quality, performance and productivity of machines. It ignores other resources of a production line such as the usage of materials, the environment as well as man's efficiency. Hence, a modified approach, Overall Resource Effectiveness (ORE) becomes a more applicable method to a broader range of industries for the measure of production process effectiveness. It addresses the losses associated with the resources and the new factors are known as 'Readiness', 'Availability of Facility', 'Changeover Efficiency', 'Availability of Material', 'Availability of Manpower'. *[14]*

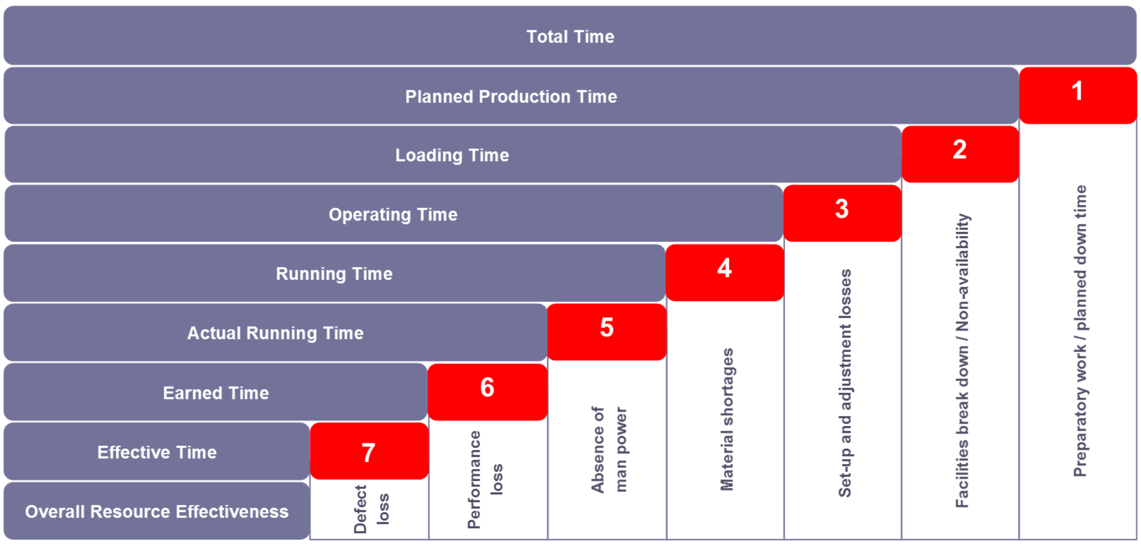


Figure 5: Structure of ORE *[14]*

Therefore, the calculation of the Overall Resource Effectiveness is the multiplication of all the factors: *[14]*

Overall Resource Effectiveness (ORE) = Readiness (R) x Availability of Facility (Af) x Changeover Efficiency (C) x Availability of Material (Am) x Availability of Manpower (Amp) x Performance (P) x Quality (Q)

Calculation of conveyor capacity

The maximum conveyor capacity in kilograms depends on the density of the transported material, its shape, the belt width, etc. In this chapter the analysis is focused on the maximum angle a conveyor can have, in case of designing 3-dimension conveyor systems. For that testing, angle of repose has to be calculated.

"The **angle of repose** of a material is the acute angle which the surface of a normal, freely formed pile makes to the horizontal."

"The **angle of surcharge** of a material is the angle to the horizontal which the surface of the material assumes while the material is at rest on a moving conveyor belt. This angle usually is 5 degrees to 15 degrees less than the angle of repose, though in some materials it may be as much as 20 degrees less." *[15]*

The angle of repose depends on which material is carried, its density, size and many other characteristics.

PREPARATION OF THE METHODOLOGY

Continuous conveyor classification

Continuous conveyor systems cover many different types of transportation and the conditions and units to be carried can vary completely. In order to meet all requirements that a conveyor system could face, there are some different types of conveyors to choose. Each conveyor has some characteristics to meet our performance, angle or budget requirements.
Before choosing one conveyor we have consider the following parameters:

- Product size
- Product weight
- Product variety
- Required throughput
- Technical availability
- Facility dimension
- Height change

Available continuous conveyors: *[16]*

1. Chute Conveyor:
   Inexpensive system to connect conveyors between floors. It makes use of gravity, and it is used for accumulation in shipping areas.

2. Wheel conveyor:
   A series of wheels mounted in a shaft system. The distance between the axis depends on the loads to be transported. More economical than roller conveyor.

3. Roller conveyor:
   Transported materials must have a rigid surface. There are 2 main types, powered and non-powered (gravity).
   - Gravity roller conveyor: Alternative to wheel conveyor with better weight capacities.

- Live Powered Roller conveyor: Belt or chain driven system

4. Chain conveyor:

   Loads are carried directly in one or more endless chains. Parallel chain configuration used to transport pallets. Does not work with accumulation.

5. Slat conveyor:

   Uses discretely spaced slats connected to a chain. It works as a belt conveyor, with the transported unit retaining its position. The orientation and placement of the unit is controlled. Used for heavy loads.

   Tilt slats used for sorting.

6. Flat Belt Conveyor:

   Used to transport light and medium weight loads between operations, departments, levels and buildings. Inclination is possible. High control of the unit.

7. Magnetic Belt Conveyor:

   Magnetic steel belt used to transport ferrous materials such as scrap in vertical movement.

8. Troughed Belt Conveyor:

   Used to transport bulk materials. The belt forms a U-shape in order to hold the transported units in the middle of the belt.

9. Bucket Conveyor:

   Used to move bulk materials in a vertical way. A bucket attached to a cable, automatically unloaded at the end of the movement.

10. Vibrating Conveyor:

    It is a tube or through that vibrates at a relatively high frequency and small amplitude in order to convey individual units products. Used to convey almost all granular materials.

11. Screw Conveyor:

    Tube or U-shaped through which a shaft-mounted helix revolves to push loose material forward in a horizontal or inclined direction. Mostly used in the processing industry. Many applications in agricultural and chemical processing.

12. Pneumatic Conveyor:

    Can be used for bulk and for unit movement of materials. Air pressure moves the products through the system of horizontal and vertical pipes. This system is useful to easily turn and make vertical moves.

    There are two types of Pneumatic Conveyors:

    - Dilute-Phase Pneumatic Conveyor
    - Carrier-System Pneumatic Conveyor

13. Vertical Conveyor:

    Used for low-frequency intermittent vertical transfers.

    - Vertical lift Conveyor
    - Reciprocating Vertical Conveyor

14. Cart-On-Track Conveyor:

    transportation of carts along a track. Carts are moved by a rotating tube. there is a drive wheel connected to each cart and it can control the cart speed. Each cart is independently controlled.

    Accumulation can be achieved by maintaining the drive wheel parallel to the tube.

15. Tow Conveyor:

    Uses towline to provide power to wheeled carriers such as trucks, dollies, or carts that move along the floor. Used for fixed-path travel of carriers.

    The towline can be located either overhead, flush with the floor or in the floor.

    Selector-pin or pusher-dog arrangements can be used to allow automatic switching.

    Used for long distance transportation and high frequency moves.

16. Trolley Conveyor:

    Uses a series of trolleys supported from or within an overhead track. Trolleys are equally spaced in a closed loop path and are suspended from a chain.

    Carriers are used to carry multiple units of products. Not used for accumulation.

    Used in processing, assembly, packaging, and storage operations.

17. Power-and-Free Conveyor:

    Similar to trolley conveyor. It uses discretely spaced carriers transported by an overhead chain. This type of conveyor uses two tracks, one powered and one non-powered (free). Carriers can be disengaged from the power chain and accumulated or switched onto spurs.

18. Monorail:

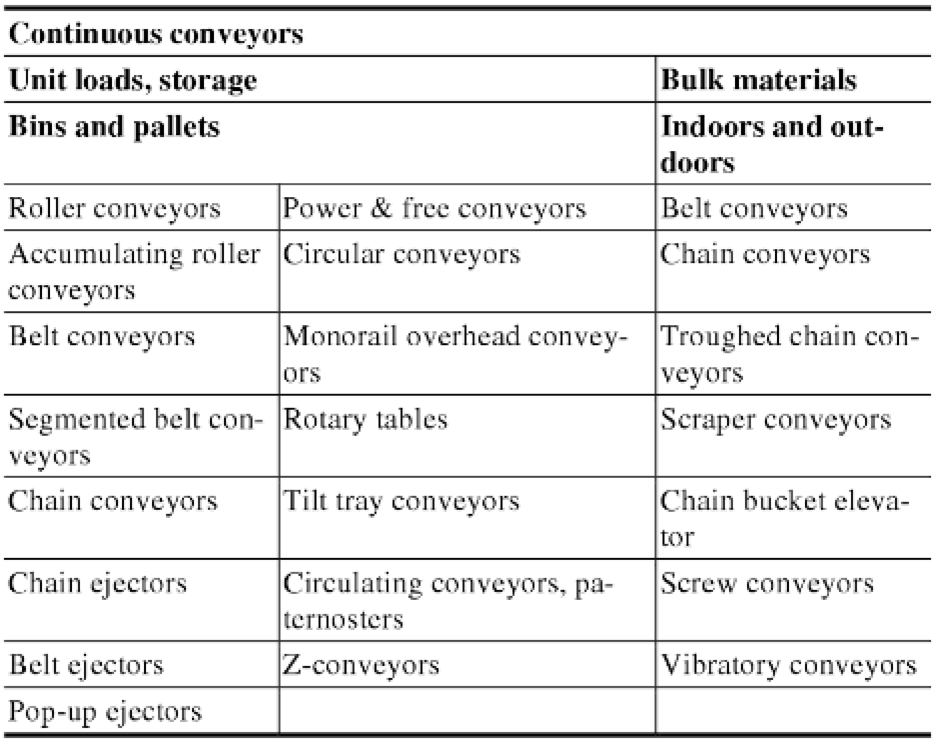
    A single track on which the carriers ride. The carriers can be electrically or pneumatically powered, or non-powered. They can be from a simple hook to an intelligent vehicle device. There are single-carrier and multi-carrier. Single-carrier type have a single-track monorail, similar to a bridge. Multi-carrier type have a track network, in which the carriers operate independently and the track need not be in a closed loop, and a fixed-path automatic guided vehicle system, except that it operates overhead.

19. Sortation Conveyor:

    Sortation conveyors are complex systems that identity, induct, merge and separate products regarding its characteristics or destination.

    - Diverter: Movable or fixed arm used to change the destination of a product. It pulls, pushes or deviate the product
    - Pop-Up Device: Rows of powered rollers that can lift the product and move it out of the conveyor at a certain angle. When the product needs to continue in the conveyor, the device is lowered and works as a normal conveyor.
    - Sliding Shoe Sorter: It uses a series of diverter slats that slide across the horizontal surface to engage product and guide it off conveyor.
    - Tilting Device: Tilting device consist on a tray that inclines in order to place a product from the conveyor to a chute.
    - Cross-Belt Transfer Device: Either continuous loop, where individual carriages are linked together to form an endless loop, or train style where a small number of carriers tied together with potential for several trains running track simultaneously. Each carriage equipped with small belt conveyor, called the cell, that is mounted perpendicularly to the original direction, discharges the product at the appropriate destination.

Table 1: overview of continuous conveyors: *[17]*



Continuous conveyors		
Unit loads, storage		**Bulk materials**
Bins and pallets		**Indoors and outdoors**
Roller conveyors	Power & free conveyors	Belt conveyors
Accumulating roller conveyors	Circular conveyors	Chain conveyors
Belt conveyors	Monorail overhead conveyors	Troughed chain conveyors
Segmented belt conveyors	Rotary tables	Scraper conveyors
Chain conveyors	Tilt tray conveyors	Chain bucket elevator
Chain ejectors	Circulating conveyors, paternosters	Screw conveyors
Belt ejectors	Z-conveyors	Vibratory conveyors
Pop-up ejectors		

For the conveying of unit loads often a combination of continuous and intermittent conveyors are required. A decision for one of these two types of conveyors particularly depends on the following criteria: *[17]*

- Desired quantity conveyed and distance
- Properties of the material involved
- Investment and transport costs

Assembly analysis

Assembly analysis would be the answer to the question, which subsystems are installed within a conveyor system. Conveyor systems consists on different parts or subsystems that are assembled in order to work on a certain way. Depending of the type of conveyor some parts of the system are different, but there are many subsystems that are common for every conveyor.
For example, for a conveyor belt the assembly parts are: *[18]*

- Pulley: consist of two or more pulleys. Pulleys wrap around the belt and transfer the movement to the belt.
- Continuous Belt
- Rollers held by a frame
- Gears and motor: the gears connect the motor with the Driving pulley
- Idlers
- Peripheral Devices

Supports

Metallic supports to hold the structure at the required height

Frame

The frame is a common part in a conveyor system. In every type of conveyor system there is a block or structure that holds the whole assembly and its weight. A frame can be designed with an open design (showing the products that are being transported), with a close design, or with a hybrid model.

Drive unit

The drive unit supplies the power for the movement of the conveyor. Depending on the type of conveyor the transmission can be different. It consists on a electrical motor, brake and power transmission system of any type.

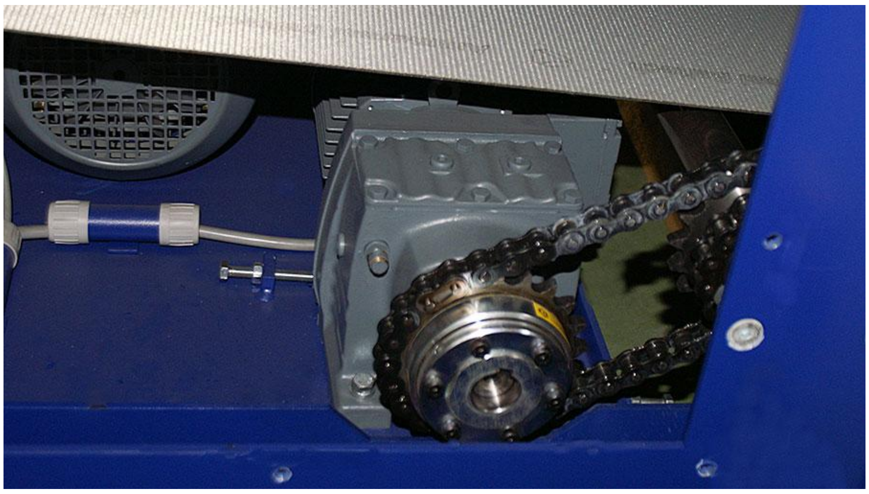


Figure 6: Drive unit module for a telescopic belt conveyor *[19]*

Pulley

Pulleys are assembled in many parts on the system to support or transfer movement in the conveyor system. Some pulleys are used to wrap the belt or transport mechanism on it and other pulleys are installed to transfer the movement of the motor and gears. There are different types of Pulleys: *[20]*

- Like roller only belt wraps around
- Head Pulley: turns belt back around to return. It may be coupled to drive
- Tail Pulley: turns empty belt around for loading. Occasionally coupled to drive
- Drive Pulley: Coupled to motor pulls belt. Usually special grip surface.

Idlers

Idlers are used to support the load of the belt and the transported material, consisting on a shaft surrounded by bearings and then a roll of steel or rubber.

Regarding the design there are flat idlers and composite idlers to have a U-shape in order to keep the materials inside the conveyor.

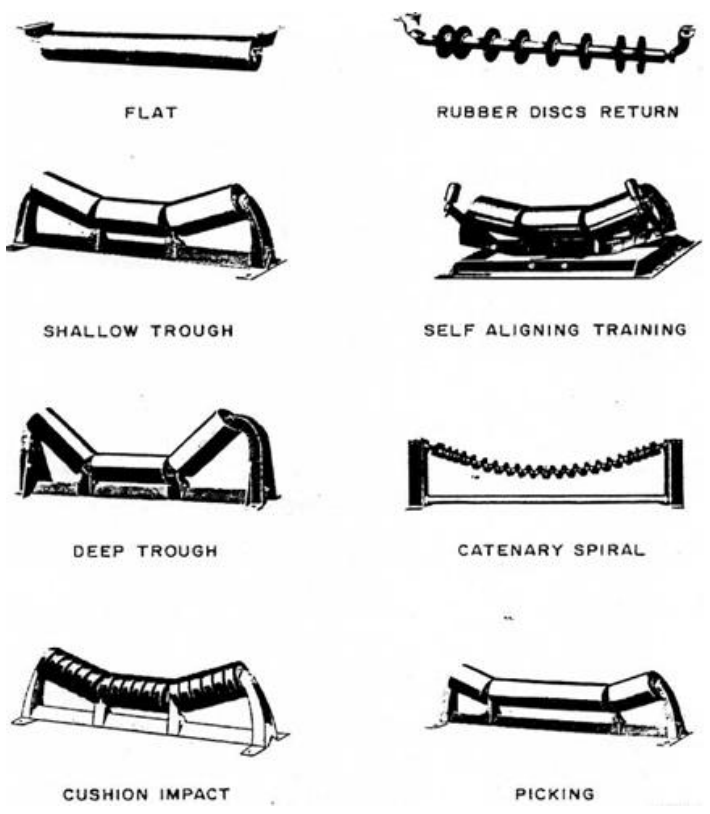


Figure 7: Various types of idlers *[20]*

As seen in the picture above, there are different types of idlers and composition of them. For continuous conveyor systems flat idlers are used and therefore in this study flat conveyors are considered. The other idler types and shapes are usually used for bulk material handling.
There are two main types of idlers, ones used for carrying materials and a belt and the others used to support the belt in the return trip, when the belt is in the inferior side of the belt system. Usually return idlers are flat and carrying idlers can have different shapes. [*18]*
Depending on the required characteristics idler can have different diameter, length, weight capacity, etc.

Tensioning station

A tensioning station is used to control the tension within the belt conveyor. That way the belt can carry materials better, without having spare belt parts or wrinkles. There are different design depending on the needs and size of the conveyor system. It can be installed in the frame.

Chute

Using gravity it moves products to a lower level. Many times is used at the beginning of the conveyor belt to throw the material to it.

It is a cheap way to transport products and it can be used when the design of the conveyor system has different levels and heights.

Lubricator

This equipment inserts oil in the module with the purpose of having the system well lubricated. It focuses specially in the gear structure and high friction areas.

Switches

Safety equipment to interrupt the operation of a conveyor when it is need.

Structure analysis

Which modules are used to build the conveying network?

- straight conveyor
- curve conveyor
- elevator
- Branch (Fork)
- Confluence- merge
- Combination (Merge and fork)
- Node
- Source
- Sink
- Pick-up unit
- Rack or Storage
- Chute

Conveyor systems consist on the combination of different modules with a certain work and design. The design of the system with abstract geometry is a group of straight lines, rectangles, curves, points, etc depending on the represented module. For example, a straight conveyor would be a line, curve conveyor a curve line or a node can be represented as a square.

Those different modules are linked in the system design and in the structure analysis it is noted that many attributes of a module are in relation with the modules connected to it. For example, certain throughput in a conveyor module will be the same in the next module. Technical availability, system capacity and storage capacity are also factors that affect modules in relation to each other.

<u>straight conveyor</u>

This module transport products in a straight way. This module is common and is found in almost every conveyor system. Normal straight belt conveyor consist on the assembly of a Frame, Supports, Drive unit, Continuous belt, Pulleys, Idlers and a Tensioning station.

The abstract geometry for a straight conveyor is a rectangle with rounded corners.

Figure 8: Illustration of a straight conveyor

curve conveyor

It transports parcels turning or rotating the direction a certain angle. The assembly parts for this module are the same as in a straight conveyor but the design is different, as the aim is to turn instead of moving straight.

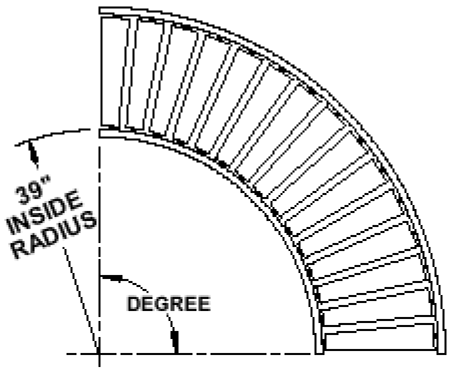


Figure 9: Illustration of a curve conveyor *[21]*

elevator

There are used to move parcels from one level to another one, lower or higher. It also has a certain throughput and technical availability that is usually lower compared with a conveyor, as it has lower parcel handling capacity. This work only considers horizontal networks, so it is not developed in the implementation.

Consist on a Frame, supports, a drive unit that transfer the force to a shelve, pulley.

Branch

The branch divide the parcel flow from one conveyor line to two or more conveyor lines. Each direction can have different amount of previously defined throughput amount, that is, a percent of the incoming parcels per direction. In the calculation it is considered only the percent, not the order. For example, in the 40% of the flow is going to one conveyor it is not considered if the first parcel is going there or not, only the final amount of parcels is taken into account.

The assembly parts needed are the same as a straight conveyor.

In abstract geometry a branch is represented as a circle, with the incoming conveyor and the following outgoing conveyors.

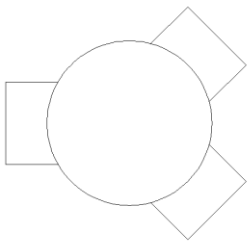


Figure 10: Illustration of a Branch

Confluence - merge

Merges unite different conveyor into one conveyor. The parcel flow is then united into one. It can be a number of conveyors merged from 2 on.

The assembly parts needed are the same as a straight conveyor.

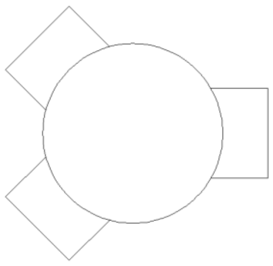


Figure 11: Illustration of a merge

Combination

In a combination there is a mixture between a merge and a branch. A number of conveyors lines come and another number continues from that point

Node

A node is the point where conveyor lines join. It can be a branch, a merge, combination or any other type of different conveyor unions.

Source

The source of a conveyor system is the point where parcel or units are loaded to the conveyor system, their "entrance". The loading method can vary depending of the material handled. Automated systems, forklifts can be used or through manual loading is possible depending the weight and size of the handled units.

The Sources of a conveyor system is usually related to the building in which the warehouse is installed. Products can be transported using trucks, vans or other transportation method and the products are unloaded outside the building.

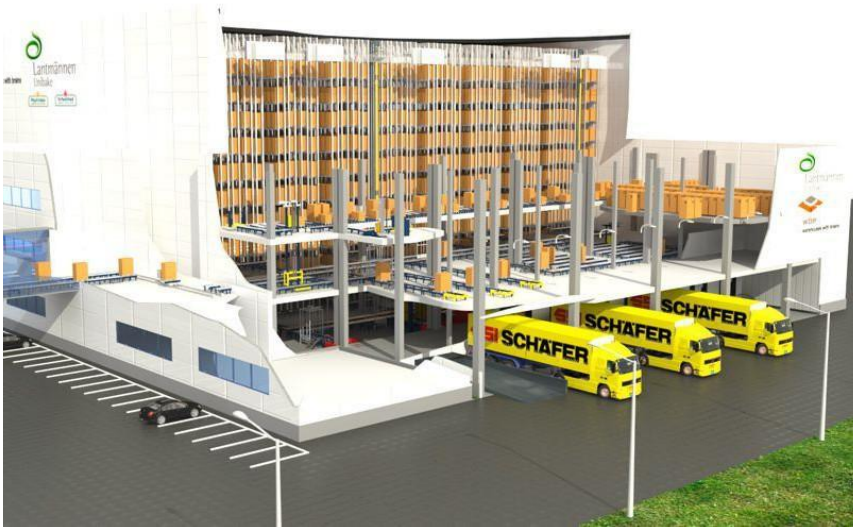


Figure 12: Schäfer Loading/unloading area *[22]*

As we can see in the previous picture, the loading or unloading area is constraint to the size of the building, and then the system will be developed inside the building. Therefore the source modules in the simulation should be constraint to a position or size instead of being variable and movable. The symbol in abstract geometry is a circle with an inner point:

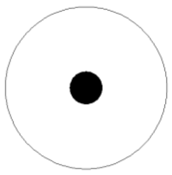


Figure 13: Illustration of a source

Sink

The sink of a conveyor system is it last step, when the parcel flow ends. In this point the parcels are unloaded. In the abstract representation is the end, in the reality it can be loaded into trucks or put in a warehouse using automated systems or forklifts.

The sink of the system is the point where products are unloaded from the conveyor system and loaded into trucks or other transport. This loading area is often outside the building, so it can be constraint to the building size. Although it can be installed inside the warehouse construction too.

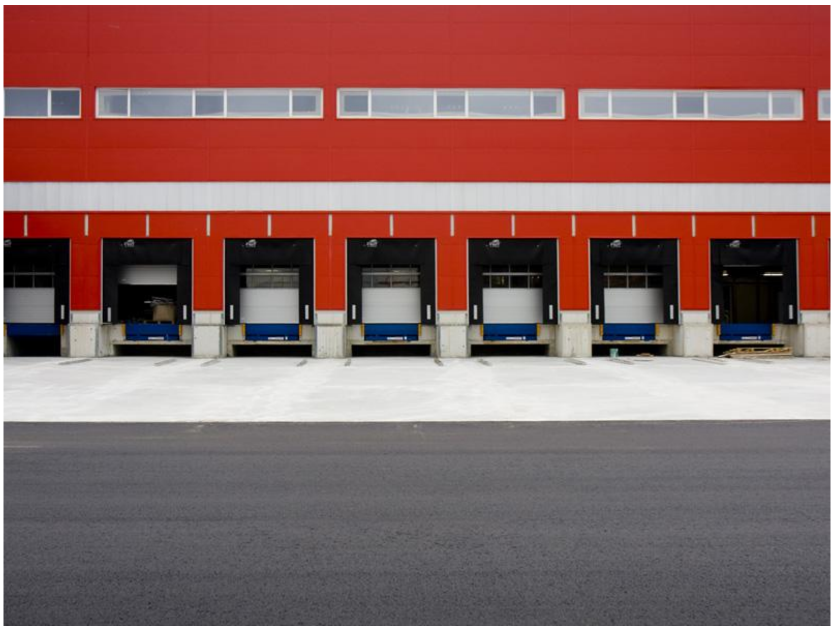


Figure 14: Eroski logistic centre automatic unloading area *[23]*

The symbol of a sink in abstract geometry to draw sketches of conveyor system is a circle with an inner cross:

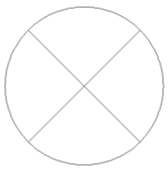


Figure 15: Illustration of a sink

<u>Pick-up unit</u>

Pick-up units are common in warehouses. This module can be at any point of the conveyor system and it is used to unload or load units. The method to load or unload is usually a worker using a picking system such as RF picking, voice picking, pick to light, etc.

<u>Rack or Storage</u>

The storage or warehouse is the module to storage goods or parcels. Usually consist on shelves where the parcels are put. The system can be automated (ASRS,etc), manual or using forklifts.

In abstract geometry, It can be considered as a combination between a source and a sink, as it has an entrance and a way out. with the goods leaving place, the storage is complete.

Chute

Chute conveyor is a very simple and less expensive method to transport products. Using gravity the parcels slide through the chute. In abstract geometry in the calculation it can be considered as a normal straight conveyor, as we just consider its throughput and technical availability.

Attribute study

The examination of the attributes of each module consist on the definition of the parameters affecting the performance of the conveyor system. The attributes are used to describe the modules and in to understand how the system works.

Every module have a sort of attributes that explain its characteristics and the behaviour when the module is functioning. Some attributes can be shared by different modules, as well as other attributes are unique for a certain module type.

Those attributes can be classified in different ways and they can be sorted in groups. Some parameters are predetermined and cannot be changed. To be precise, some attributes are easily obtained in the specifications of the module and are constant and not changeable without changing the whole module. Other parameters can change even if the module stays identical, as it can be influenced by adjacent ones. For example, even if the throughput of a conveyor would be the one of the previous conveyor even if its maximum throughout or the parcel handing capacity is higher.

Parameters are divided into this two big groups, defined parameters and dependent parameters. In order to classify the attributes in each group, an analysis is required. After testing which attributes are need to calculate others is possible to classify them and find their characteristics.

First of all, the attributes to be studied have to be defined. List of parameters:

- Length
- Width
- Position (x,y)
- Angle
- Distance between parcels
- Throughput
- Technical availability
- Capacity
- Number of branches/Merges
- Angle of the curve
- Stock
- Conveying velocity
- Drive torque
- Drive speed (rpm)
- Applicable motor capacity (KW)

Attributes of the modules

Every module have some attributes that can share with others or be unique for a certain module. At this point, when the required attributes for the calculation are defined they are related to the module types.

<u>straight conveyor:</u>

- Conveyor ID
- Length
- Width
- Position (x,y)
- Angle
- Throughput
- Technical availability
- Distance between parcels
- Capacity

<u>Curve conveyor:</u>

- Conveyor ID
- Length
- Width
- Position (x,y)
- Angle
- Throughput
- Technical availability
- Distance between parcels
- Capacity
- Angle of the curve

<u>Elevator:</u>

- ID
- Height
- Width
- Throughput
- Technical availability
- Capacity

<u>Branch:</u>

- ID
- Number of branches
- Length
- Width
- Position (x,y)
- Throughput
- Technical availability
- Distance between parcels
- Capacity

<u>Merge:</u>

- ID
- Number of Lines merged
- Length
- Width
- Position (x,y)
- Throughput
- Technical availability
- Distance between parcels
- Capacity

<u>Combination:</u>

- ID
- Number of Lines merged and branched
- Length
- Width
- Position (x,y)

- Throughput
- Technical availability
- Distance between parcels
- Capacity

Source:

- ID
- Length
- Width
- Position (x,y)
- Throughput
- Technical availability
- Distance between parcels
- Capacity

Sink:

- ID
- Length
- Width
- Position (x,y)
- Throughput
- Technical availability
- Distance between parcels
- Capacity

Pick-up unit:

- ID
- Size
- Position (x,y)
- Throughput
- Technical availability
- Distance between parcels
- Capacity

Rack of storage:

- ID
- Dimensions
- Throughput (if it is automated)
- Technical availability (if it is automated)
- Stock Capacity

Chute:

- ID
- Length
- Position (x,y)
- Angle
- Throughput
- Technical availability
- Distance between parcels
- Stock
- Capacity

Relation of attributes between modules

Attributes of the conveyors can be defined and completely independent, or driven by other parameters. In order to classify those parameters the following relations are done. Each table contains the parameters of two module types, so next a relation of module attributes is done in order to know which parameters from the first module are required to calculate the parameters from the second module.

Table 2: Relation of attributes between two straight conveyors:

Straight 1		Straight 2	
Module Parameters	Layout Parameters	Module Parameters	Layout Parameters
Length Width Throughput T. availability Capacity	Position x,y Angle	Length Width Throughput T. availability Capacity	Position x,y Angle

Relation:

Position(x,y), Length, Width, Angle → Position (x,y)

Angle → Angle

Maximum throughput, Technical availability → Throughput

Table 3: Relation of attributes between a straight conveyor and a branch:

Straight		Branch	
Module Parameters	Layout Parameters	Module Parameters	Layout Parameters
Length Width Throughput T. availability Capacity	Position (x,y) Angle	Length Width Throughput T. availability Capacity	Position (x,y) Number of branches % of flow per branch

Relation:

Position(x,y), Length, Width, Angle $\rightarrow$ Position (x,y)

Maximum throughput, Technical availability $\rightarrow$ Throughput

Table 4:Relation of attributes between a straight conveyor and a merge:

Straight		Merge	
Module Parameters	Layout Parameters	Module Parameters	Layout Parameters
Length Width Throughput T. availability Capacity	Position (x,y) Angle	Length Width Throughput T. availability Capacity	Position (x,y) Number of branches % of flow per merge

Relation:

Position (x,y), Length, Width, Angle $\rightarrow$ Position (x,y)

Maximum throughput, Technical availability $\rightarrow$ Throughput

Table 5: Relation of attributes between one straight conveyor and a curve:

Straight		Curve	
Module Parameters	Layout Parameters	Module Parameters	Layout Parameters
Length Width Throughput T. availability Capacity	Position (x,y) Angle	Length Width Throughput T. availability Capacity Angle of the curve	Position (x,y) Angle

Relation:

Position (x,y), Length, Width, Angle $\rightarrow$ Position (x,y)

Angle $\rightarrow$ Angle

Maximum throughput, Technical availability $\rightarrow$ Throughput

Table 6: Relation of attributes between one straight conveyor and a sink:

Straight		Sink	
Module Parameters	Layout Parameters	Module Parameters	Layout Parameters
Length Width Throughput T. availability Capacity	Position x,y Angle	Length Width Throughput T. availability Capacity	Position x,y Angle

Relation:

Position (x,y), Length, Width, Angle $\rightarrow$ Position (x,y)

Maximum throughput, Technical availability $\rightarrow$ Throughput

Table 7: Relation of attributes between a source and a straight conveyor:

Source		Straight	
Module Parameters	Layout Parameters	Module Parameters	Layout Parameters
Length Width Throughput T. availability Capacity	Position x,y Angle	Length Width Throughput T. availability Capacity	Position x,y Angle

Relation:

Position (x,y), Length, Width, Angle $\rightarrow$ Position (x,y)

Maximum throughput, T. availability $\rightarrow$ Throughput

Module Attributes

In this chapter the attributes of the different module attributes inside the module, without interconnections with any other module. Design Structure Matrix (DSM) of the attributes of each module is created.

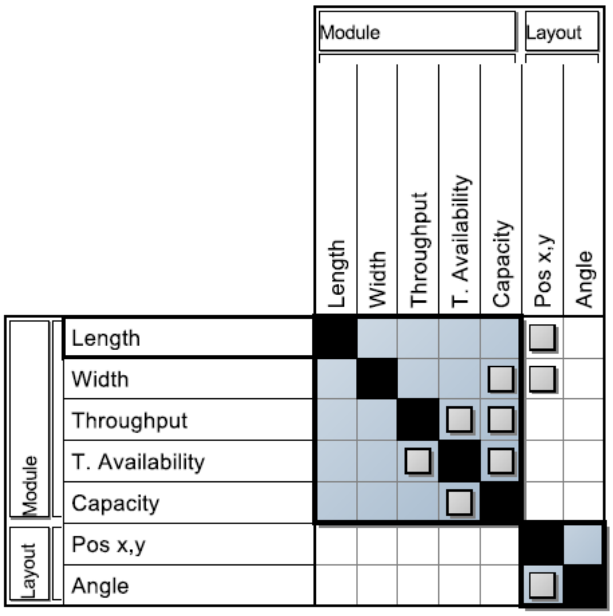
The tools used is the Cambridge Advanced Modeller (CAM). As the description of the program made by Cambridge University: "the Cambridge Advanced Modeller is a software tool for modelling and analysing the dependencies and flows in complex systems - such as products, processes and organisations. It provides a diagrammer, a simulation tool, and a DSM tool".

CAM is free for research, teaching and evaluation. The only requirement for the usage is to cite their work if CAM is used in support of published work.

The analysis consist on the relation between the attributes of the module, that is, if one attribute influences another one of the same attribute.

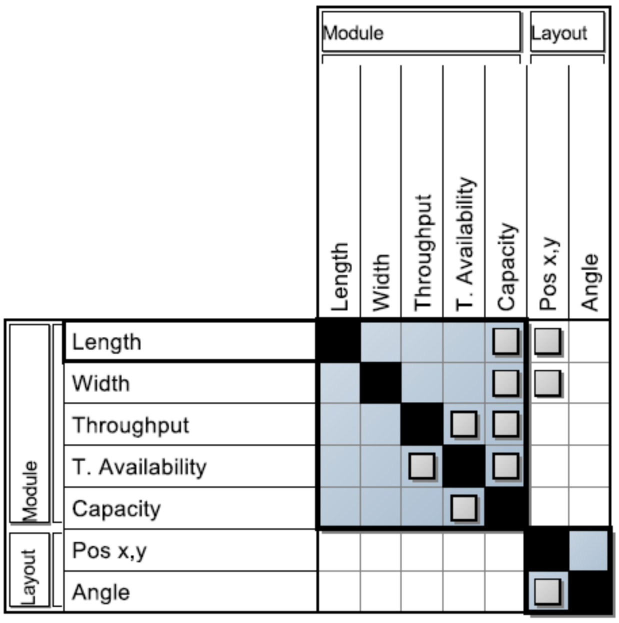
The direction to read the matrix is starting from the left to the right. In that line there are squares in each attribute from the top that is related to the attribute in the left.

Table 8: Straight conveyor module DSM:



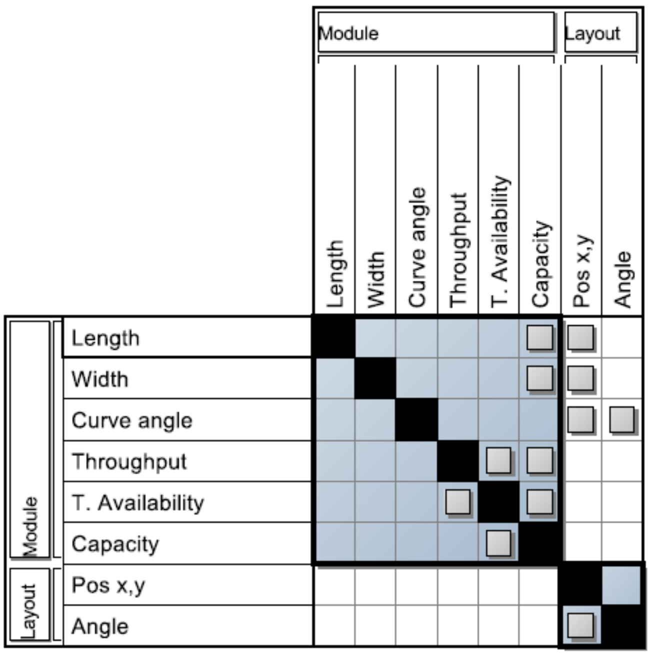
		Module					Layout	
		Length	Width	Throughput	T. Availability	Capacity	Pos x,y	Angle
Module	Length	■					□	
Module	Width		■			□	□	
Module	Throughput			■	□	□		
Module	T. Availability			□	■	□		
Module	Capacity				□	■		
Layout	Pos x,y						■	
Layout	Angle						□	■

Table 9: Source module DSM:



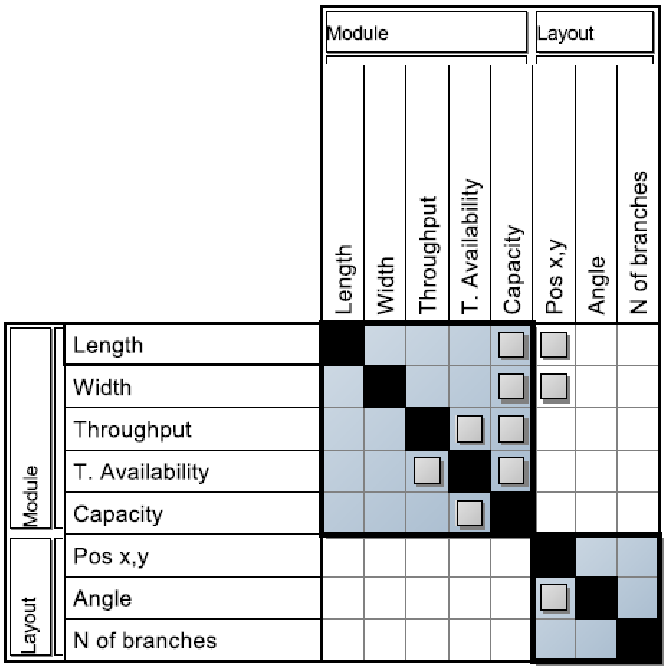
		Module					Layout	
		Length	Width	Throughput	T. Availability	Capacity	Pos x,y	Angle
Module	Length	■				□	□	
	Width		■			□	□	
	Throughput			■	□	□		
	T. Availability			□	■	□		
	Capacity				□	■		
Layout	Pos x,y						■	
	Angle						□	■

Table 10: Sink module DSM:



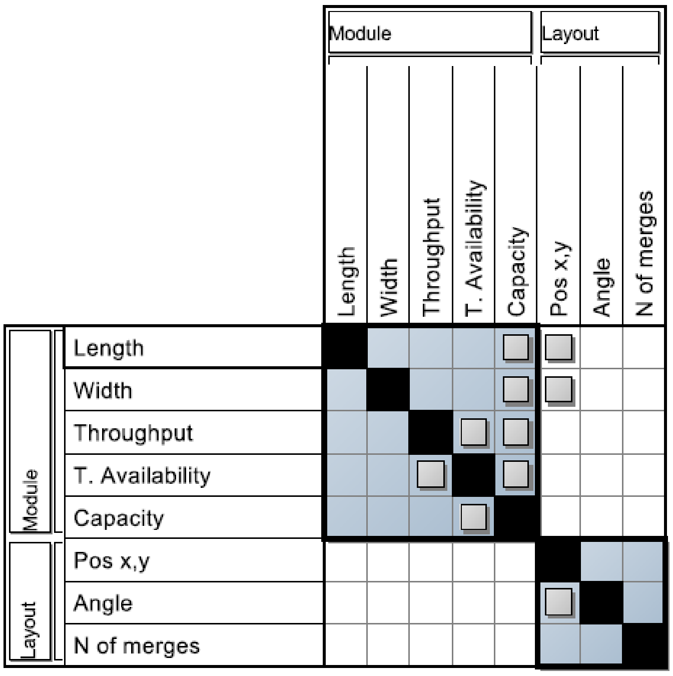
		Module						Layout	
		Length	Width	Curve angle	Throughput	T. Availability	Capacity	Pos x,y	Angle
Module	Length	■					□	□	
	Width		■				□	□	
	Curve angle			■				□	□
	Throughput				■	□	□		
	T. Availability				□	■	□		
	Capacity					□	■		
Layout	Pos x,y							■	
	Angle							□	■

Table 11: Branch module DSM:



		Module					Layout		
		Length	Width	Throughput	T. Availability	Capacity	Pos x,y	Angle	N of branches
Module	Length	■				☐	☐		
	Width		■			☐	☐		
	Throughput			■	☐	☐			
	T. Availability			☐	■	☐			
	Capacity				☐	■			
Layout	Pos x,y						■		
	Angle						☐	■	
	N of branches								■

Table 12: Merge module DSM:



		Module					Layout		
		Length	Width	Throughput	T. Availability	Capacity	Pos x,y	Angle	N of merges
Module	Length	■				☐	☐		
	Width		■			☐	☐		
	Throughput			■	☐	☐			
	T. Availability			☐	■	☐			
	Capacity				☐	■			
Layout	Pos x,y						■		
	Angle						☐	■	
	N of merges								■

Once all the module attribute interrelations are done it is the moment for the analysis of the drive unit. The table clarifies the parameters that influence other ones:

Table 13: Drive unit attributes DSM:

	Drive torque	Drive speed	Conveying velocity	Motor capacity KW
Drive torque	■	□	□	
Drive speed		■	□	
Conveying velocity			■	
Motor capacity KW	□		□	■

<u>Conclusion</u>

As we can see in the modules Design Structure Matrix and the tables there are some attributes that are related to others or need a calculation in order to be known. Those attributes are "driven parameters" and are related to the figure of "defined parameters", the ones that are fixed and have to be known before the computing starts.

<u>Defined parameters</u>

Defined parameters are the ones that can be specified by the physical components of the conveyor. This information is available at the product specifications and it is not possible to be changed. therefore, Defined parameters are the ones in relation with geometry, drive unit, design, etc.

The defined parameters are:

- Geometry:
  - -Length
  - -Width
- Layout:
  - -Position x,y
  - -Angle (Regarding Coordinate system)
  - -Distance between Parcels

- Logistics output:
  - -Conveying velocity

<u>Driven parameters</u>

Driven parameters are related to the performance of the conveyor system and every module of the conveyor can influence the other modules parameters. Defined parameters such as size, position or distance between parcels are used to calculate those parameters.

Driven parameters can be divided into three groups, logistic attributes, technical aspects and drive unit specifications.

- Logistic output:
  - -Throughput

- Technical aspects:
  - -Technical availability
  - -Capacity
  - -Branch: % of throughput each direction
  - -Stock

- Drive unit Specifications:
  - -Drive torque
  - -Drive speed
  - -Applicable motor capacity (KW)

DEVELOPMENT OF THE METHODOLOGY

Network composition

The network composition consist on the mapping of a conveyor system modules. On an IT-basis, modules need to be linked to each other. When a module is placed next to the previous module, linked to it, every module need to have a position. Therefore, A coordinate system is used in order to position each modules place. Usually the 0 point is located at the beginning of the conveyor system, in the source or one of the sources.

After finding the attributes each module have, it has to be noticed that in a conveyor system there is a relation between modules that affect their attributes. For example, the position of a module (x_1, y_1) is totally in relation with the previous module. other attributes such as the throughput is influenced by the smallest throughput in the conveyor line.

Those relations are simple for linear conveyors without any parallel conveyor lines, with a source and sink, and conveyor modules in between. For conveyor systems with branches, merges and parallel conveyor lines the relation is more complex, as there are more technicalities.

Levels of abstraction

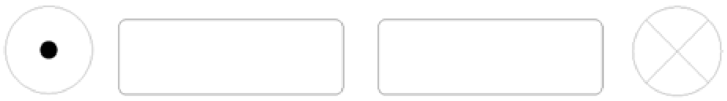
In the Development of the methodology, the analysis can have different levels of complexity, that is, the calculation can be abstract or simple depending on the attributes taken into account and the abstraction of the analysis.

In this study, three levels of abstraction are defined. With the attributes defined in the previous chapters each parameter relationships are clear and there is the possibility to group them into three different groups.

The first level of abstraction is the position of the conveyor system. Therefore, the attributes for the calculation are the geometry parameters and the layout parameters. The geometry parameters are the size and length of the modules, while the layout ones are the angle and the position (x, y, z). With this level of abstraction the conveyor system mapped and the position and the situation of a module regarding the other modules.

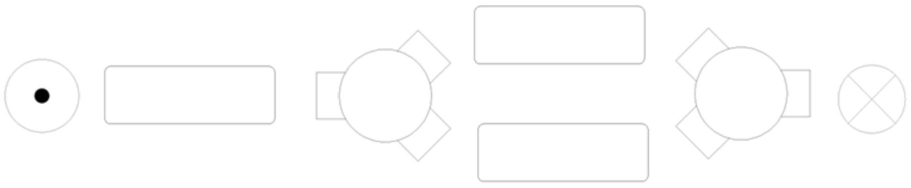
The second level of abstraction consists in the parameters that shows the performance of the modules. Using the information gained in the previous level and some attributes of each module, the parameters can be calculated. Those attributes are the throughput, regarding a logistic output, and technical aspects such as technical availability and capacity (stock). Branch modules also contain the parameter of percent of throughput for each direction that shows the division of the low through the different conveyors.
The third level of abstraction set the parameters needed for the previous results. In order to have the required throughput we need the drive unit working at a certain speed. Therefore, the third level of abstraction parameters are the ones concerning the drive unit and the performance of the conveyor module itself. That is, the drive torque, drive speed and applicable motor capacity (Kw) regarding the drive unit specifications, and the conveying velocity regarding the conveyor belt (or other system) specifications.

.<u>First level of abstraction</u>



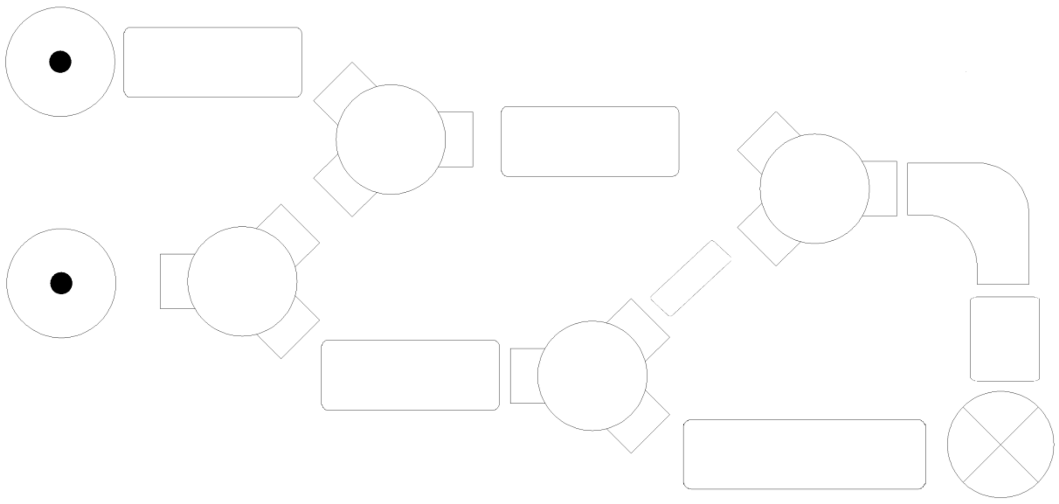
x_1, y_1 0,0	$x_2 = x_1 + {}^{l_1}/_2$ $y_2 = y_1$	$x_3 = x_2 + {}^{l_1}/_2 + {}^{l_2}/_2$ $y_3 = y_2$	$x_4 = x_2 + {}^{l_2}/_2$ $y_4 = y_3$

Now, the position is calculated in a system with a branch and a merge. Not only the x position but the y position changes.



x_1, y_1 0,0	$x_2 = x_1 + {}^{l_1}/_2$ $y_2 = y_1$	$x_3 = x_2 + {}^{l_1}/_2 + {}^{l_2}/_2$ $y_3 = y_2$	$x_4 = x_3 + {}^{l_2}/_2 + {}^{l_3}/_2$ $y_4 = y_3 + {}^{w_{branch}}/_2 - {}^{w_3}/_2$
$x_5 = x_3 + {}^{l_2}/_2 + {}^{l_4}/_2$ $y_5 = y_3 + {}^{w_{branch}}/_2 - {}^{w_4}/_2$	$x_6 = x_4 + {}^{l_5}/_2 + {}^{l_3}/_2$ $y_6 = y_4 - {}^{w_{Merge}}/_2 + {}^{w_4}/_2$	$x_7 = x_6 + {}^{l_5}/_2$ $y_7 = y_6$	

In order to have an example of a more complex conveyor system the following system is analyzed. there are two sources to load the system and one sink. As there is a system consisting on branches, merges and conveyors, both x and y position is changing as well as the angle of the modules.



1. $x_1, y_1(0,0)$
2. $x_2 = x_1 \frac{l_1}{2}.\cos(\alpha)$ $\qquad y_2 = y_1 + \frac{l_1}{2}.\sin(\alpha\ )$
3. $x_3 = x_2 + \frac{l_1}{2} + \frac{l_{M1}}{2}$ $\qquad y_3 = y_2 + \frac{W_{M1}}{2} + \frac{W_1}{2}$
4. $x_4 = x_3 + \frac{l_{M1}}{2} + \frac{l_{B1}}{2}$ $\qquad y_4 = y_3 + \frac{W_{M1}}{2} + \frac{W_{B1}}{2}$
5. $x_5 = x_4 - \frac{l_{B1}}{2}$ $\qquad y_5 = y_4$
6. $x_6 = x_3 + \frac{l_2}{2}.\cos(\alpha)$ $\qquad y_6 = y_3 + \frac{l_2}{2}.\sin(\alpha)$
7. $x_7 = x_4 + \frac{l_{B1}}{2} + \frac{l_3}{2}.\cos(\alpha)$ $\qquad y_5 = y_3 + \frac{l_3}{2}.\sin(\alpha)$
8. $x_8 = x_7 + \frac{l_3}{2}.\cos(\alpha) + \frac{l_{B2}}{2}$ $\qquad y_8 = y_7 + \frac{l_3}{2}.\sin(\alpha)$
9. $x_9 = x_8 + \frac{l_{B2}}{2} + \frac{l_5}{2}.\cos(\alpha)$ $\qquad y_9 = y_8 - \frac{w_{B2}}{2} + \frac{l_5}{2}.\sin(\alpha)$
10. $x_{10} = x_9 + \frac{l_5}{2}.\cos(\alpha) + \frac{l_{M2}}{2}$ $\qquad y_{10} = y_9 - \frac{l_5}{2}.\sin(\alpha) - \frac{w_{M2}}{2}$
11. $x_{11} = x_{10} + \frac{l_{M2}}{2}.\cos(\alpha) + \left(\frac{l_{c1}}{2} - \frac{W_{c1}}{2}\right).\cos(\alpha)$

    $$y_{11} = x_{10} + \frac{l_{c1}}{2}.\sin(\alpha)$$

12. $x_{12} = x_{11} + \left(l_{c1} - \frac{w_{c1}}{2}\right).\cos(\alpha) + \frac{l_{12}}{2}.\cos(\alpha)$

    $$y_{12} = x_{11} + \left(l_{c1} - \frac{w_{c1}}{2}\right).\sin(\alpha) + \frac{l_{12}}{2}.\sin(\alpha)$$

13. $x_{13} = x_8 + \frac{l_{B2}}{2} + \frac{l_{13}}{2}.\cos(\alpha)$

$y_{13} = y_8 + \frac{w_{B2}}{2} + \frac{l_{13}}{2}.\sin(\alpha)$

14. $x_{14} = x_{12} + \frac{l_{12}}{2}.\cos(\alpha)$

$y_{14} = y12 + \frac{l_{12}}{2}.\sin(\alpha)$

<u>Note: </u>during the implementation of the methodology the positioning of the conveyors was changed. The solution was to create input and output interfaces within the modules, so the positioning of the conveyors was changed. Then, in the implementation the positioning is at the beginning and end of the modules, not a single point.
The explained methodology could be applied in different modelling languages, so the single positioning or interface positioning could be used.

<u>Second level of abstraction</u>

After the calculation of the position and layout of the conveyor system the calculation of the performance parameters of the conveyors. Using the known throughput and technical availability is possible to calculate the number of parcels per minute flowing in each conveyor. The previous systems are used to show the throughput calculation:

1)

1. $flow_s = \lambda_s.ta_s$
2. if $\lambda_1.ta_1 \geq \lambda_s.ta_s \rightarrow flow_1 = \lambda_s.ta_s$
   then $flow_1 = \lambda_1.ta_1$
3. if $\lambda_2.ta_2 \geq flow_1 \rightarrow flow_2 = flow_1$
   then $flow_2 = \lambda_2.ta_2$
4. if $\lambda_{\text{sin}k}.ta_{\text{sin}k} \geq flow_2 \rightarrow flow_{\text{sin}k} = flow_2$
   then $flow_{\text{sin}k} = \lambda_{\text{sin}k}.ta_{\text{sin}k}$

2)

1. $flow_s = \lambda_s.ta_s$
2. if $\lambda_1.ta_1 \geq \lambda_s.ta_s \rightarrow flow_1 = \lambda_s.ta_s$
   then $flow_1 = \lambda_1.ta_1$

3. if $\lambda_B . ta_B \geq flow_1 \rightarrow flow_2 = flow_1$
   then $flow_{branch} = \lambda_B . ta_B$
4. if $\lambda_2 . ta_2 \geq flow_{branch} . 0{,}4 \rightarrow flow_2 = flow_{branch} . 0{,}4$
   then $flow_2 = \lambda_2 . ta_2$
5. if $\lambda_3 . ta_3 \geq flow_{branch} . 0{,}6 \rightarrow flow_3 = flow_{branch} . 0{,}6$
   then $flow_3 = \lambda_3 . ta_3$
6. if $\lambda_m . ta_m \geq flow_2 + flow_3 \rightarrow flow_m = flow_2 + flow_3$
   then $flow_m = \lambda_m . ta_m$
7. if $\lambda_{\text{sin}k} . ta_{\text{sin}k} \geq flow_m \rightarrow flow_{\text{sin}k} = flow_m$
   then $flow_{\text{sin}k} = \lambda_{\text{sin}k} . ta_{\text{sin}k}$

3)

1. $flow_{s1} = \lambda_{s1} . ta_{s1}$
2. if $\lambda_1 . ta_1 \geq \lambda_{s1} . ta_{s1} \rightarrow flow_1 = \lambda_{s1} . ta_{s1}$
   then $flow_{branch} = \lambda_1 . ta_1$
3. if $\lambda_{M1} . ta_{M1} \geq \text{flow}_{2} + \text{flow}_{\text{branch1}} \rightarrow flow_{merge1} = \text{flow}_{2} + \text{flow}_{\text{branch1}}$
   then $flow_{branch1} = \lambda_{M1} . ta_{M1}$
4. if $\lambda_{B1} . ta_{B1} \geq \text{flow}_{\text{source2}} \rightarrow flow_{branch1} = \text{flow}_{\text{source2}}$
   then $flow_{branch1} = \lambda_{B1} . ta_{B1}$
5. $flow_{s2} = \lambda_{s2} . ta_{s2}$

<u>third level of abstraction</u>

Once the throughput and the other performance parameters are calculated the next step is to choose a drive unit to transfer enough power to the system, in order to match the performance parameters . Therefore, the specifications of the drive unit have to be adequate and have enough capacity to move the system.

There are many conveyor drive unit manufacturers, such as Siemens. The main task at this point is to find the proper drive unit according to our needed throughput and the parcel characteristics as size or weight.

Evaluation of calculation tools

In order to apply the methodology, some options could be selected. In this chapter, various calculation tools are analyzed with the purpose of finding the most suitable one to meet the academic requirements.

Definition of criteria

The main need is to find a calculation tool to calculate the parameters within a conveyor system. But the question is, which is the best programming tool for our aim? Probably is it possible to fulfil the objective with more than one tool as some are similar or have overlapping characteristics. The change from a tool to another can be a change in precision, speed of development, portability between platform, etc. Furthermore, we have to take into account not technical factors such as the ease to learn the program (when we are a beginner on the program), the support of the program and number of users.

The Programming language selection has been one unorganized and hard job, but these days some selection techniques has been developed.

Several factors must be considered when selecting a programming language, and whilst different curricula place greater emphasis on different factors, all must be considered. We seek to develop a comprehensive set of selection criteria and a process for the application of these criteria to evaluate programming languages to be used in programming classes. The selection criteria must take into account the programming features of each language under consideration, the appropriateness of each of these features for beginning (and perhaps advanced) programming courses, the present and future industry acceptance of each language, the availability and quality of textbooks, the costs associated with adopting each language, the infrastructure and support implications of each language, and the impact of the decision on the tactical and strategic direction of the department and curriculum. *[24]*

In that Study they selected some criteria in order to choose the best programming language.

Table 14: language selection criteria: *[24]*

Reasonable financial cost for setting up the teaching environment
Availability of student/academic version
Availability of textbooks
Language's stage in life cycle
System requirements of student/academic/full version
Operating system dependence
Open source (versus proprietary)
Academic acceptance
Industry acceptance
Marketability (regional and national) of graduates
Easy to use development environment
Ease of learning basic concepts
Support for target application domain (such as scientific or business)
Full-featured language (versus scripting)
Support for teaching approach (function first, object first or object early)
Object-oriented support
Good debugging facilities
Support of web development
Support for safe programming
Advanced features for subsequent programming courses
Availability of support
Training required for instructors and support staff

For the criteria to choose the best program for the purpose of analysing conveyors system throughput and technical availability, some of them are chosen. For the thesis work requirements all of the criteria points are not needed. Physical construction is also added as it can be positive to build conveyor systems easily and it helps the understanding.

- Object-oriented support: The method has to be **object-oriented**, that is, it has to work with different types of conveyors no matter if a belt conveyor, chain conveyor, roller conveyor, etc is installed. The parameters of the conveyor system should work with any conveyor.

- Availability of student/academic version: The software has to be available without paying.
- Ease of learning fundamental concepts: Learning curve is something we have to take into account. Basic concepts include the sequence, selection and iteration control structures, as well as arrays, procedures, basic input/output, and file manipulation.
  Sometimes the learning curve difference between programming languages is big enough to make a choosing decision.
- Language stage in life cycle: the life cycle of a programming language should also be considered. It can affect the availability of textbook and information, as well as support and impact in the industry.
  It is preferable to choose a program in the earlier stages, when it has still development and growing chances than a programming language in its declining years.
- Proprietary/open source: This refers to the entity that controls the evolution of a language and its associated development environment. Open source programming languages can be developed and improved by any member of the open source community.
- Debugging facilities: The software should control the programming language in order to control errors and evaluate the debugging facilities included in it.
- Physical construction: The possibility of having physical construction is also considered.

[24]

<u>Rating</u>

After choosing the topics to be analyzed, the next step is to define the importance of each point. The selection of the value is really vital for the last selection of the calculation tool. Furthermore, a change in the influence of change could lead to working with another programming tool.
The rating is thought to be in relation with the methodology and analysis needs. If the thesis work would be different, with different aim, the analyzed points and its rating would be different. The importance is taken from 1 (less important) to 5 (very important).

In conclusion, here is the rating for the choosing criteria:

- Object-oriented support:5
- Availability of student/academic version:3
- Ease of learning fundamental concepts:3
- Language stage in life cycle:4
- Proprietary/open source: 2
- Debugging facilities:3
- Physical construction:1

Tools to be analyzed

In order to have the best and full-scale research, the criteria analysis takes into account at least three calculation tools or programming languages. Using the following programming languages it could be possible to make calculations of conveyor layout planning, throughput and technical availability.
These are the analyzed tools:

MS Excel

Microsoft office Excel is a spreadsheet program, and the programming language attached to the program could be used for the methodology development. That is, Visual Basics for Applications (VBA) is the programming language to be considered.
VBA is a programming language and integrated development environment from Microsoft Excel. It can be considered as object-oriented.

OpenModelica

OpenModelica is based on the Modelica programming language for modelling, simulating optimizing and analyzing dynamic systems. OpenModelica software is open source and it is available free on the internet.
Modelica, quoting their own definition, is an object-oriented, declarative, multi-domain modelling language for component-oriented modelling of complex dynamic systems. It has mechanical, electronic, electrical, thermal, hydraulic, etc type of subcomponents.

LogNet

LogNet is an intelligent system that provides interactive advice to end-users on how to design cost-effective logistics networks. It implements its capabilities by utilizing a class of AI techniques known as model-based reasoning. These techniques solve problems by analyzing the structure and function of a system. *[25]*

That system focuses more on the costs than at the parameters of a conveyor system, so it is discarded.

MATLAB

Simulink/Matlab is an alternative to design and analyze conveyor systems.

Simulink is a block diagram environment that works in the Matlab programming environment. It is used for multi-domain simulation and Model based design.

Phyton

This programming language is high level, for general purpose, interpreted and dynamic.

The Phyton philosophy is to be readable and simplicity, trying to make short codes.

Phyton is free and open-source.

As it is required to reach the specifications, it can be object-oriented but it is not considered complete object-oriented. The reason is the Phyton philosophy that doesn't like to hide anything. In addition, that programming language doesn't support encapsulation, one important attribute for the object-oriented view.

Rate per programming tool

- MS Excel
    - Object-oriented support: 8
    - Availability of student/academic version: 10
    - Ease of learning fundamental concepts: 10
    - Language stage in life cycle: 3
    - Proprietary/open source: 3
    - Debugging facilities: 7
    - Physical construction:0

- OpenModelica
    - Object-oriented support:10
    - Availability of student/academic version:10
    - Ease of learning fundamental concepts:7
    - Language stage in life cycle:7
    - Proprietary/open source: 10
    - Debugging facilities:7
    - Physical construction:10

- MATLAB/ Simulink
    - Object-oriented support: 10
    - Availability of student/academic version: 4
    - Ease of learning fundamental concepts:5
    - Language stage in life cycle:5
    - Proprietary/open source: 3
    - Debugging facilities: 10
    - Physical construction: 10

- Phyton
    - Object-oriented support:4
    - Availability of student/academic version:10
    - Ease of learning fundamental concepts: 3
    - Language stage in life cycle:5
    - Proprietary/open source: 10
    - Debugging facilities: 5
    - Physical construction: 0

Calculation

Table 15: Evaluation of calculation tools results

	Object-oriented (5)	Student version (3)	Learning (3)	Life cycle (4)	Open source (2)	Debugging (3)	Physical construction (1)	Total
MS Excel	8	10	10	3	3	7	0	139
OpenModelica	10	10	7	7	10	7	10	180
Simulink	10	4	5	5	3	10	10	143
Phyton	4	10	3	5	10	5	0	114

Conclusion

It is clear that Modelica programming language, and particularly OpenModelica is the choice that best fits to the needed academic requirements. OpenModelica can be used for the programming and calculation needed and it doesn't have weaknesses regarding the topic or the current available situation.

OpenModelica contains all the non-technical advantages for the development of the methodology, such as being free and open-source. Additionally, the learning curve for Modelica is not negative.

Furthermore, It contains all the technical requirements. It is object-oriented, it includes a strong debugging tool and it allows Physical viewing and construction, giving the chance to build systems easier and helping for the understanding of the dynamic systems.

IMPLEMENTATION OF THE METHODOLOGY

Once the methodology to be followed is finished, it is the time to implement it in the computer program. Therefore, it is time to apply the calculation in OpenModelica, as it is the tool chosen for the implementation.

OpenModelica is a free and open-source modelling, simulating, analyzing and optimization environment based on Modelica language. Its long-term development is supported by a non-profit organization, the Open Source Modelica Consortium (OSMC). The OSMC describes the goal of OpenModelica effort is to create a comprehensive Open Source Modelica modelling, compilation and simulation environment based on free software distributed in binary and source code form for research, teaching, and industrial usage. They invite researchers and students, or any interested developer to participate in the project and cooperate around OpenModelica, tools, and applications.

Aim of the methodology

The aim of the thesis is to create a methodology that can be used for building complex conveyor systems in a easier way. As a result the programming part has to be avoided as much as possible and try to work in the Diagram View environment of OpenModelica. It is more likely to make errors or typos when programming at the Text View environment and it is more "user friendly" working with the mouse at the Diagram View.

Therefore, the objective is to have the different types of modules already developed and listed, prepared to be loaded and dragged to the Diagram. Then, connections can be created to link the conveyor modules creating the layout required. The attributes of each module such as size or throughput can be changed right-clicking in the *parameters* section. In case of changes in the layout it is possible to delete a single module and drag another one, followed by new connections.

Methodology and interfaces

In order to create and connect different types of conveyors, a technique to transfer position, throughput and other data has to be developed. In this work the solution was to create interfaces within the modules in order to connect those interfaces and transmitting the position and throughput. Each module contains an input interface and output or way out interface. In the case of branches, merges or other special cases a module can contain more than one input or output interfaces.

The interface contains three data that can be increased programming them if required. The data in the interface are *x* position, *y* position and throughput. Therefore, each module will have defined the entrance position(x, y) and throughput and the exit position(x, y) and throughput, depending on the characteristics and layout of the module such as length, width and positioning of the module in the building.

Although the modules are all "model" specialization types in OpenModelica, the interface has to be a connector in order to transmit the data properly.

The annotation of the interface is a black rectangle so it can be easily drag into a model and noticed in the diagram of any module. The Interface code is the following:

```
connector blockInterface
  input Real x(start=0);
  input Real y(start=0);
  input Real throughput(start=1);
 annotation(Icon(coordinateSystem(initialScale = 0.2), graphics
= {Rectangle(origin = {0, -1}, fillPattern = FillPattern.Solid,
extent = {{-100, 101}, {100, -99}})}));
end blockInterface;
```

The methodology to build modules and systems is to load this connector in OpenModelica and drag it into the diagram view the times it is required. Then, The name of the interface should be changed into InterfaceIn and InterfaceOut in order to know the entrance and exit.

The method is open to increase the number of parameters calculated increasing the number of input parameters. The parameters have been set with a start number in order to optimize the calculation and avoid Modelica warnings.

Some module types contain the angle parameter. That parameter is design to set 0º angle in a left to right direction:

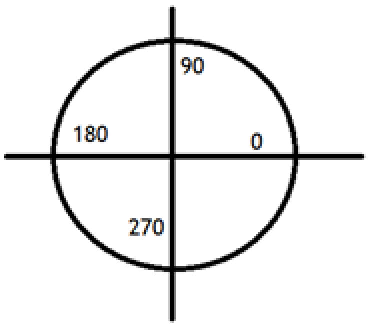


Figure 16: Angle setting in OpenModelica layout

Drawing and colour

The shape of the modules is done according to their usual representation in abstract geometry, and trying to maintain some size relation.

The colour chosen for the modules is depending of their variability or degree of pre-definition. Blue modules are completely defined, that is, the parameters of the module are determined by the user. Those parameters can be x-position, y-position, length, width, angle in the layout, throughput, technical availability, etc. Red Modules are variable ones, so the parameters of the module are calculated in relation to the system and can change, they adapt to the system.

Modules

Prior to starting to create conveyor systems the modules required are programmed at OpenModelica. In order to have a set of different model of the same type of conveyor the selected OpenModelica specialization is Package, so it can be loaded in the library list and select the proper design of the module easily. Each module type have all possible varieties depending on angle, layout or required calculation.

In the following explanation of the constructed modules there are only some code parts or models. For inquiries, the package codes and constructed conveyor systems are in the appendix chapter. The complete codes or text views are in that section.

Source

The source of the system introduces a defined amount of throughput in the system and it sets the starting position (x, y) of the whole system.

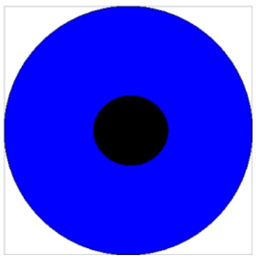


Figure 17: Icon view of a source

The Source has usually a predefined position, as in real warehouses or logistic centres conveyor system there are constraint to the building size. Therefore, It is necessary to define the amount of incoming throughput, the x-position and y-position.

The following code consist on a defined position Source. Notice that it has only one interface, for the outgoing data (interfaceOut):

```
model FixSource
    parameter Real paraThroughPut = 100 "convSource throughPut";
    parameter Real TechnicalAvailability = 100 "availability %";
    parameter Real paraPosX = 0;
    parameter Real paraPosY = 0;
    blockInterface interfaceOut annotation(Placement(visible =
true, transformation(origin = {80, 0}, extent = {{-20, -20},
{20, 20}}, rotation = 0), iconTransformation(origin = {2, 0},
extent = {{-20, -20}, {20, 20}}, rotation = 0)));
  equation
    interfaceOut.x = paraPosX;
    interfaceOut.y = paraPosY;
    interfaceOut.throughput = paraThroughPut *
TechnicalAvailability / 100;
    annotation(Diagram(coordinateSystem(extent = {{-100, -100},
{100, 100}}, preserveAspectRatio = true, initialScale = 0.1,
grid = {2, 2})), Icon(coordinateSystem(initialScale = 0.1),
graphics = {Text(origin = {-17, 2}, lineColor = {255, 255, 255},
extent = {{-40, -20}, {40, 20}}, textString = "Source", fontSize
= 100), Ellipse(origin = {-30, -18}, fillColor = {0, 0, 255},
fillPattern = FillPattern.Solid, extent = {{130, 118}, {-70, -
82}}, endAngle = 360), Ellipse(origin = {5, 4}, fillPattern =
FillPattern.Solid, extent = {{-33, 24}, {27, -32}}, endAngle =
360)}));
  end FixSource;
```

The source package also includes another Source type, VariableSource, without the possibility for the user to choose the position. This variable position is calculated when the other modules of the system are defined, consequently it is not possible to add more variability to the system or the system would crash trying the calculation. Finally, after loading the source the library browser in OpenModelica looks as the next picture:

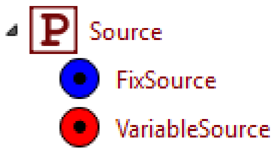


Figure 18: OpenModelica library browser view of sources

Sink

The sink is the last step of a conveyor system, where the units are unloaded and the last position of the system. In abstract geometry is represented as a circle with an inner X.

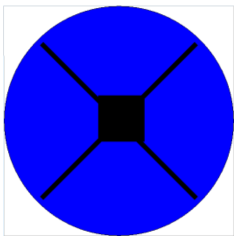


Figure 19: Icon view of a sink

As the Source, the sink can be a fixed sink with a position determined by the user in order to fit the position with the building layout. The other option is the design of a sink in which the position is variable and defined by the layout and parameters of the other modules of the system.

The OpenModelica specialization package consist on a set of models with different variations. The following model is a predefined position sink:

```
model FixedSink
    parameter Real SinkPosX = 100;
    parameter Real SinkPosY = 0;

    blockInterface blockInterface1 annotation(Placement(visible
= true, transformation(origin = {-80, 2}, extent = {{-20, -20},
{20, 20}}, rotation = 0), iconTransformation(origin = {2, 2},
extent = {{-20, -20}, {20, 20}}, rotation = 0)));
  equation
```

```
    blockInterface1.x = SinkPosX;
    blockInterface1.y = SinkPosY;
    annotation(Diagram(coordinateSystem(extent = {{-100, -100},
{100, 100}}, preserveAspectRatio = true, initialScale = 0.1,
grid = {2, 2})), Icon(coordinateSystem(initialScale = 0.1),
graphics = {Ellipse(origin = {-8, 2}, fillColor = {0, 0, 255},
fillPattern = FillPattern.Solid, extent = {{108, 98}, {-92, -
102}}, endAngle = 360), Line(points = {{-66, 66}, {66, -66}},
thickness = 5), Line(points = {{-66, -66}, {66, 66}}, thickness
= 5)}));

end FixedSink;
```

Additionally the Fix and variable position models, there is a multi-input sink model. That sink type can be connected to more than one conveyor and sum the amount of incoming throughput, in contrast with the previous ones that can only be connected to one conveyor module.

Straight conveyor

Straight conveyor modules are the base of any conveyor system. In the methodology they transfer the throughput amount and calculate the position of incoming interface and outgoing interface, as well as calculating the angle within the layout.

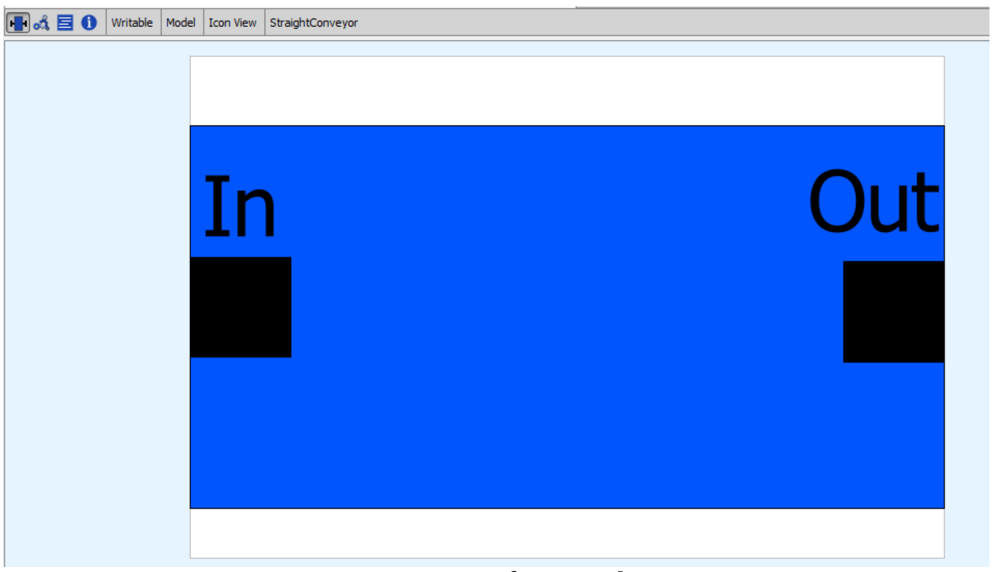


Figure 20: Icon view of a straight conveyor

The straight conveyor package consist of four different models, depending on the requirements when designing a conveyor system:

- Straight_Fix: the length and angle of the conveyor is determined by the user.
- Straight_Variable: The length and angle of the system are variable and depends on the connections of the interfaces.
- Straight_VariableLength: The Angle is determined by the user and the length is variable.
- Straight_VariableAngle: The length is determined by the user and the angle is variable.

Depending on the need at the moment the user is designing a conveyor system, one of those models will be dragged. The most used and valuable modules are the straight conveyor with all defined or the one with all parameters variable. The two types with only the angle or the length fixed may unbalance the amount of variables and equations and can be used only in some cases.

The code of the fixed straight conveyor is the following:

```
model StraightFix

  parameter Real Length_straight=100 "Conveyor length";
  parameter Real angle_straight=0 "Conveyor angle";

  blockInterface blockInterfaceIn annotation(Placement(visible =
true, transformation(origin = {-80, 6}, extent = {{-20, -20},
{20, 20}}, rotation = 0), iconTransformation(origin = {-80, 0},
extent = {{-20, -20}, {20, 20}}, rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible
= true, transformation(origin = {180, 4}, extent = {{-20, -20},
{20, 20}}, rotation = 0), iconTransformation(origin = {180, -2},
extent = {{-20, -20}, {20, 20}}, rotation = 0)));
equation
  blockInterfaceOut.x = blockInterfaceIn.x +
Length_straight*cos(angle_straight*3.141592/180);
  blockInterfaceOut.y = blockInterfaceIn.y +
Length_straight*sin(angle_straight*3.141592/180);
  blockInterfaceOut.throughput = blockInterfaceIn.throughput;

  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(origin = {-2, 0},fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-98, 72}, {202, -
80}}), Text(origin = {-78, 49}, extent = {{-40, -20}, {40, 20}},
textString = "In", fontSize = 100), Text(origin = {172, 43},
extent = {{-40, -20}, {40, 20}}, textString = "Out", fontSize =
100)}, coordinateSystem(extent = {{-100, -100}, {200, 100}})),
Diagram(coordinateSystem(extent = {{-100, -100}, {200, 100}})),
version = "");
end StraightFix;
```

Partial models

In order to solve occurring problems at the variable straight conveyor, especially regarding negative angles, another package named "Partial Models" is created in order to insert any partial model that could be required in the other packages. In this case only one partial model is in the package but it could be increased with the aim of solving any other problem it could appear during the creation of new conveyor modules.

The partial model for the straight conveyor solves all the possibilities that the angle can have. Then, that partial model can be introduced in the variable straight conveyor model using "extent" command.

```
package Partial_Models

partial model straightConveyorBase
    blockInterface blockInterfaceIn
      annotation(
        Placement(
          visible = true, transformation(origin = {-80, 10},
extent = {{-20, -20}, {20, 20}}, rotation = 0),
          iconTransformation( origin = {-80, 0}, extent = {{-20,
-20}, {20, 20}}, rotation = 0 )
        )
      );
    blockInterface blockInterfaceOut
      annotation(
        Placement(
          visible = true, transformation(origin = {80, 12},
extent = {{-20, -20}, {20, 20}}, rotation = 0),
          iconTransformation(origin = {80, 0}, extent = {{-20, -
20}, {20, 20}}, rotation = 0)
        )
      );
    Real conveyorLength(min = 0);
    Real conveyorAngle(start = 0, min = -180, max = 180);

    equation
      blockInterfaceOut.throughput =
blockInterfaceIn.throughput;
    algorithm
      conveyorLength := ((blockInterfaceOut.y -
blockInterfaceIn.y) ^ 2 + (blockInterfaceOut.x -
blockInterfaceIn.x) ^ 2) ^ 0.5;

      if blockInterfaceOut.x < blockInterfaceIn.x then
        if blockInterfaceOut.y < blockInterfaceIn.y then
          conveyorAngle :=
Modelica.SIunits.Conversions.to_deg(atan((blockInterfaceOut.y -
blockInterfaceIn.y) / (blockInterfaceOut.x -
blockInterfaceIn.x))) - 180;
        elseif blockInterfaceOut.y > blockInterfaceIn.y then
```

```
     conveyorAngle :=
Modelica.SIunits.Conversions.to_deg(atan((blockInterfaceOut.y -
blockInterfaceIn.y) / (blockInterfaceOut.x -
blockInterfaceIn.x))) + 180;
    else
     conveyorAngle := -180;
    end if;
   elseif blockInterfaceOut.x > blockInterfaceIn.x then
    conveyorAngle :=
Modelica.SIunits.Conversions.to_deg(atan((blockInterfaceOut.y -
blockInterfaceIn.y) / (blockInterfaceOut.x -
blockInterfaceIn.x)));
   else

    if blockInterfaceOut.y < blockInterfaceIn.y then
     conveyorAngle := -90;
    else
     conveyorAngle := 90;
    end if;
   end if;
 end straightConveyorBase;

end Partial_Models;
```

Curve conveyor

When the design of a conveyor system requires to join two conveyor with a different angle in the layout, a curve conveyor is the solution to avoid problems. In case two straight conveyors would fix together with a considerable angle difference, it could be a dimension problem with the width of each conveyor.

In the methodology 8 types of curve conveyors have been developed, one module for each possible direction. The possible curve options are the following:

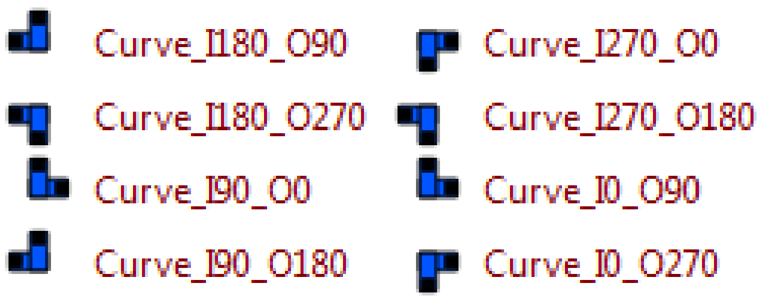


Figure 21: OpenModelica catalogue of curve conveyors

The names of the curves is defined through the angle position of the incoming and outgoing position in the angle standard of the introduction. Each Curve conveyor consist of one incoming and one outgoing interface. The length and width of the system has to be defined by the user, being the length the extent from the incoming interface of the conveyor until the edge, and the width from the edge to the outgoing interface. Therefore, depending on the positioning of the conveyor (notice the upper image) the length or width can change either the x or y axis. The axis is written in the code and will be shown when defining the parameters.

This OpenModelica model is a curve that mover the position positively in x and y axis, after defining the length and curve. The predefined number is one for both units:

```
model Curve_I180_O90

   parameter Real length_curve=1 "Conveyor length (X axis)";
   parameter Real width_curve=1 "Conveyor width (Y axis)";
  // parameter Real angle_curve=0 "Conveyor angle";

    blockInterface blockInterfaceIn annotation(Placement(visible
= true, transformation(origin = {-80,         10}, extent = {{-
20, -20}, {20, 20}}, rotation = 0), iconTransformation(origin =
{-80, 0}, extent = {{-20, -20}, {20, 20}}, rotation = 0)));
    blockInterface blockInterfaceOut
annotation(Placement(visible = true, transformation(origin =
{0,80}, extent = {{-20, -20}, {20, 20}}, rotation = 0),
iconTransformation(origin = {0, 80}, extent = {{-20, -20}, {20,
20}}, rotation = 0)));
  equation
   blockInterfaceOut.x = blockInterfaceIn.x + length_curve;
   blockInterfaceOut.y = blockInterfaceIn.y + width_curve;
   blockInterfaceOut.throughput = blockInterfaceIn.throughput;

   annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-100, -30}, {30, 30}}),
Rectangle(fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-30, -30}, {30, 100}}),
Text(origin = {-82, -47}, extent = {{-40, -20}, {40, 20}},
textString = "In", fontSize = 100), Text(origin = {64, 81},
extent = {{-40, -20}, {40, 20}}, textString = "Out", fontSize =
100)}, coordinateSystem(initialScale = 0.1)));
  end Curve_I180_O90;
```

Branch

Conveyor modules designed to split the throughput in two or more different flows, being the amount of flow in each direction variable and decided by the user. In the methodology two varieties of branches have been designed. The first one is named branch square and it consist of one block or module that splits the flow in different directions. The following directions have been designed.

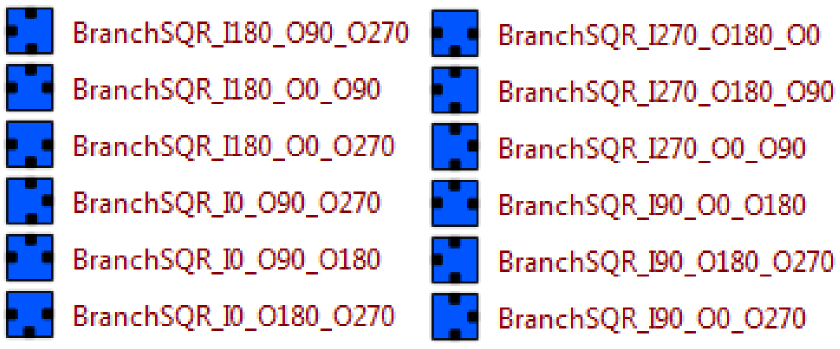

BranchSQR_I180_O90_O270
BranchSQR_I180_O0_O90
BranchSQR_I180_O0_O270
BranchSQR_I0_O90_O270
BranchSQR_I0_O90_O180
BranchSQR_I0_O180_O270
BranchSQR_I270_O180_O0
BranchSQR_I270_O180_O90
BranchSQR_I270_O0_O90
BranchSQR_I90_O0_O180
BranchSQR_I90_O180_O270
BranchSQR_I90_O0_O270

Figure 22: OpenModelica catalogue of Branches

As seen above, there are three types of branch square per direction. One split type where the flow comes and splits left and right, and the others in which the flow continues straight and turns left or right. There are named to technically know the position of the incoming and outgoing interfaces. The first model code of the package is:

```
model BranchSQR_I180_O90_O270
  parameter Real splitProp = 1 "blockInterfaceOut1 /
blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (X axis)";
  parameter Real width_branch = 1 "Conveyor width (Y axis)";

  blockInterface blockInterfaceIn annotation(Placement(visible =
true, transformation(origin = {-60, 4}, extent = {{-10, -10},
{10, 10}}, rotation = 0), iconTransformation(origin = {-80, 0},
extent = {{-20, -20}, {20, 20}}, rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible
= true, transformation(origin = {80, -80}, extent = {{-10, -10},
{10, 10}}, rotation = 0), iconTransformation(origin = {8, -80},
extent = {{-20, -20}, {20, 20}}, rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible
= true, transformation(origin = {80, 80}, extent = {{-10, -10},
{10, 10}}, rotation = 0), iconTransformation(origin = {4, 80},
extent = {{-20, -20}, {20, 20}}, rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x + Length_branch/2;
  blockInterfaceOut1.y = blockInterfaceIn.y + width_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x + Length_branch/2;
  blockInterfaceOut2.y = blockInterfaceIn.y - width_branch / 2;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput
= blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput /
blockInterfaceOut2.throughput;

 annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-100, -100}, {100, 100}}),
Text(origin = {-40, 3}, extent = {{-40, -20}, {40, 20}},
textString = "In", fontSize = 100), Text(origin = {64, 81},
extent = {{-40, -20}, {40, 20}}, textString = "Out1", fontSize =
100), Text(origin = {67, -85}, extent = {{-39, 23}, {33, -15}},
textString = "Out2")}, coordinateSystem(initialScale = 0.1)));
end BranchSQR_I180_O90_O270;
```

The defined parameters of a Branch square module are length, width and split proportion. Split proportion is the division of the throughputs in the outgoing interfaces, so that the percent of outgoing throughput can be defined:

$$SplitProportion = \frac{Throughput(InterfaceOut1)}{Throughput(InterfaceOut2)}$$

Driven parameters of the module are x-position, y-position and throughput.

In order to make the designing of conveyor systems easier, conveyor branch system have been build. Those branch systems contain one incoming and two outgoing interfaces, and to connect them there are three straight conveyors, two curves and one branch square.

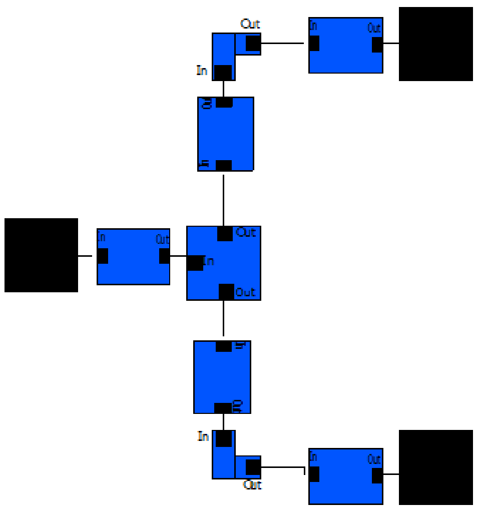


Figure 23: Diagram view of a branch system

The length and width of each conveyor is pre-set as 1 meter, but it can be changed by the user. Similar to the single branch squares, the branch system package contains four different directions.

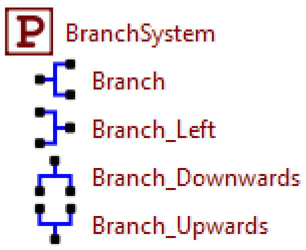

BranchSystem
Branch
Branch_Left
Branch_Downwards
Branch_Upwards

Figure 24: OpenModelica library browser view of sources

Those models are basically simple conveyor systems that can be dragged into another construction. There are drawn to look as a system, using blue colour as there are defined modules.

Merge

Merging works opposite to a branch, but using similar methodology. There are two incoming interfaces and one outgoing interface. The incoming throughput is summed and the outgoing position is calculated regarding the size and the connections of the incoming interfaces.

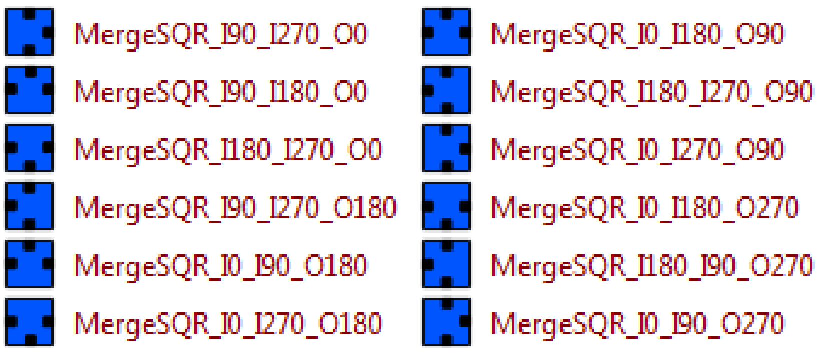

MergeSQR_I90_I270_O0
MergeSQR_I90_I180_O0
MergeSQR_I180_I270_O0
MergeSQR_I90_I270_O180
MergeSQR_I0_I90_O180
MergeSQR_I0_I270_O180
MergeSQR_I0_I180_O90
MergeSQR_I180_I270_O90
MergeSQR_I0_I270_O90
MergeSQR_I0_I180_O270
MergeSQR_I180_I90_O270
MergeSQR_I0_I90_O270

Figure 25: OpenModelica catalogue of merges

Therefore, there are three types of design for each cardinal direction(upwards, downwards, leftwards and rightwards). In the first design both incoming conveyors are at 90º and -90º, and in the other designs one incoming conveyor is straight to the outgoing interface and the other at 90º or -90º depending on the need.
The defined parameters of a Branch square module are length and width. Driven parameters of the module are x-position, y-position and throughput. Here a proportion is not required, there is a sum of throughput. The first model code is:

```
model MergeSQR_I90_I270_O0
parameter Real Length_merge = 1 "Conveyor Merge length (X
Axis)";
```

```
parameter Real width_merge = 1 "Conveyor Merge width (Y axis)";

blockInterface blockInterfaceIn1 annotation(Placement(visible =
true, transformation(origin = {-2, 84}, extent = {{-10, -10},
{10, 10}}, rotation = 0), iconTransformation(origin = {-2, 80},
extent = {{-20, -20}, {20, 20}}, rotation = 0)));
blockInterface blockInterfaceIn2 annotation(Placement(visible =
true, transformation(origin = {-2, -86}, extent = {{-10, -10},
{10, 10}}, rotation = 0), iconTransformation(origin = {-2, -80},
extent = {{-20, -20}, {20, 20}}, rotation = 0)));
blockInterface blockInterfaceOut annotation(Placement(visible =
true, transformation(origin = {72, 2}, extent = {{-10, -10},
{10, 10}}, rotation = 0), iconTransformation(origin = {80, -2},
extent = {{-20, -20}, {20, 20}}, rotation = 0)));
equation
blockInterfaceOut.x = blockInterfaceIn1.x + Length_merge/2;
blockInterfaceOut.y = blockInterfaceIn1.y - width_merge /2;
blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput


annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-99, 110}, {101, -
100}})}));annotation(Icon(graphics = {Rectangle(fillColor = {0,
85, 255}, fillPattern = FillPattern.Solid, extent = {{-100,
100}, {100, -100}}), Text(origin = {72, 35}, extent = {{-26,
21}, {26, -21}}, textString = "Out"), Text(origin = {-54, 85},
extent = {{-20, 17}, {28, -23}}, textString = "In1"),
Text(origin = {-61, -76}, extent = {{-27, 16}, {37, -22}},
textString = "In2")}, coordinateSystem(initialScale = 0.1)));

end MergeSQR_I90_I270_O0;
```

Using those merge squares it is the opportunity to build basic merge system, in order to make the work easier when building conveyor systems. For that reason, a merge system for each cardinal direction is built.

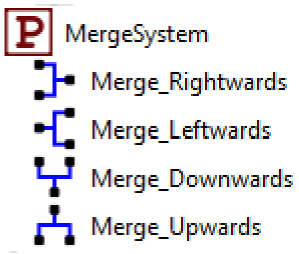

MergeSystem
Merge_Rightwards
Merge_Leftwards
Merge_Downwards
Merge_Upwards

Figure 26: OpenModelica library browser view of Merge Systems

Each Construction contain three interfaces (two incoming and one outgoing), straight conveyors, two curves and one branch square. The following picture is the example of a merge system with a flow direction to the right:

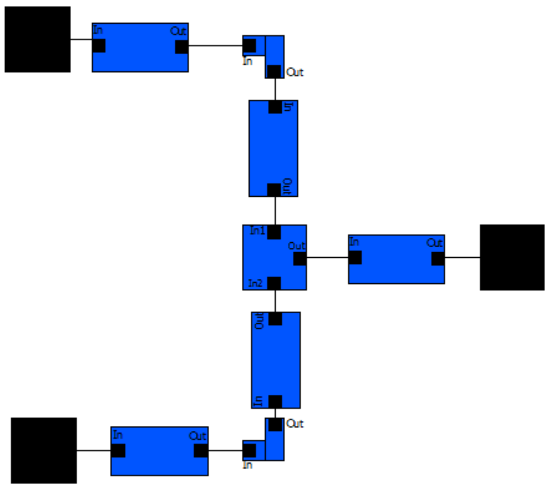


Figure 27: Diagram view of a merge system in OpenModelica

The user can load the merge system package and drag one model to the screen, so complex systems can be constructed. Then the previous system would look as in the icon view, with the design of the *annotation* in OpenModelica:

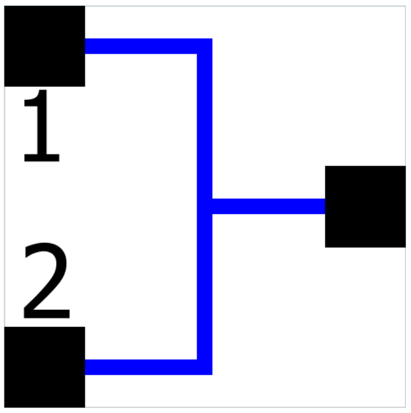


Figure 28: Icon view of a merge in OpenModelica

There are drawn using a blue colour in order to show that the size and angle of the system is defined.

Utilization of the methodology

1. Load the interface and the required module packages.
2. In the diagram view, build a conveyor system dragging models from the library.
3. Position the modules and rotate conveyors in order to have a clear look.
4. Right-click defined conveyors:
    a. Name them in attributes category.
    b. Enter parameters and set the quantity of parameters at parameter category.
5. Connect the modules at diagram view.
6. Note that the number of equations and variables have to be the same.
7. Simulate the system.
8. Check results: parameters of the system's modules (position, throughput, length, angle).

That process can be illustrated using a Business Process Model an Notation, one standardized method to model different types of processes.

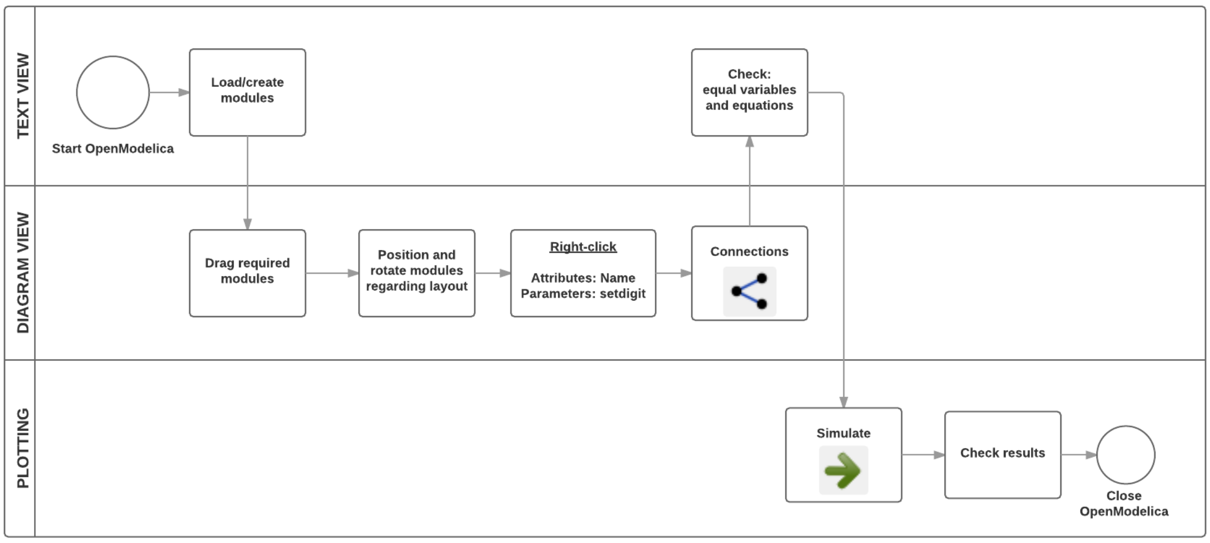


Figure 29: Procedure methodology BPMN

Export data

Once the simulation is run successfully and the resulting data is checked, it can be interesting for the user to export the data from OpenModelica. The intention can be make calculations in other software such as Matlab or get the interface positions and throughput data.

At the upper menu bar there are some drop down menus. Click "Tools" menu and the following option come into view:

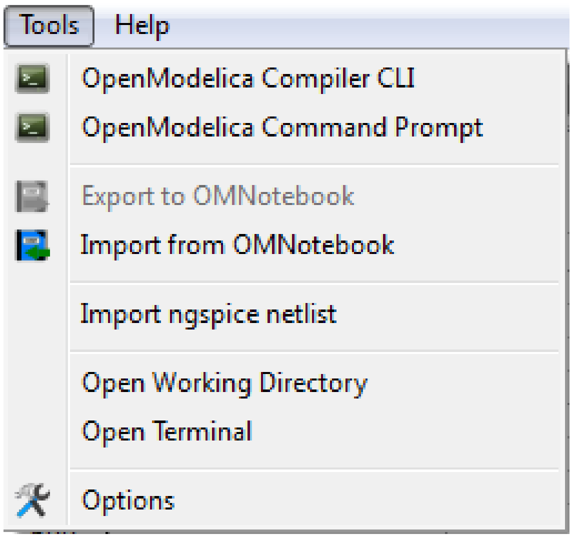


Figure 30: OpenModelica Tools drop down menu

In this menu, the first two buttons open the compiler CLI and the command Prompt. Next, there is the option of importing and exporting OMNotebooks, One OpenModelica feature that shows the name, the diagram view and the text view in one picture file. It can be useful to illustrate the designs of conveyor systems.

In order to have the data files, "Open Working Directory" has to be clicked. Thereby, it is opened the folder containing all the simulation results files and many other files. those files can be exported to other Software type, if the new software is compatible with this type of file.

Examples of conveyor systems

Using the methodology complex conveyor systems can be build, just taking special care of the number of variables and equations to be the same. A system with all the modules of the defined type will not be possible to simulate, as well as other with more variable modules than the correct number.

Standards: source to sink layouts

With the aim of making conveyor system design easier some standard system are build. The procedure is to unite a source and a sink using the different possible layouts conveyor systems can have to join both points.

From point A (source to point B (sink) the way can have L figure, S figure, straight line, etc. Those shapes have also the reverse one. With the library browser view the package contains the following models, which model quantity could be increased if required:

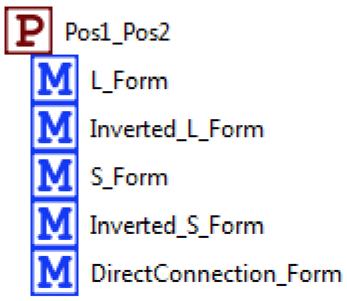


Figure 31: Pos1_Pos2 package library browser view

The L figure layout contains three straight conveyors and two curve conveyors. As the source and the sink position is defined by the user and can change, variable-length straight conveyors are used. Straight conveyors have a determined angle and the length is variable depending on the position of the source and the sink. The inverted L form layout is similar but with different determined angles.

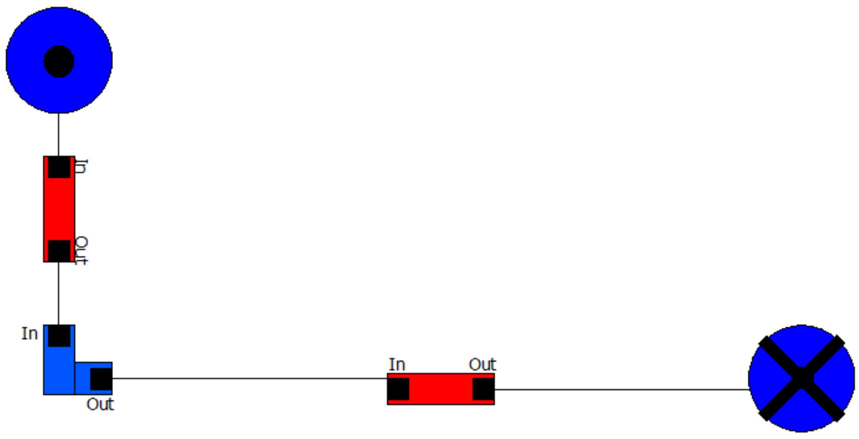


Figure 32: L layout type diagram view

Another common layout is the S figure layout. It consist of three straight conveyors and two curve conveyors. As the source and the sink position is defined by the user and can change, variable-length straight conveyors are used. The conveyor at 90º (Y-axis) is variable and only one on the conveyors at 0º, as the simulation would crash due to being under-determined.

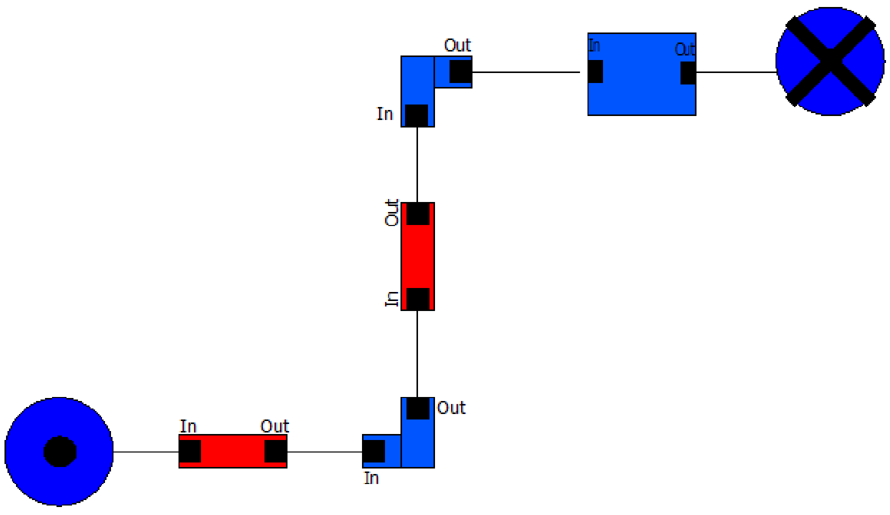


Figure 33: S layout type diagram view

The model with the inverted S layout is similar but the angle of the conveyors is different.
Direct connection through a single straight conveyor is another alternative, although it is not very used alternative in reality due to using common angles in the layout design (0º,

90º, etc). It consist on a variable straight conveyor that will adapt its angle and length depending on the position of the source and sink.

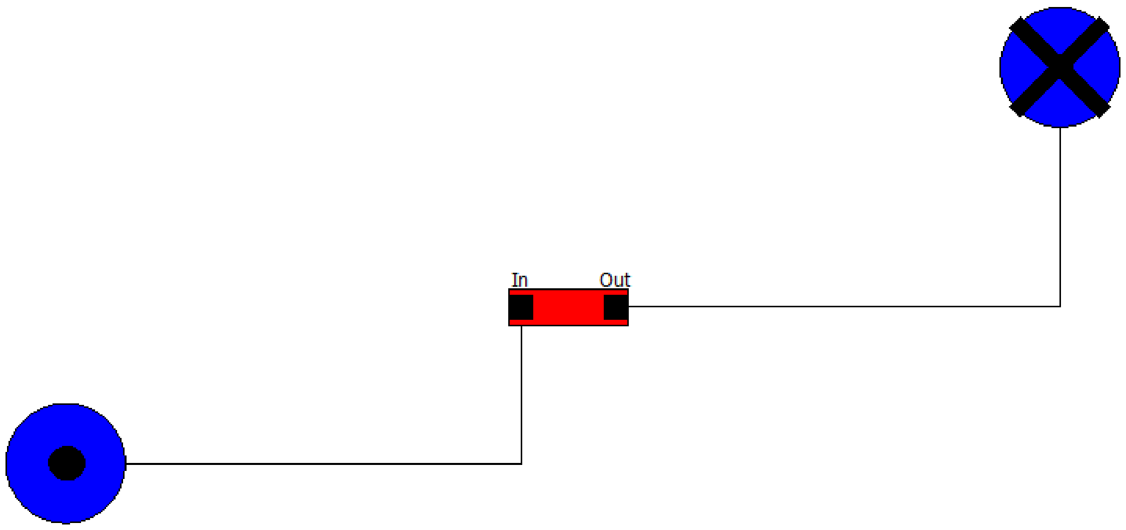


Figure 34: Straight line layout diagram view

For inquiries, the OpenModelica code for the whole package is in the attachments section.

Conveyor system assembly

With regard to showing examples of results using the methodology, in this section some conveyor systems are shown. As it is explained before, the main effort of the user is designing the conveyors with the same amount of variables and equations, as well as setting up the parameters such as angle, length or position properly. The variability of the system is usually just correct using logic. For example, two variable conveyors in a row will be incorrect as the length is undeterminable.

With the aim of trying different modules and their correct performance in simulations, a system with a paralell separation is designed. It contains models of all available packages, as the branch and merge systems contain curves and split or union squares.

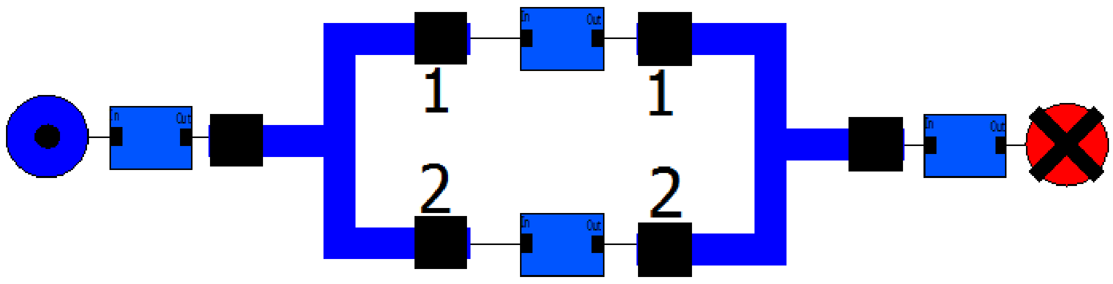
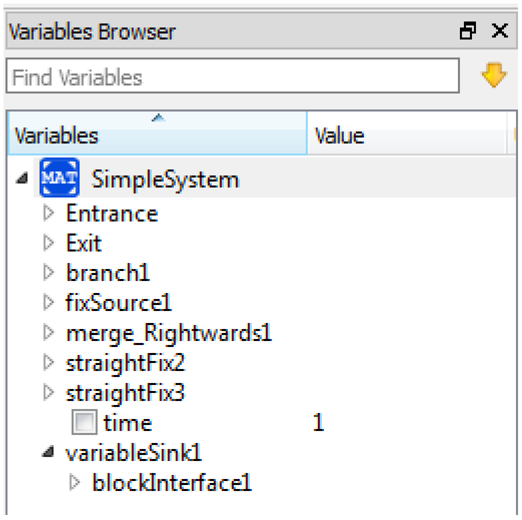


Figure 35: Simple system example diagram view

The source(renamed entrance) is located in the 0 position and has an incoming throughput of 100 units/hour, the straight conveyors have a length of 1 meter and angle 0º and the sink(renamed Exit) is variable and will be positioned depending on the system layout. Simulating the system the results are obtained and shown at variables browser view:



```
Variables Browser
Find Variables
Variables              Value
◢ SimpleSystem
  ▷ Entrance
  ▷ Exit
  ▷ branch1
  ▷ fixSource1
  ▷ merge_Rightwards1
  ▷ straightFix2
  ▷ straightFix3
     ☐ time             1
  ◢ variableSink1
     ▷ blockInterface1
```

Figure 36: Variable browser; results of a simulation

Expanding the modules shows the interfaces and defined parameters. Expanding interfaces shows driven parameters of each interface, such as x and y position and the throughput.

Then, more complex systems can be created, containing more sources or sinks. For instance, the following design of a conveyor system layout with two sources and two sinks. Each source throughput is set to 100 units/hour and the sinks position is variable. The other modules are length and angle defined. Branches split proportion is of 50%.

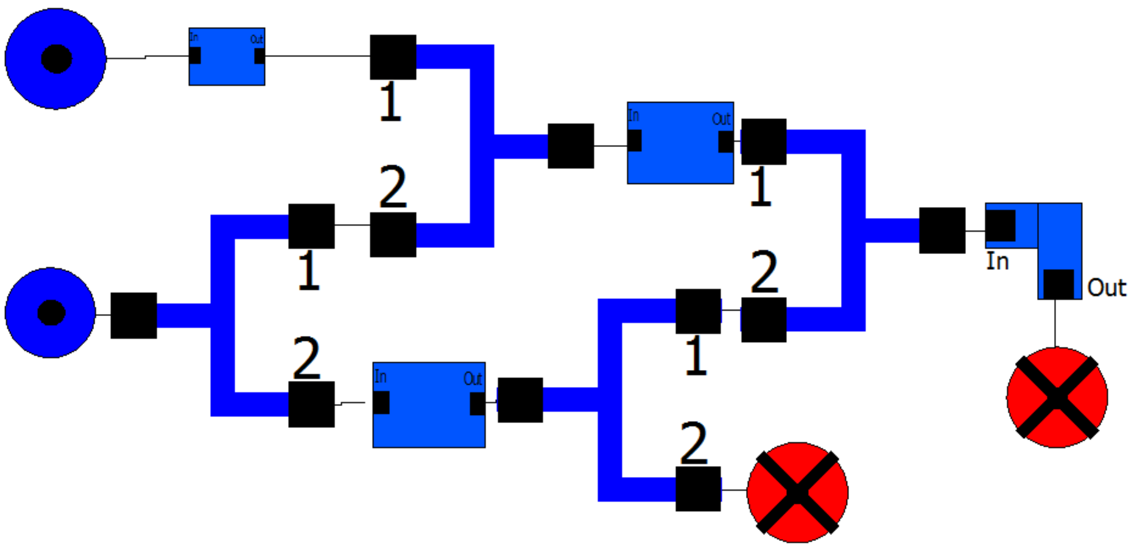
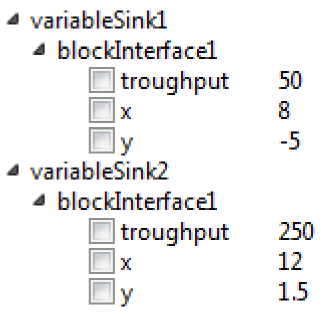


Figure 37: Conveyor system diagram view; two sources and two sinks

The simulation is possible due to a proper variable and equation quantity. The result shows the parameters of each part, and the driven parameter of every interface. Therefore, throughput amount and position of each point inside the system is known. The sinks have a driven position due to the size of the layout and a throughput of 250 and 50 units/hour respectively.



```
◢ variableSink1
  ◢ blockInterface1
      ☐ troughput    50
      ☐ x            8
      ☐ y            -5
◢ variableSink2
  ◢ blockInterface1
      ☐ troughput    250
      ☐ x            12
      ☐ y            1.5
```

Figure 38: Simulation results of the sinks

As seen in the figure above, the sink only includes one interface as it has only incoming throughput and one position.

Warehouse planning

The simulation of a warehouse behaviour is one important feature of the methodology. A warehouse layout can be constructed, its characteristics set and its performance simulated.

In order to have a proper test field, a conveyor system consisting on three fixed sources, three fix sinks and the connection in between is designed.

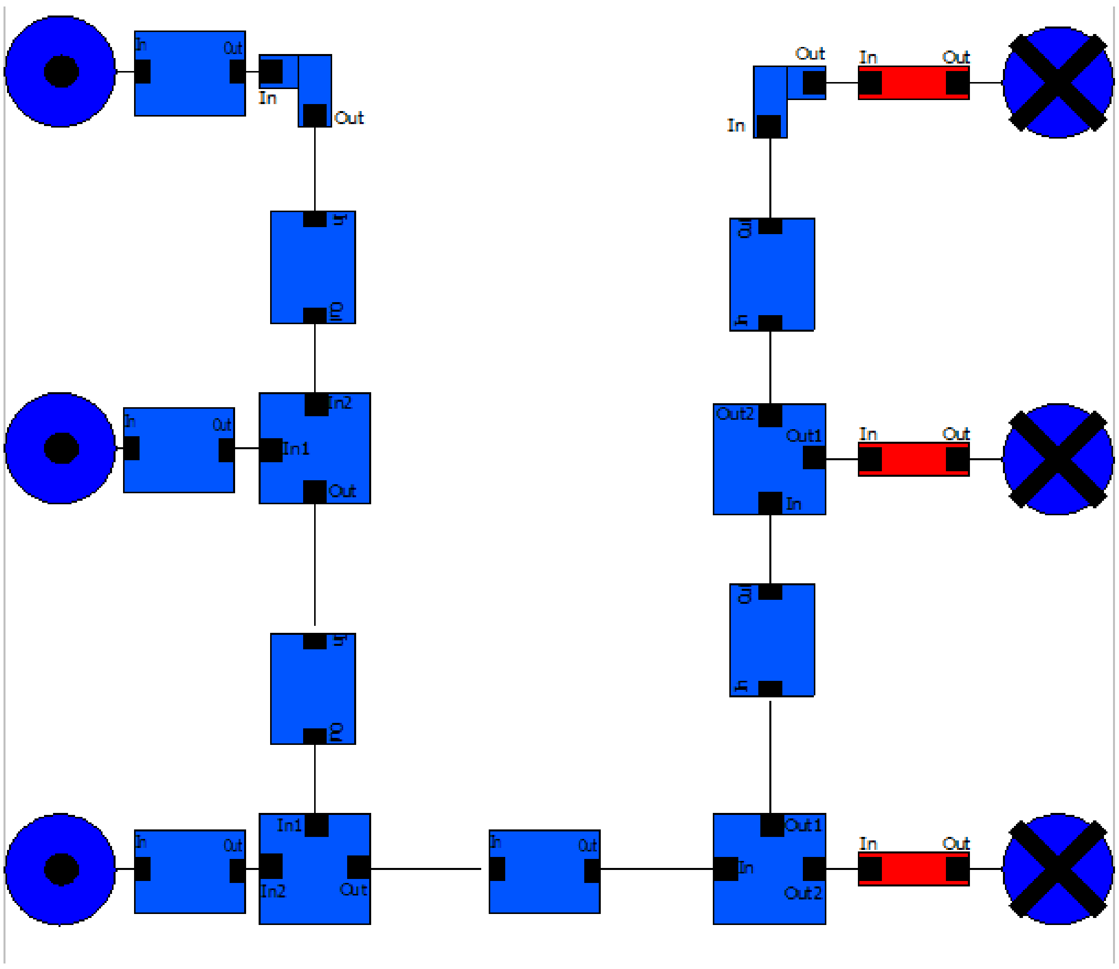


Figure 39: Diagram view of a warehouse

The layout helps to control the parcels flow, from the entrance to the way out. The amount of incoming throughput per source is determined by the user and the outgoing throughput per sink can be adjusted using the split proportion parameter in each branch square.

For example, setting the incoming throughput in each source to 100 units/hour will mean to have a theoretical 300u/hour at the entrance of the first branch. Depending on the amount of throughput wanted in the first sink the split proportion of the branch will be set, noticing the percent of 300 units/hour required in that sink. In a system with a throughput of 100 units/hour and a split proportion of 1 (50% each direction) in the branches, the outgoing amount of throughput would be 150, 75 and 75 units/hour in the first, second and third sinks respectively.

Way definition approach

The previous approach works with a theoretical amount of throughput and calculating the total amount of the incoming flow from the sources. The user cannot decide where the parcels from one source will end. Therefore, with the intention of having the possibility to decide the exit of each incoming parcel the following attempt is done.

In the previous warehouse, a System containing three sources (S1, S2, S3) and three sinks (E1, E2, E3), this throughput management is desired:

- S1 = 150 units/hour
    - 75 units/hour to E1
    - 75 units/hour to E3
- S2 = 100 units/hour
    - 100 units/hour to E2
- S3 = 120 units/hour
    - 40 units/hour to E1
    - 40 units/hour to E2
    - 40 units/hour to E3

For the possible type to solve the problem, **assert** class was taken into consideration. It triggers and prints error message in case of the assertion condition is not fulfilled. The OpenModelica syntax is:

```
assert(condition, message, level = AssertionLevel.error)
```

The idea was to set the conditions written above, with de input and output positions of the parcels, but due to the data structure of the conveyors was build in a way that one module cannot change the data of the previous one. The amount of throughput is summed and it is not possible to determine the throughput of a connected conveyor. In a merge the amount of throughput is accumulated and the origin of the amounts is not defined.

In conclusion, it is not possible to determine the journey of a certain parcel, due to the throughput calculation structure. However, the parcel shipping number can be designed even in complex structured warehouses.

CONCLUSION

During the work of this thesis, a continuous conveyor system layout planning method using OpenModelica has been successfully developed. In order to reach that point, a development of a methodology has been done, after a research of the state of the art of continuous conveyor system.

The influencing attributes in a conveyor system have been found. Then, the influence and relation between those attributes clarified and finally a classification has been made, classifying the attributes as defined parameters or driven parameters. Defined parameters cannot be calculated, they need to be set by the user and are necessary to calculate driven parameters.

The work with OpenModelica resulted in a catalogue of different types of conveyor required for planning conveyor systems. Source, sink, straight and curve conveyor, branch or merge models are available to load and use in creating conveyor systems. Every group of conveyor type have different characteristics models to be used depending on the need. The same type of conveyor can be selected with fixed length, position or angle, as well as with those parameters being variable. Therefore, the planning can be done even if some parts are unknown.

With the use of the Modelica connector specialization to create interfaces, the connections between parts is possible. Required conveyor parts are dragged, the parameters set and connected The position and throughput information is translated trough the system. The data of each inner point and ongoing throughput is available when simulating the systems.

In conclusion, following the methodology explained in this work, conveyor systems can be designed and the layout and throughput be determined.

APPENDIX

The attachment contain OpenModelica codes that are used for the construction of conveyor systems.

Interface

the technique used to unite different conveyors is via interfaces:

```
connector blockInterface
  input Real x(start=0);
  input Real y(start=0);
  input Real throughput(start=1);
 annotation(Icon(coordinateSystem(initialScale = 0.2), graphics =
{Rectangle(origin = {0, -1}, fillPattern = FillPattern.Solid, extent = {{-100,
101}, {100, -99}})}));
end blockInterface;
```

Source

```
package Source

  model FixSource
    parameter Real paraThroughPut = 100 "convSource throughPut";
    parameter Real TechnicalAvailability = 100 "availability %";
    parameter Real paraPosX = 0;
    parameter Real paraPosY = 0;
    blockInterface interfaceOut annotation(Placement(visible = true,
transformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {2, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  equation
    interfaceOut.x = paraPosX;
    interfaceOut.y = paraPosY;
    interfaceOut.throughput = paraThroughPut * TechnicalAvailability / 100;
    annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
Icon(coordinateSystem(initialScale = 0.1), graphics = {Text(origin = {-17, 2},
lineColor = {255, 255, 255}, extent = {{-40, -20}, {40, 20}}, textString =
"Source", fontSize = 100), Ellipse(origin = {-30, -18}, fillColor = {0, 0,
255}, fillPattern = FillPattern.Solid, extent = {{130, 118}, {-70, -82}},
endAngle = 360), Ellipse(origin = {5, 4}, fillPattern = FillPattern.Solid,
extent = {{-33, 24}, {27, -32}}, endAngle = 360)}));
  end FixSource;

  model VariableSource
    parameter Real paraThroughPut = 100 "convSource throughPut";
    parameter Real TechnicalAvailability = 100 "availability %";
    blockInterface interfaceOut annotation(Placement(visible = true,
transformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {2, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  equation
    interfaceOut.throughput = paraThroughPut * TechnicalAvailability / 100;
    annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
Icon(coordinateSystem(initialScale = 0.1), graphics = {Text(origin = {-17, 2},
lineColor = {255, 255, 255}, extent = {{-40, -20}, {40, 20}}, textString =
"Source", fontSize = 100), Ellipse(origin = {-7, -10}, fillColor = {255, 0,
0}, fillPattern = FillPattern.Solid, extent = {{107, 110}, {-93, -90}},
endAngle = 360), Ellipse(origin = {3, 1}, fillPattern = FillPattern.Solid,
extent = {{-29, 29}, {27, -27}}, endAngle = 360)}));
  end VariableSource;
end Source;
```

Sink

```
package Sink
  model FixedSink
    parameter Real SinkPosX = 100;
    parameter Real SinkPosY = 0;

    blockInterface blockInterface1 annotation(Placement(visible = true,
transformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {2, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  equation
    blockInterface1.x = SinkPosX;
    blockInterface1.y = SinkPosY;
    annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
Icon(coordinateSystem(initialScale = 0.1), graphics = {Ellipse(origin = {-8,
2}, fillColor = {0, 0, 255}, fillPattern = FillPattern.Solid, extent = {{108,
98}, {-92, -102}}, endAngle = 360), Line(points = {{-66, 66}, {66, -66}},
thickness = 5), Line(points = {{-66, -66}, {66, 66}}, thickness = 5)}));
  end FixedSink;

  model VariableSink
    blockInterface blockInterface1 annotation(Placement(visible = true,
transformation(origin = {-80, -2}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {0, -2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
Icon(coordinateSystem(initialScale = 0.1), graphics = {Ellipse(origin = {-8,
2}, fillColor = {255, 0, 0}, fillPattern = FillPattern.Solid, extent = {{108,
98}, {-92, -102}}, endAngle = 360), Line(points = {{-66, 66}, {66, -66}},
thickness = 5), Line(points = {{-66, -66}, {66, 66}}, thickness = 5)}));
  end VariableSink;
end Sink;
```

Straight conveyor

```
package StraightConveyor

  //Straight conveyor with the length and angle defined by the user

model StraightFix

  parameter Real Length_straight=100 "Conveyor length";
  parameter Real angle_straight=0 "Conveyor angle";

  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-80, 6}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {180, 4}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {180, -2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut.x = blockInterfaceIn.x +
Length_straight*cos(angle_straight*3.141592/180);
  blockInterfaceOut.y = blockInterfaceIn.y +
Length_straight*sin(angle_straight*3.141592/180);
  blockInterfaceOut.throughput = blockInterfaceIn.throughput;

  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(origin = {-2, 0},fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-98, 72}, {202, -80}}), Text(origin = {-78, 49},
extent = {{-40, -20}, {40, 20}}, textString = "In", fontSize = 100),
Text(origin = {172, 43}, extent = {{-40, -20}, {40, 20}}, textString = "Out",
fontSize = 100)}, coordinateSystem(extent = {{-100, -100}, {200, 100}})),
Diagram(coordinateSystem(extent = {{-100, -100}, {200, 100}})), version = "");
end StraightFix;


    //Straight conveyor With defined angle and variable length

model Straight_VariableLength

 Real Length_straight(min=0);
 parameter  Real angle_straight=0 ;

  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-80, 10}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {80, 12}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));


equation

  blockInterfaceOut.throughput = blockInterfaceIn.throughput;
  tan(angle_straight*3.141592/180)=(blockInterfaceOut.y-
blockInterfaceIn.y)/(blockInterfaceOut.x-blockInterfaceIn.x);
  cos(angle_straight*3.141592/180)=(blockInterfaceOut.x-
blockInterfaceIn.x)/Length_straight;

  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {255, 0, 0}, fillPattern = FillPattern.Solid, extent =
{{-100, 30}, {100, -30}}), Text(origin = {-80, 47}, extent = {{-40, -20}, {40,
20}}, textString = "In", fontSize = 100), Text(origin = {80, 47}, extent = {{-
40, -20}, {40, 20}}, textString = "Out", fontSize = 100)},
coordinateSystem(initialScale = 0.1)));

end Straight_VariableLength;


    //Straight conveyor With defined length and variable angle

model Straight_VariableAngle

parameter Real Length_straight=10 "Conveyor length";
 Real angle_straight(start=-90) ;

  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-80, 10}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {80, 12}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));


equation

  blockInterfaceOut.throughput = blockInterfaceIn.throughput;
  tan(angle_straight*3.141592/180)=(blockInterfaceOut.y-
blockInterfaceIn.y)/(blockInterfaceOut.x-blockInterfaceIn.x);
  cos(angle_straight*3.141592/180)=(blockInterfaceOut.x-
blockInterfaceIn.x)/Length_straight;

  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {255, 0, 0}, fillPattern = FillPattern.Solid, extent =
{{-100, 30}, {100, -30}}), Text(origin = {-80, 47}, extent = {{-40, -20}, {40,
20}}, textString = "In", fontSize = 100), Text(origin = {80, 47}, extent = {{-
40, -20}, {40, 20}}, textString = "Out", fontSize = 100)},
coordinateSystem(initialScale = 0.1)));

end Straight_VariableAngle;



  // Straight conveyor with variable length and angle parameters

  model straightConveyor_Variable
    extends Partial_Models.straightConveyorBase;



    annotation(
      uses(Modelica(version = "3.2.1")),
      Icon(
        graphics = {
          Rectangle(fillColor = {255, 0, 0}, fillPattern = FillPattern.Solid,
extent = {{-100, 30}, {100, -30}}),
          Text(origin = {-80, 47}, extent = {{-40, -20}, {40, 20}}, textString
= "In", fontSize = 100),
          Text(origin = {80, 47}, extent = {{-40, -20}, {40, 20}}, textString
= "Out", fontSize = 100)},
          coordinateSystem(initialScale = 0.1)));
  end straightConveyor_Variable;

end StraightConveyor;
```

<u>Partial models</u>

```
package Partial_Models

partial model straightConveyorBase
    blockInterface blockInterfaceIn
      annotation(
        Placement(
          visible = true, transformation(origin = {-80, 10}, extent = {{-20, -
20}, {20, 20}}, rotation = 0),
          iconTransformation( origin = {-80, 0}, extent = {{-20, -20}, {20,
20}}, rotation = 0 )
        )
      );
    blockInterface blockInterfaceOut
      annotation(
        Placement(
          visible = true, transformation(origin = {80, 12}, extent = {{-20, -
20}, {20, 20}}, rotation = 0),
          iconTransformation(origin = {80, 0}, extent = {{-20, -20}, {20,
20}}, rotation = 0)
        )
      );
    Real conveyorLength(min = 0);
    Real conveyorAngle(start = 0, min = -180, max = 180);

    equation
      blockInterfaceOut.throughput = blockInterfaceIn.throughput;
    algorithm
      conveyorLength := ((blockInterfaceOut.y - blockInterfaceIn.y) ^ 2 +
(blockInterfaceOut.x - blockInterfaceIn.x) ^ 2) ^ 0.5;

      if blockInterfaceOut.x < blockInterfaceIn.x then
        if blockInterfaceOut.y < blockInterfaceIn.y then
          conveyorAngle :=
Modelica.SIunits.Conversions.to_deg(atan((blockInterfaceOut.y -
blockInterfaceIn.y) / (blockInterfaceOut.x - blockInterfaceIn.x))) - 180;
        elseif blockInterfaceOut.y > blockInterfaceIn.y then
```

```
          conveyorAngle :=
Modelica.SIunits.Conversions.to_deg(atan((blockInterfaceOut.y -
blockInterfaceIn.y) / (blockInterfaceOut.x - blockInterfaceIn.x))) + 180;
        else
          conveyorAngle := -180;
        end if;
      elseif blockInterfaceOut.x > blockInterfaceIn.x then
        conveyorAngle :=
Modelica.SIunits.Conversions.to_deg(atan((blockInterfaceOut.y -
blockInterfaceIn.y) / (blockInterfaceOut.x - blockInterfaceIn.x)));
      else

        if blockInterfaceOut.y < blockInterfaceIn.y then
          conveyorAngle := -90;
        else
          conveyorAngle := 90;
        end if;
      end if;
  end straightConveyorBase;
end Partial_Models;
```

<u>Curve conveyor</u>

```
package CurveConveyor

 model Curve_I180_O90

   parameter Real length_curve=1 "Conveyor length (X axis)";
   parameter Real width_curve=1 "Conveyor width (Y axis)";
  // parameter Real angle_curve=0 "Conveyor angle";

    blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-80,        10}, extent = {{-20, -20}, {20, 20}},
rotation = 0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20},
{20, 20}}, rotation = 0)));
    blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {0,80}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {0, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  equation
   blockInterfaceOut.x = blockInterfaceIn.x + length_curve;
   blockInterfaceOut.y = blockInterfaceIn.y + width_curve;
   blockInterfaceOut.throughput = blockInterfaceIn.throughput;

   annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -30}, {30, 30}}), Rectangle(fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-30, -30}, {30, 100}}), Text(origin = {-82, -
47}, extent = {{-40, -20}, {40, 20}}, textString = "In", fontSize = 100),
Text(origin = {64, 81}, extent = {{-40, -20}, {40, 20}}, textString = "Out",
fontSize = 100)}, coordinateSystem(initialScale = 0.1)));
  end Curve_I180_O90;

  model Curve_I180_O270

    parameter Real length_curve=1 "Conveyor length (X axis)";
    parameter Real width_curve=1 "Conveyor width (Y axis)";
   // parameter Real angle_curve=0 "Conveyor angle";

  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-80, 10}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {-2, -80}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {-2, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  equation
   blockInterfaceOut.x = blockInterfaceIn.x + length_curve;
   blockInterfaceOut.y = blockInterfaceIn.y - width_curve;
   blockInterfaceOut.throughput = blockInterfaceIn.throughput;

  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -30}, {30, 30}}), Rectangle(origin = {0, -70},fillColor = {0, 85,
255}, fillPattern = FillPattern.Solid, extent = {{-30, -30}, {30, 100}}),
Text(origin = {-82, -47}, extent = {{-40, -20}, {40, 20}}, textString = "In",
fontSize = 100), Text(origin = {64, -83}, extent = {{-40, -20}, {40, 20}},
textString = "Out", fontSize = 100)}, coordinateSystem(initialScale = 0.1)));
  end Curve_I180_O270;

  model Curve_I90_O0
  parameter Real length_curve = 1 "Conveyor length (Y axis)";
  parameter Real width_curve = 1 "Conveyor width (X axis)";
 // parameter Real angle_curve = 0 "Conveyor angle";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-2, 80}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {0, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {80, -14}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut.y = blockInterfaceIn.y - length_curve;
  blockInterfaceOut.x = blockInterfaceIn.x + width_curve;
  blockInterfaceOut.throughput = blockInterfaceIn.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(origin = {70, 0},fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-100, -30}, {30, 30}}), Rectangle(fillColor =
{0, 85, 255}, fillPattern = FillPattern.Solid, extent = {{-30, -30}, {30,
100}}), Text(origin = {-52, 85}, extent = {{-40, -20}, {40, 20}}, textString =
"In", fontSize = 100), Text(origin = {80, -47}, extent = {{-40, -20}, {40,
20}}, textString = "Out", fontSize = 100)}, coordinateSystem(initialScale =
0.1)));
end Curve_I90_O0;

model Curve_I90_O180

parameter Real length_curve = 1 "Conveyor length (Y axis)";
parameter Real width_curve = 1 "Conveyor width (X axis)";
//parameter Real angle_curve = 0 "Conveyor angle";
blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-2, 80}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {0, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {-80, -12}, extent = {{-20, -20}, {20, 20}}, rotation
= 0), iconTransformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
blockInterfaceOut.y = blockInterfaceIn.y - length_curve;
blockInterfaceOut.x = blockInterfaceIn.x - width_curve;
blockInterfaceOut.throughput = blockInterfaceIn.throughput;
annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -30}, {30, 30}}), Rectangle(fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-30, -30}, {30, 100}}), Text(origin = {-52, 85},
extent = {{-40, -20}, {40, 20}}, textString = "In", fontSize = 100),
Text(origin = {-76, -55}, extent = {{-40, -20}, {40, 20}}, textString = "Out",
fontSize = 100)}, coordinateSystem(initialScale = 0.1)));
```

```
end Curve_I90_O180;

model Curve_I270_O0

  parameter Real length_curve = 1 "Conveyor length (Y axis)";
  parameter Real width_curve = 1 "Conveyor width (X axis)";
  //parameter Real angle_curve = 0 "Conveyor angle";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-8, -80}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {-2, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {80, 10}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut.y = blockInterfaceIn.y + length_curve;
  blockInterfaceOut.x = blockInterfaceIn.x + width_curve;
  blockInterfaceOut.throughput = blockInterfaceIn.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(origin = {70, 0},fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-100, -30}, {30, 30}}), Rectangle(origin = {0, -
70},fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent = {{-30,
-30}, {30, 100}}), Text(origin = {-58, -77}, extent = {{-40, -20}, {40, 20}},
textString = "In", fontSize = 100), Text(origin = {76, 51}, extent = {{-40, -
20}, {40, 20}}, textString = "Out", fontSize = 100)},
coordinateSystem(initialScale = 0.1)));
end Curve_I270_O0;

model Curve_I270_O180

  parameter Real length_curve = 1 "Conveyor length (Y axis)";
  parameter Real width_curve = 1 "Conveyor width (X axis)";
  //parameter Real angle_curve = 0 "Conveyor angle";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-4, -80}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {-80, 4}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut.y = blockInterfaceIn.y + length_curve;
  blockInterfaceOut.x = blockInterfaceIn.x - width_curve;
  blockInterfaceOut.throughput = blockInterfaceIn.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -30}, {30, 30}}), Rectangle(origin = {0, -70},fillColor = {0, 85,
255}, fillPattern = FillPattern.Solid, extent = {{-30, -30}, {30, 100}}),
Text(origin = {56, -79}, extent = {{-40, -20}, {40, 20}}, textString = "In",
fontSize = 100), Text(origin = {-68, 47}, extent = {{-40, -20}, {40, 20}},
textString = "Out", fontSize = 100)}, coordinateSystem(initialScale = 0.1)));
end Curve_I270_O180;

model Curve_I0_O90

   parameter Real length_curve=1 "Conveyor length (X axis)";
   parameter Real width_curve=1 "Conveyor width (Y axis)";
  // parameter Real angle_curve=0 "Conveyor angle";

    blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {80, 2}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {0,80}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {0, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  equation
   blockInterfaceOut.x = blockInterfaceIn.x - length_curve;
   blockInterfaceOut.y = blockInterfaceIn.y + width_curve;
   blockInterfaceOut.throughput = blockInterfaceIn.throughput;

   annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(origin = {70, 0},fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-100, -30}, {30, 30}}), Rectangle(fillColor =
{0, 85, 255}, fillPattern = FillPattern.Solid, extent = {{-30, -30}, {30,
100}}), Text(origin = {84, -49}, extent = {{-40, -20}, {40, 20}}, textString =
"In", fontSize = 100), Text(origin = {-64, 81}, extent = {{-40, -20}, {40,
20}}, textString = "Out", fontSize = 100)}, coordinateSystem(initialScale =
0.1)));
end Curve_I0_O90;

model Curve_I0_O270

   parameter Real length_curve=1 "Conveyor length (X axis)";
   parameter Real width_curve=1 "Conveyor width (Y axis)";
   //parameter Real angle_curve=0 "Conveyor angle";

    blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {80, 2}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  equation
   blockInterfaceOut.x = blockInterfaceIn.x - length_curve;
   blockInterfaceOut.y = blockInterfaceIn.y - width_curve;
   blockInterfaceOut.throughput = blockInterfaceIn.throughput;

   annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(origin = {70, 0},fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-100, -30}, {30, 30}}), Rectangle(origin = {0, -
70},fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent = {{-30,
-30}, {30, 100}}), Text(origin = {84, -49}, extent = {{-40, -20}, {40, 20}},
textString = "In", fontSize = 100), Text(origin = {-66, -75}, extent = {{-40,
-20}, {40, 20}}, textString = "Out", fontSize = 100)},
coordinateSystem(initialScale = 0.1)));

end Curve_I0_O270;
end CurveConveyor;
```

Branch Square

```
package BranchSquare

//Branch in direction +y and -y

model BranchSQR_I180_O90_O270
  parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (X axis)";
  parameter Real width_branch = 1 "Conveyor width (Y axis)";

  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-60, 4}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {80, -80}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {8, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {80, 80}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {4, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x + Length_branch/2;
  blockInterfaceOut1.y = blockInterfaceIn.y + width_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x + Length_branch/2;
  blockInterfaceOut2.y = blockInterfaceIn.y - width_branch / 2;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;

 annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {-40, 3}, extent = {{-40, -20},
{40, 20}}, textString = "In", fontSize = 100), Text(origin = {64, 81}, extent
= {{-40, -20}, {40, 20}}, textString = "Out1", fontSize = 100), Text(origin =
{67, -85}, extent = {{-39, 23}, {33, -15}}, textString = "Out2")},
coordinateSystem(initialScale = 0.1)));
end BranchSQR_I180_O90_O270;


// Branch to straight (x) and +y direction

model BranchSQR_I180_O0_O90
  parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (X axis)";
  parameter Real width_branch = 1 "Conveyor width (Y axis)";

  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-90, 4}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {90, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-8, 90}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {4, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x + Length_branch/2 ;
  blockInterfaceOut1.y = blockInterfaceIn.y + width_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x + Length_branch ;
  blockInterfaceOut2.y = blockInterfaceIn.y;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;

 annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {-40, 3}, extent = {{-40, -20},
{40, 20}}, textString = "In", fontSize = 100), Text(origin = {64, 81}, extent
= {{-40, -20}, {40, 20}}, textString = "Out1", fontSize = 100), Text(origin =
{67, -45}, extent = {{-39, 23}, {33, -15}}, textString = "Out2")},
coordinateSystem(initialScale = 0.1)));
end BranchSQR_I180_O0_O90;


// Branch to straight (x) and -y direction

model BranchSQR_I180_O0_O270
  parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (X axis)";
  parameter Real width_branch = 1 "Conveyor width (Y axis)";

  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-90, 4}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {4, -90}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {4, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {90, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, -2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x + Length_branch;
  blockInterfaceOut1.y = blockInterfaceIn.y;
  blockInterfaceOut2.x = blockInterfaceIn.x + Length_branch/2;
  blockInterfaceOut2.y = blockInterfaceIn.y - width_branch / 2;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {-40, 3}, extent = {{-40, -20},
{40, 20}}, textString = "In", fontSize = 100), Text(origin = {62, 39}, extent
= {{-40, -20}, {40, 20}}, textString = "Out1", fontSize = 100), Text(origin =
{63, -85}, extent = {{-39, 23}, {33, -15}}, textString = "Out2")},
coordinateSystem(initialScale = 0.1)));
end BranchSQR_I180_O0_O270;


// Branch to backwards input(-x) and +y and -y direction split

model BranchSQR_I0_O90_O270

  parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (X axis)";
  parameter Real width_branch = 1 "Conveyor width (Y axis)";

  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {84, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {2, -80}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {8, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-2, 82}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {4, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x - Length_branch/2;
  blockInterfaceOut1.y = blockInterfaceIn.y + width_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x - Length_branch/2;
  blockInterfaceOut2.y = blockInterfaceIn.y - width_branch / 2;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;

 annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {38, 3}, extent = {{-40, -20}, {40,
20}}, textString = "In", fontSize = 100), Text(origin = {64, 81}, extent = {{-
40, -20}, {40, 20}}, textString = "Out1", fontSize = 100), Text(origin = {67,
-83}, extent = {{-39, 23}, {33, -15}}, textString = "Out2")},
coordinateSystem(initialScale = 0.1)));
```

```
// Branch to backwards input(-x) and +y and straight(-x) direction split

end BranchSQR_I0_O90_O270;

model BranchSQR_I0_O90_O180
  parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (X axis)";
  parameter Real width_branch = 1 "Conveyor width (Y axis)";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {84, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {2, -80}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-2, 82}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {4, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x - Length_branch / 2;
  blockInterfaceOut1.y = blockInterfaceIn.y + width_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x - Length_branch;
  blockInterfaceOut2.y = blockInterfaceIn.y;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {38, 3}, extent = {{-40, -20}, {40,
20}}, textString = "In", fontSize = 100), Text(origin = {64, 81}, extent = {{-
40, -20}, {40, 20}}, textString = "Out1", fontSize = 100), Text(origin = {-61,
-39}, extent = {{-39, 23}, {33, -15}}, textString = "Out2")},
coordinateSystem(initialScale = 0.1)));
end BranchSQR_I0_O90_O180;


// Branch to backwards input(-x) and +y and -y direction split

model BranchSQR_I0_O180_O270

  parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (X axis)";
  parameter Real width_branch = 1 "Conveyor width (Y axis)";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {84, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {2, -80}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-2,-80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-84, 0}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x - Length_branch;
  blockInterfaceOut1.y = blockInterfaceIn.y;
  blockInterfaceOut2.x = blockInterfaceIn.x - Length_branch/2;
  blockInterfaceOut2.y = blockInterfaceIn.y - width_branch / 2;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {38, 3}, extent = {{-40, -20}, {40,
20}}, textString = "In", fontSize = 100), Text(origin = {-64, -39}, extent =
{{-40, -20}, {40, 20}}, textString = "Out1", fontSize = 100), Text(origin =
{59, -85}, extent = {{-39, 23}, {33, -15}}, textString = "Out2")},
coordinateSystem(initialScale = 0.1)));

end BranchSQR_I0_O180_O270;


// Branch with upwards input(+y) and +x and -x direction split

model BranchSQR_I270_O180_O0

 parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (Y axis)";
  parameter Real width_branch = 1 "Conveyor width (X axis)";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {0, -84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {82,2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, -2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-80, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x - width_branch / 2;
  blockInterfaceOut1.y = blockInterfaceIn.y + Length_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x + Length_branch/2;
  blockInterfaceOut2.y = blockInterfaceIn.y + Length_branch / 2 ;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {46, -81}, extent = {{-40, -20},
{40, 20}}, textString = "In", fontSize = 100), Text(origin = {-60, 41}, extent
= {{-40, -20}, {40, 20}}, textString = "Out1", fontSize = 100), Text(origin =
{67, 35}, extent = {{-39, 23}, {33, -15}}, textString = "Out2")},
coordinateSystem(initialScale = 0.1)));
end BranchSQR_I270_O180_O0;


// Branch with upwards input(+y) and +y and -x direction split

model BranchSQR_I270_O180_O90

 parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (Y axis)";
  parameter Real width_branch = 1 "Conveyor width (X axis)";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {0, -84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {-6,82}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0,80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-80, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
```

```
  blockInterfaceOut1.x = blockInterfaceIn.x - width_branch / 2;
  blockInterfaceOut1.y = blockInterfaceIn.y + Length_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x;
  blockInterfaceOut2.y = blockInterfaceIn.y + Length_branch  ;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {46, -81}, extent = {{-40, -20},
{40, 20}}, textString = "In", fontSize = 100), Text(origin = {-62, 41}, extent
= {{-40, -20}, {40, 20}}, textString = "Out1", fontSize = 100), Text(origin =
{61,75}, extent = {{-39, 23}, {33, -15}}, textString = "Out2")},
coordinateSystem(initialScale = 0.1)));
end BranchSQR_I270_O180_O90;


// Branch with upwards input(+y) and +y and +x direction split

model BranchSQR_I270_O0_O90
 parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (Y axis)";
  parameter Real width_branch = 1 "Conveyor width (X axis)";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {0, -84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {-6,82}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0,80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {84, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 4}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x + width_branch / 2;
  blockInterfaceOut1.y = blockInterfaceIn.y + Length_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x;
  blockInterfaceOut2.y = blockInterfaceIn.y + Length_branch  ;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {46, -81}, extent = {{-40, -20},
{40, 20}}, textString = "In", fontSize = 100), Text(origin = {66, 43}, extent
= {{-40, -20}, {40, 20}}, textString = "Out1", fontSize = 100), Text(origin =
{-55, 79}, extent = {{-39, 23}, {33, -15}}, textString = "Out2")},
coordinateSystem(initialScale = 0.1)));
end BranchSQR_I270_O0_O90;


// Branch with downwards input(-y) and +x and -x direction split
model BranchSQR_I90_O0_O180

  parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (Y axis)";
  parameter Real width_branch = 1 "Conveyor width (X axis)";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-2, 84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-2, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {82,2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, -2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
```

```
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-80, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x - width_branch / 2;
  blockInterfaceOut1.y = blockInterfaceIn.y - Length_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x + width_branch/2;
  blockInterfaceOut2.y = blockInterfaceIn.y - Length_branch / 2 ;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {38, 81}, extent = {{-40, -20},
{40, 20}}, textString = "In", fontSize = 100), Text(origin = {-62, -37},
extent = {{-40, -20}, {40, 20}}, textString = "Out1", fontSize = 100),
Text(origin = {67, -43}, extent = {{-39, 23}, {33, -15}}, textString =
"Out2")}, coordinateSystem(initialScale = 0.1)));
end BranchSQR_I90_O0_O180;


// Branch with downwards input(-y) and -y and -x direction split

model BranchSQR_I90_O180_O270
  parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (Y axis)";
  parameter Real width_branch = 1 "Conveyor width (X axis)";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-2, 84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-2, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {0, -84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-80, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x - width_branch / 2;
  blockInterfaceOut1.y = blockInterfaceIn.y - Length_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x;
  blockInterfaceOut2.y = blockInterfaceIn.y - Length_branch ;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {38, 81}, extent = {{-40, -20},
{40, 20}}, textString = "In", fontSize = 100), Text(origin = {-64, -37},
extent = {{-40, -20}, {40, 20}}, textString = "Out1", fontSize = 100),
Text(origin = {63, -85}, extent = {{-39, 23}, {33, -15}}, textString =
"Out2")}, coordinateSystem(initialScale = 0.1)));
end BranchSQR_I90_O180_O270;


// Branch with downwards input(-y) and -y and +x direction split

model BranchSQR_I90_O0_O270

  parameter Real splitProp = 1 "blockInterfaceOut1 / blockInterfaceOut2";
  parameter Real Length_branch = 1 "Conveyor length (Y axis)";
  parameter Real width_branch = 1 "Conveyor width (X axis)";
  blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-2, 84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-2, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {0, -84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-80, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut1.x = blockInterfaceIn.x + width_branch / 2;
  blockInterfaceOut1.y = blockInterfaceIn.y - Length_branch / 2;
  blockInterfaceOut2.x = blockInterfaceIn.x;
  blockInterfaceOut2.y = blockInterfaceIn.y - Length_branch ;
  blockInterfaceOut1.throughput + blockInterfaceOut2.throughput =
blockInterfaceIn.throughput;
  splitProp = blockInterfaceOut1.throughput / blockInterfaceOut2.throughput;
  annotation(uses(Modelica(version = "3.2.1")), Icon(graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, -100}, {100, 100}}), Text(origin = {38, 81}, extent = {{-40, -20},
{40, 20}}, textString = "In", fontSize = 100), Text(origin = {66, -39}, extent
= {{-40, -20}, {40, 20}}, textString = "Out1", fontSize = 100), Text(origin =
{-55, -83}, extent = {{-39, 23}, {33, -15}}, textString = "Out2")},
coordinateSystem(initialScale = 0.1)));
end BranchSQR_I90_O0_O270;
end BranchSquare;
```

Branch system

```
package BranchSystem

// Branch system in +x direction

  model Branch
    blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {-90, 4}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    StraightConveyor.StraightFix straightFix1(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {-66, 4}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
    StraightConveyor.StraightFix straightFix2(Length_straight = 1,
angle_straight = -90) annotation(Placement(visible = true,
transformation(origin = {-38, -26}, extent = {{-10, -10}, {10, 10}}, rotation
= -90)));
    CurveConveyor.Curve_I90_O0 Curve_I90_O01 annotation(Placement(visible =
true, transformation(origin = {-38, -54}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
    StraightConveyor.StraightFix straightFix3(Length_straight = 1,
angle_straight = 90) annotation(Placement(visible = true,
transformation(origin = {-38, 34}, extent = {{-10, -10}, {10, 10}}, rotation =
90)));
    CurveConveyor.Curve_I270_O0 Curve_I270_O01 annotation(Placement(visible =
true, transformation(origin = {-38, 62}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
    blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {20, 62}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {20, -54}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    StraightConveyor.StraightFix straightFix4(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {-8, 62}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
```

```
    StraightConveyor.StraightFix straightFix5(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {-8, -56}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
    BranchSquare.BranchSQR_I180_O90_O270 BranchSQR_I180_O90_O2701
annotation(Placement(visible = true, transformation(origin = {-38, 2}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));

  equation
    connect(BranchSQR_I180_O90_O2701.blockInterfaceOut2,
straightFix2.blockInterfaceIn) annotation(Line(points = {{-38, -6}, {-38, -6},
{-38, -18}, {-38, -18}}));
    connect(BranchSQR_I180_O90_O2701.blockInterfaceOut1,
straightFix3.blockInterfaceIn) annotation(Line(points = {{-38, 10}, {-38, 10},
{-38, 26}, {-38, 26}}));
    connect(straightFix1.blockInterfaceOut,
BranchSQR_I180_O90_O2701.blockInterfaceIn) annotation(Line(points = {{-58, 4},
{-44, 4}, {-44, 2}, {-46, 2}}));
    connect(straightFix5.blockInterfaceOut, blockInterfaceOut2)
annotation(Line(points = {{0, -56}, {20, -56}, {20, -54}, {20, -54}}));
    connect(Curve_I90_O01.blockInterfaceOut, straightFix5.blockInterfaceIn)
annotation(Line(points = {{-30, -54}, {-16, -54}, {-16, -56}, {-16, -56}}));
    connect(straightFix4.blockInterfaceOut, blockInterfaceOut1)
annotation(Line(points = {{0, 62}, {20, 62}, {20, 62}, {20, 62}}));
    connect(Curve_I270_O01.blockInterfaceOut, straightFix4.blockInterfaceIn)
annotation(Line(points = {{-30, 62}, {-16, 62}, {-16, 62}, {-16, 62}}));
    connect(straightFix2.blockInterfaceOut, Curve_I90_O01.blockInterfaceIn)
annotation(Line(points = {{-38, -34}, {-38, -34}, {-38, -46}, {-38, -46}}));
    connect(straightFix3.blockInterfaceOut, Curve_I270_O01.blockInterfaceIn)
annotation(Line(points = {{-38, 42}, {-38, 42}, {-38, 54}, {-38, 54}}));
    connect(blockInterfaceIn, straightFix1.blockInterfaceIn)
annotation(Line(points = {{-90, 4}, {-74, 4}, {-74, 4}, {-74, 4}}));
    annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
Icon(coordinateSystem(initialScale = 0.1), graphics = {Line(points = {{90,
80}, {0, 80}, {0, 0}, {-90, 0}}, color = {0, 0, 255}, thickness = 10),
Line(points = {{90, -80}, {0, -80}, {0, 0}}, color = {0, 0, 255}, thickness =
10), Text(origin = {136, -28}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-90, 41}, {-26, 99}}, textString = "1"),
Text(origin = {70, -7}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-15, -1}, {29, -61}}, textString = "2")}));
  end Branch;


// Branch system in -x direction

  model Branch_Left
    blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {78, -2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {78, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-76, 56}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {-74, -64}, extent = {{-10, -10}, {10, 10}}, rotation
= 0), iconTransformation(origin = {-80, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    StraightConveyor.StraightFix straightFix1(Length_straight = 1,
angle_straight = 180) annotation(Placement(visible = true,
transformation(origin = {54, -2}, extent = {{-10, -10}, {10, 10}}, rotation =
180)));
    StraightConveyor.StraightFix straightFix2(Length_straight = 1,
angle_straight = 90) annotation(Placement(visible = true,
transformation(origin = {22, 24}, extent = {{-10, -10}, {10, 10}}, rotation =
90)));
    StraightConveyor.StraightFix straightFix3(Length_straight = 1,
angle_straight = -90) annotation(Placement(visible = true,
transformation(origin = {24, -32}, extent = {{-10, -10}, {10, 10}}, rotation =
-90)));
    CurveConveyor.Curve_I270_O180 Curve_I270_O1801
annotation(Placement(visible = true, transformation(origin = {22, 54}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
    CurveConveyor.Curve_I90_O180 Curve_I90_O1801 annotation(Placement(visible
= true, transformation(origin = {24, -64}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
    StraightConveyor.StraightFix straightFix4(Length_straight = 1,
angle_straight = 180) annotation(Placement(visible = true,
transformation(origin = {-26, -66}, extent = {{-10, -10}, {10, 10}}, rotation
= 180)));
    StraightConveyor.StraightFix straightFix5(Length_straight = 1,
angle_straight = 180) annotation(Placement(visible = true,
transformation(origin = {-26, 54}, extent = {{-10, -10}, {10, 10}}, rotation =
180)));
    BranchSquare.BranchSQR_I0_O90_O270 BranchSQR_I0_O90_O2701
annotation(Placement(visible = true, transformation(origin = {22, -2}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  equation
    connect(blockInterfaceIn, straightFix1.blockInterfaceIn)
annotation(Line(points = {{78, -2}, {59, -2}}));
    connect(straightFix1.blockInterfaceOut,
BranchSQR_I0_O90_O2701.blockInterfaceIn) annotation(Line(points = {{42, -2},
{30, -2}}));
    connect(straightFix2.blockInterfaceOut, Curve_I270_O1801.blockInterfaceIn)
annotation(Line(points = {{22, 36}, {22, 46}}));
    connect(BranchSQR_I0_O90_O2701.blockInterfaceOut1,
straightFix2.blockInterfaceIn) annotation(Line(points = {{22, 6}, {22, 19}}));
    connect(BranchSQR_I0_O90_O2701.blockInterfaceOut2,
straightFix3.blockInterfaceIn) annotation(Line(points = {{22, -10}, {24, -10},
{24, -24}, {24, -24}}));
    connect(straightFix4.blockInterfaceOut, blockInterfaceOut2)
annotation(Line(points = {{-34, -66}, {-70, -66}, {-70, -64}, {-74, -64}}));
    connect(Curve_I90_O1801.blockInterfaceOut, straightFix4.blockInterfaceIn)
annotation(Line(points = {{16, -64}, {-18, -64}, {-18, -66}, {-18, -66}}));
    connect(straightFix5.blockInterfaceOut, blockInterfaceOut1)
annotation(Line(points = {{-34, 54}, {-70, 54}, {-70, 56}, {-76, 56}}));
    connect(Curve_I270_O1801.blockInterfaceOut, straightFix5.blockInterfaceIn)
annotation(Line(points = {{14, 54}, {-18, 54}, {-18, 54}, {-18, 54}}));
    connect(straightFix3.blockInterfaceOut, Curve_I90_O1801.blockInterfaceIn)
annotation(Line(points = {{24, -40}, {24, -40}, {24, -56}, {24, -56}}));
    annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
Icon(coordinateSystem(initialScale = 0.1), graphics = {Line(points = {{-90,
80}, {0, 80}, {0, 0}, {90, 0}}, color = {0, 0, 255}, thickness = 10),
Line(points = {{-90, -80}, {0, -80}, {0, 0}}, color = {0, 0, 255}, thickness =
10), Text(origin = {136, -28}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-90, 41}, {-26, 99}}, textString = "1"),
Text(origin = {70, -7}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-15, -1}, {29, -61}}, textString = "2")}));
  end Branch_Left;


// Branch system in -y direction

  model Branch_Downwards
    blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {2, 70}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-2, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-66, -74}, extent = {{-10, -10}, {10, 10}}, rotation
= 0), iconTransformation(origin = {-80, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {68, -74}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    StraightConveyor.StraightFix straightFix1(Length_straight = 1,
angle_straight = -90) annotation(Placement(visible = true,
transformation(origin = {2, 42}, extent = {{-10, -10}, {10, 10}}, rotation = -
90)));
    BranchSquare.BranchSQR_I90_O0_O180 BranchSQR_I90_O0_O1801
annotation(Placement(visible = true, transformation(origin = {2, 14}, extent =
{{-10, -10}, {10, 10}}, rotation = 0)));
    StraightConveyor.StraightFix straightFix2(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {34, 14}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
    StraightConveyor.StraightFix straightFix3(Length_straight = 1,
angle_straight = 180) annotation(Placement(visible = true,
transformation(origin = {-28, 14}, extent = {{-10, -10}, {10, 10}}, rotation =
180)));
    CurveConveyor.Curve_I180_O270 Curve_I180_O2701
annotation(Placement(visible = true, transformation(origin = {66, 14}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
    CurveConveyor.Curve_I0_O270 Curve_I0_O2701 annotation(Placement(visible =
true, transformation(origin = {-66, 14}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
    StraightConveyor.StraightFix straightFix4(Length_straight = 1,
angle_straight = -90) annotation(Placement(visible = true,
transformation(origin = {-66, -32}, extent = {{-10, -10}, {10, 10}}, rotation
= -90)));
    StraightConveyor.StraightFix straightFix5(Length_straight = 1,
angle_straight = -90) annotation(Placement(visible = true,
transformation(origin = {66, -28}, extent = {{-10, -10}, {10, 10}}, rotation =
-90)));
  equation
    connect(straightFix4.blockInterfaceOut, blockInterfaceOut1)
annotation(Line(points = {{-66, -40}, {-68, -40}, {-68, -74}, {-66, -74}}));
    connect(Curve_I0_O2701.blockInterfaceOut, straightFix4.blockInterfaceIn)
annotation(Line(points = {{-66, 6}, {-66, -24}}));
    connect(straightFix3.blockInterfaceOut, Curve_I0_O2701.blockInterfaceIn)
annotation(Line(points = {{-36, 14}, {-56, 14}, {-56, 14}, {-58, 14}}));
    connect(BranchSQR_I90_O0_O1801.blockInterfaceOut1,
straightFix3.blockInterfaceIn) annotation(Line(points = {{-6, 14}, {-20, 14},
{-20, 14}, {-20, 14}}));
    connect(straightFix5.blockInterfaceOut, blockInterfaceOut2)
annotation(Line(points = {{66, -36}, {68, -36}, {68, -74}, {68, -74}}));
    connect(Curve_I180_O2701.blockInterfaceOut, straightFix5.blockInterfaceIn)
annotation(Line(points = {{66, 6}, {66, 6}, {66, -20}, {66, -20}}));
    connect(straightFix2.blockInterfaceOut, Curve_I180_O2701.blockInterfaceIn)
annotation(Line(points = {{42, 14}, {58, 14}, {58, 14}, {58, 14}}));
    connect(BranchSQR_I90_O0_O1801.blockInterfaceOut2,
straightFix2.blockInterfaceIn) annotation(Line(points = {{10, 14}, {26, 14},
{26, 14}, {26, 14}}));
    connect(straightFix1.blockInterfaceOut,
BranchSQR_I90_O0_O1801.blockInterfaceIn) annotation(Line(points = {{2, 34},
{2, 34}, {2, 22}, {2, 22}}));
    connect(blockInterfaceIn, straightFix1.blockInterfaceIn)
annotation(Line(points = {{2, 70}, {2, 70}, {2, 50}, {2, 50}}));
    annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
Icon(coordinateSystem(initialScale = 0.1), graphics = {Line(points = {{0, 90},
{0, 0}, {-80, 0}, {-80, -90}}, color = {0, 0, 255}, thickness = 10),
Line(points = {{0, 0}, {80, 0}, {80, -80}}, color = {0, 0, 255}, thickness =
10), Text(origin = {12, -144}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-90, 41}, {-26, 99}}, textString = "1"),
Text(origin = {32, -41}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-15, -1}, {29, -61}}, textString = "2")}));
  end Branch_Downwards;


// Branch system in +y direction
```

```
  model Branch_Upwards
    blockInterface blockInterfaceIn annotation(Placement(visible = true,
transformation(origin = {0, -82}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    blockInterface blockInterfaceOut1 annotation(Placement(visible = true,
transformation(origin = {-66, 78}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    blockInterface blockInterfaceOut2 annotation(Placement(visible = true,
transformation(origin = {68, 74}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {82, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
    StraightConveyor.StraightFix straightFix1(Length_straight = 1,
angle_straight = 90) annotation(Placement(visible = true,
transformation(origin = {0, -50}, extent = {{-10, -10}, {10, 10}}, rotation =
90)));
    BranchSquare.BranchSQR_I270_O180_O0 BranchSQR_I270_O180_O01
annotation(Placement(visible = true, transformation(origin = {0, -16}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
    StraightConveyor.StraightFix straightFix2(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {34, -16}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
    StraightConveyor.StraightFix straightFix3(Length_straight = 1,
angle_straight = 180) annotation(Placement(visible = true,
transformation(origin = {-32, -16}, extent = {{-10, -10}, {10, 10}}, rotation
= 180)));

    CurveConveyor.Curve_I0_O90 Curve_I0_O901 annotation(Placement(visible =
true, transformation(origin = {-66, -16}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
    StraightConveyor.StraightFix straightFix4(Length_straight = 1,
angle_straight = 90) annotation(Placement(visible = true,
transformation(origin = {-66, 18}, extent = {{-10, -10}, {10, 10}}, rotation =
90)));
    StraightConveyor.StraightFix straightFix5(Length_straight = 1,
angle_straight = 90) annotation(Placement(visible = true,
transformation(origin = {66, 18}, extent = {{-10, -10}, {10, 10}}, rotation =
90)));
  CurveConveyor.Curve_I180_O90 curve_I180_O901 annotation(Placement(visible =
true, transformation(origin = {66, -16}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  equation
    connect(curve_I180_O901.blockInterfaceOut, straightFix5.blockInterfaceIn)
annotation(Line(points = {{66, -8}, {66, -8}, {66, 12}, {66, 12}}));
    connect(straightFix2.blockInterfaceOut, curve_I180_O901.blockInterfaceIn)
annotation(Line(points = {{46, -16}, {58, -16}, {58, -16}, {58, -16}}));
    connect(straightFix4.blockInterfaceOut, blockInterfaceOut1)
annotation(Line(points = {{-66, 26}, {-68, 26}, {-68, 78}, {-66, 78}}));
    connect(Curve_I0_O901.blockInterfaceOut, straightFix4.blockInterfaceIn)
annotation(Line(points = {{-66, -8}, {-66, -8}, {-66, 10}, {-66, 10}}));
    connect(straightFix3.blockInterfaceOut, Curve_I0_O901.blockInterfaceIn)
annotation(Line(points = {{-40, -16}, {-58, -16}, {-58, -16}, {-58, -16}}));
    connect(BranchSQR_I270_O180_O01.blockInterfaceOut1,
straightFix3.blockInterfaceIn) annotation(Line(points = {{-8, -16}, {-24, -
16}, {-24, -16}, {-24, -16}}));
    connect(straightFix5.blockInterfaceOut, blockInterfaceOut2)
annotation(Line(points = {{66, 26}, {68, 26}, {68, 74}, {68, 74}}));


    connect(BranchSQR_I270_O180_O01.blockInterfaceOut2,
straightFix2.blockInterfaceIn) annotation(Line(points = {{8, -16}, {26, -16},
{26, -16}, {26, -16}}));
    connect(straightFix1.blockInterfaceOut,
BranchSQR_I270_O180_O01.blockInterfaceIn) annotation(Line(points = {{0, -42},
{0, -42}, {0, -24}, {0, -24}}));
    connect(blockInterfaceIn, straightFix1.blockInterfaceIn)
annotation(Line(points = {{0, -82}, {0, -82}, {0, -58}, {0, -58}}));
```

```
    annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
Icon(coordinateSystem(initialScale = 0.1), graphics = {Line(points = {{0, -
90}, {0, 0}, {-80, 0}, {-80, 90}}, color = {0, 0, 255}, thickness = 10),
Line(points = {{0, 0}, {80, 0}, {80, 80}}, color = {0, 0, 255}, thickness =
10), Text(origin = {12, 12}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-90, 41}, {-26, 99}}, textString = "1"),
Text(origin = {40, 111}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-15, -1}, {29, -61}}, textString = "2")}));
  end Branch_Upwards;
end BranchSystem;
```

Merge Square

```
package MergeSquare

//Merge Square Block From up and down

  model MergeSQR_I90_I270_O0
  parameter Real Length_merge = 1 "Conveyor Merge length (X Axis)";
  parameter Real width_merge = 1 "Conveyor Merge width (Y axis)";


  blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-2, 84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-2, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {-2, -86}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-2, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {72, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, -2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut.x = blockInterfaceIn1.x + Length_merge/2;
  blockInterfaceOut.y = blockInterfaceIn1.y - width_merge /2;
  blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput


annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-99, 110}, {101, -
100}})}));annotation(Icon(graphics = {Rectangle(fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-100, 100}, {100, -100}}),
Text(origin = {72, 35}, extent = {{-26, 21}, {26, -21}}, textString = "Out"),
Text(origin = {-54, 85}, extent = {{-20, 17}, {28, -23}}, textString = "In1"),
Text(origin = {-61, -76}, extent = {{-27, 16}, {37, -22}}, textString =
"In2")}, coordinateSystem(initialScale = 0.1)));

end MergeSQR_I90_I270_O0;


//Merge Square Block From up and straight

model MergeSQR_I90_I180_O0

  parameter Real Length_merge = 1 "Conveyor Merge length (X Axis)";
  parameter Real width_merge = 1 "Conveyor Merge width (Y axis)";
  blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {-82, 8}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 6}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
```

```
  blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {0, 82}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {4, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {82, 4}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 4}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut.x = blockInterfaceIn2.x + Length_merge;
  blockInterfaceOut.y = blockInterfaceIn2.y;
  blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput




annotation(Icon(coordinateSystem(initialScale =
0.1)));annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, 100}, {100, -100}}), Text(origin = {-42, 80}, extent = {{-22, 20},
{22, -20}}, textString = "In1"), Text(origin = {-76, -35}, extent = {{-18,
21}, {26, -27}}, textString = "In2"), Text(origin = {63, -31}, extent = {{-17,
15}, {33, -31}}, textString = "Out")}));

end MergeSQR_I90_I180_O0;


//Merge Square Block From Down and straight

model MergeSQR_I180_I270_O0
  parameter Real Length_merge = 1 "Conveyor Merge length (X Axis)";
  parameter Real width_merge = 1 "Conveyor Merge width (Y axis)";
  blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {-2, -88}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-78, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {82, 4}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 4}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut.x = blockInterfaceIn1.x + Length_merge;
  blockInterfaceOut.y = blockInterfaceIn2.y + width_merge/2 ;
  blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput annotation(Icon(coordinateSystem(initialScale =
0.1)));
  annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(fillColor = {0, 85, 255}, fillPattern = FillPattern.Solid, extent =
{{-100, 100}, {100, -100}}), Text(origin = {-74, 44}, extent = {{-22, 20},
{22, -20}}, textString = "In1"), Text(origin = {-50, -75}, extent = {{-18,
21}, {26, -27}}, textString = "In2"), Text(origin = {63, 55}, extent = {{-17,
15}, {33, -31}}, textString = "Out")}));
end MergeSQR_I180_I270_O0;


//Merge Square Block, -x direction From above and below

model MergeSQR_I90_I270_O180

parameter Real Length_merge = 1 "Conveyor Merge length (X Axis)";
parameter Real width_merge = 1 "Conveyor Merge width (Y axis)";
```

```
blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-2, 84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-2, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {-2, -86}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-2, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {-84, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
blockInterfaceOut.x = blockInterfaceIn1.x - Length_merge/2;
blockInterfaceOut.y = blockInterfaceIn1.y - width_merge /2;
blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput


annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-99, 110}, {101, -
100}})}));annotation(Icon(graphics = {Rectangle(fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-100, 100}, {100, -100}}),
Text(origin = {-30, 3}, extent = {{-26, 21}, {26, -21}}, textString = "Out"),
Text(origin = {44, 83}, extent = {{-20, 17}, {28, -23}}, textString = "In1"),
Text(origin = {45, -76}, extent = {{-27, 16}, {37, -22}}, textString =
"In2")}, coordinateSystem(initialScale = 0.1)));

end MergeSQR_I90_I270_O180;


//Merge Square Block, -x direction From above and straight

model MergeSQR_I0_I90_O180


parameter Real Length_merge = 1 "Conveyor Merge length (X Axis)";
parameter Real width_merge = 1 "Conveyor Merge width (Y axis)";


blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-2, 84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-2, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {82, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {-84, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
blockInterfaceOut.x = blockInterfaceIn1.x - Length_merge/2;
blockInterfaceOut.y = blockInterfaceIn2.y;
blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput


annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-99, 110}, {101, -
100}})}));annotation(Icon(graphics = {Rectangle(fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-100, 100}, {100, -100}}),
Text(origin = {-72, -37}, extent = {{-26, 21}, {26, -21}}, textString =
"Out"), Text(origin = {44, 83}, extent = {{-20, 17}, {28, -23}}, textString =
"In1"), Text(origin = {71, -34}, extent = {{-27, 16}, {37, -22}}, textString =
"In2")}, coordinateSystem(initialScale = 0.1)));

end MergeSQR_I0_I90_O180;


//Merge Square Block, -x direction From below and straight

model MergeSQR_I0_I270_O180


parameter Real Length_merge = 1 "Conveyor Merge length (X Axis)";
parameter Real width_merge = 1 "Conveyor Merge width (Y axis)";


blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-2, 84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {4, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {82, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {-84, 2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
blockInterfaceOut.x = blockInterfaceIn1.x - Length_merge/2;
blockInterfaceOut.y = blockInterfaceIn2.y;
blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput


annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-99, 110}, {101, -
100}})}));annotation(Icon(graphics = {Rectangle(fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-100, 100}, {100, -100}}),
Text(origin = {-68, 41}, extent = {{-26, 21}, {26, -21}}, textString = "Out"),
Text(origin = {48, -77}, extent = {{-20, 17}, {28, -23}}, textString = "In1"),
Text(origin = {69, 44}, extent = {{-27, 16}, {37, -22}}, textString = "In2")},
coordinateSystem(initialScale = 0.1)));
end MergeSQR_I0_I270_O180;


//Merge Square Block, +y direction From left and right

model MergeSQR_I0_I180_O90

parameter Real Length_merge = 1 "Conveyor Merge length (Y Axis)";
parameter Real width_merge = 1 "Conveyor Merge width (X axis)";


blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-80, -2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, -2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {82, 0}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80,-2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {0, 84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
blockInterfaceOut.y = blockInterfaceIn1.y + Length_merge/2;
blockInterfaceOut.x = blockInterfaceIn1.x + width_merge /2;
blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput


annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-99, 110}, {101, -
100}})}));annotation(Icon(graphics = {Rectangle(fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-100, 100}, {100, -100}}),
Text(origin = {2, 45}, extent = {{-26, 21}, {26, -21}}, textString = "Out"),
Text(origin = {-78, -37}, extent = {{-20, 17}, {28, -23}}, textString =
"In1"), Text(origin = {69, -36}, extent = {{-27, 16}, {37, -22}}, textString =
"In2")}, coordinateSystem(initialScale = 0.1)));

end MergeSQR_I0_I180_O90;


//Merge Square Block, +y direction From left and straight

model MergeSQR_I180_I270_O90

parameter Real Length_merge = 1 "Conveyor Merge length (Y Axis)";
parameter Real width_merge = 1 "Conveyor Merge width (X axis)";


blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-80, -2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, -2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {4, -84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {2, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {0, 84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
blockInterfaceOut.x = blockInterfaceIn2.x;
blockInterfaceOut.y = blockInterfaceIn1.y + Length_merge /2;
blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput


annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-99, 110}, {101, -
100}})}));annotation(Icon(graphics = {Rectangle(fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-100, 100}, {100, -100}}),
Text(origin = {2, 45}, extent = {{-26, 21}, {26, -21}}, textString = "Out"),
Text(origin = {-78, -37}, extent = {{-20, 17}, {28, -23}}, textString =
"In1"), Text(origin = {43, -76}, extent = {{-27, 16}, {37, -22}}, textString =
"In2")}, coordinateSystem(initialScale = 0.1)));
end MergeSQR_I180_I270_O90;


//Merge Square Block, +y direction From right and straight

model MergeSQR_I0_I270_O90


parameter Real Length_merge = 1 "Conveyor Merge length (Y Axis)";
parameter Real width_merge = 1 "Conveyor Merge width (X axis)";


blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {80, -4}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, -4}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {4, -84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {2, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {0, 84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
blockInterfaceOut.x = blockInterfaceIn2.x;
blockInterfaceOut.y = blockInterfaceIn1.y + Length_merge /2;
blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput


annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-99, 110}, {101, -
100}})}));annotation(Icon(graphics = {Rectangle(fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-100, 100}, {100, -100}}),
Text(origin = {2, 45}, extent = {{-26, 21}, {26, -21}}, textString = "Out"),
Text(origin = {72, -37}, extent = {{-20, 17}, {28, -23}}, textString = "In1"),
Text(origin = {-49, -76}, extent = {{-27, 16}, {37, -22}}, textString =
"In2")}, coordinateSystem(initialScale = 0.1)));
end MergeSQR_I0_I270_O90;


//Merge Square Block, -y direction From left and right

model MergeSQR_I0_I180_O270

parameter Real Length_merge = 1 "Conveyor Merge length (Y Axis)";
parameter Real width_merge = 1 "Conveyor Merge width (X axis)";


blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-80, -2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, -2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {82, 0}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80,-2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {4, -84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
blockInterfaceOut.x = blockInterfaceIn1.x + width_merge/2;
blockInterfaceOut.y = blockInterfaceIn1.y - Length_merge /2;
blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput


annotation(Icon(coordinateSystem(initialScale = 0.1), graphics =
{Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255}, fillPattern =
FillPattern.Solid, extent = {{-99, 110}, {101, -
100}})}));annotation(Icon(graphics = {Rectangle(fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-100, 100}, {100, -100}}),
Text(origin = {-2, -39}, extent = {{-26, 21}, {26, -21}}, textString = "Out"),
Text(origin = {-76, 39}, extent = {{-20, 17}, {28, -23}}, textString = "In1"),
Text(origin = {69, 38}, extent = {{-27, 16}, {37, -22}}, textString = "In2")},
coordinateSystem(initialScale = 0.1)));

end MergeSQR_I0_I180_O270;
```

```
//Merge Square Block, -y direction From left and straight

model MergeSQR_I180_I90_O270
  parameter Real Length_merge = 1 "Conveyor Merge length (Y Axis)";
  parameter Real width_merge = 1 "Conveyor Merge width (X axis)";
  blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-80, -2}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, -2}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {-2, 80}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {2,80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {4, -84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut.x = blockInterfaceIn2.x;
  blockInterfaceOut.y = blockInterfaceIn2.y - Length_merge;
  blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput annotation(Icon(coordinateSystem(initialScale =
0.1), graphics = {Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-99, 110}, {101, -100}})}));
  annotation(Icon(graphics = {Rectangle(fillColor = {0, 85, 255}, fillPattern
= FillPattern.Solid, extent = {{-100, 100}, {100, -100}}), Text(origin = {52,
-77}, extent = {{-26, 21}, {26, -21}}, textString = "Out"), Text(origin = {-
34, 3}, extent = {{-20, 17}, {28, -23}}, textString = "In1"), Text(origin =
{43, 84}, extent = {{-27, 16}, {37, -22}}, textString = "In2")},
coordinateSystem(initialScale = 0.1)));
end MergeSQR_I180_I90_O270;


//Merge Square Block, -y direction From right and straight

model MergeSQR_I0_I90_O270
  parameter Real Length_merge = 1 "Conveyor Merge length (Y Axis)";
  parameter Real width_merge = 1 "Conveyor Merge width (X axis)";
  blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {78, 0}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 4}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {2,82}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {2,80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {4, -84}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
equation
  blockInterfaceOut.x = blockInterfaceIn2.x;
  blockInterfaceOut.y = blockInterfaceIn1.y - Length_merge / 2;
  blockInterfaceOut.throughput = blockInterfaceIn1.throughput +
blockInterfaceIn2.throughput annotation(Icon(coordinateSystem(initialScale =
0.1), graphics = {Rectangle(origin = {-1, -10}, fillColor = {0, 85, 255},
fillPattern = FillPattern.Solid, extent = {{-99, 110}, {101, -100}})}));
  annotation(Icon(graphics = {Rectangle(fillColor = {0, 85, 255}, fillPattern
= FillPattern.Solid, extent = {{-100, 100}, {100, -100}}), Text(origin = {-50,
-79}, extent = {{-26, 21}, {26, -21}}, textString = "Out"), Text(origin = {30,
11}, extent = {{-20, 17}, {28, -23}}, textString = "In1"), Text(origin = {-53,
82}, extent = {{-27, 16}, {37, -22}}, textString = "In2")},
coordinateSystem(initialScale = 0.1)));
end MergeSQR_I0_I90_O270;
end MergeSquare;
```

Merge system

```
package MergeSystem

//Merge System in direction to the right

  model Merge_Rightwards
  blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-78, 78}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {-76, -50}, extent = {{-10, -10}, {10, 10}}, rotation
= 0), iconTransformation(origin = {-80, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {70, 10}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  StraightConveyor.StraightFix straightFix1(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {-56, 76}, extent
= {{-10, -10}, {20, 10}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix2(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {-50, -50},
extent = {{-10, -10}, {20, 10}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix3(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {24, 10}, extent
= {{-10, -10}, {20, 10}}, rotation = 0)));
  CurveConveyor.Curve_I180_O270 Curve_I180_O2701 annotation(Placement(visible
= true, transformation(origin = {-4, 76}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));

  StraightConveyor.StraightFix straightFix4(Length_straight = 1,
angle_straight = -90)  annotation(Placement(visible = true,
transformation(origin = {-4, 54}, extent = {{-10, -10}, {20, 10}}, rotation =
-90)));
  StraightConveyor.StraightFix straightFix5(Length_straight = 1,
angle_straight = 90)  annotation(Placement(visible = true,
transformation(origin = {-4, -32}, extent = {{-10, -10}, {20, 10}}, rotation =
90)));
  MergeSquare.MergeSQR_I90_I270_O0 MergeSQR_I90_I270_O01
annotation(Placement(visible = true, transformation(origin = {-4, 10}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
 CurveConveyor.Curve_I180_O90 curve_I180_O901 annotation(Placement(visible =
true, transformation(origin = {-4, -50}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));

equation
    connect(curve_I180_O901.blockInterfaceOut, straightFix5.blockInterfaceIn)
annotation(Line(points = {{-4, -42}, {-4, -42}, {-4, -34}, {-4, -34}}));
    connect(straightFix2.blockInterfaceOut, curve_I180_O901.blockInterfaceIn)
annotation(Line(points = {{-26, -50}, {-12, -50}, {-12, -50}, {-12, -50}}));
  connect(MergeSQR_I90_I270_O01.blockInterfaceOut,
straightFix3.blockInterfaceIn) annotation(Line(points = {{4, 10}, {21, 10}}));
  connect(straightFix5.blockInterfaceOut,
MergeSQR_I90_I270_O01.blockInterfaceIn2) annotation(Line(points = {{-4, -8},
{-4, 2}}));
  connect(straightFix4.blockInterfaceOut,
MergeSQR_I90_I270_O01.blockInterfaceIn1) annotation(Line(points = {{-4, 30},
{-4, 18}}));
  connect(straightFix3.blockInterfaceOut, blockInterfaceOut)
annotation(Line(points = {{47, 10}, {70, 10}}));

   connect(blockInterfaceIn2, straightFix2.blockInterfaceIn)
annotation(Line(points = {{-76, -50}, {-52, -50}, {-52, -50}, {-52, -50}}));
  connect(Curve_I180_O2701.blockInterfaceOut, straightFix4.blockInterfaceIn)
annotation(Line(points = {{-4, 68}, {-4, 68}, {-4, 58}, {-4, 58}}));
```

```
  connect(straightFix1.blockInterfaceOut, Curve_I180_O2701.blockInterfaceIn)
annotation(Line(points = {{-32, 76}, {-12, 76}, {-12, 76}, {-12, 76}}));
  connect(blockInterfaceIn1, straightFix1.blockInterfaceIn)
annotation(Line(points = {{-78, 78}, {-58, 78}, {-58, 76}, {-60, 76}}));

 annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
 Icon(coordinateSystem(initialScale = 0.1), graphics = {
    Line(points = {{-90, 80}, {0, 80}, {0, 0}, {90, 0}}, color = {0, 0, 255},
thickness = 10),
    Line(points = {{-90, -80}, {0, -80}, {0, 0}}, color = {0, 0, 255},
thickness = 10),
    Text(origin = {-24, -28}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-90, 41}, {-26, 99}}, textString = "1"),
    Text(origin = {-86, -5}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-15, -1}, {29, -61}}, textString = "2")}));

end Merge_Rightwards;

//Merge System in direction to the left

model Merge_Leftwards
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {-82, 4}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 0}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {80, 74}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {80, -66}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  StraightConveyor.StraightFix straightFix1(Length_straight = 1,
angle_straight = 180)  annotation(Placement(visible = true,
transformation(origin = {56, 74}, extent = {{-10, -10}, {20, 10}}, rotation =
180)));
  StraightConveyor.StraightFix straightFix2(Length_straight = 1,
angle_straight = 180) annotation(Placement(visible = true,
transformation(origin = {58, -66}, extent = {{-10, -10}, {20, 10}}, rotation =
180)));
  StraightConveyor.StraightFix straightFix3(Length_straight = 1,
angle_straight = 180) annotation(Placement(visible = true,
transformation(origin = {-32, 4}, extent = {{-10, -10}, {20, 10}}, rotation =
180)));
  CurveConveyor.Curve_I0_O90 Curve_I0_O901 annotation(Placement(visible =
true, transformation(origin = {4, -66}, extent = {{-14, -14}, {14, 14}},
rotation = 0)));
  CurveConveyor.Curve_I0_O270 Curve_I0_O2701 annotation(Placement(visible =
true, transformation(origin = {4, 74}, extent = {{-18, -18}, {18, 18}},
rotation = 0)));
  MergeSquare.MergeSQR_I90_I270_O180 MergeSQR_I90_I270_O1801
annotation(Placement(visible = true, transformation(origin = {4, 4}, extent =
{{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix4(Length_straight = 1,
angle_straight = -90)  annotation(Placement(visible = true,
transformation(origin = {4, 46}, extent = {{-10, -10}, {20, 10}}, rotation = -
90)));
  StraightConveyor.StraightFix straightFix5(Length_straight = 1,
angle_straight = 90)  annotation(Placement(visible = true,
transformation(origin = {4, -38}, extent = {{-10, -10}, {20, 10}}, rotation =
90)));
equation
  connect(straightFix1.blockInterfaceOut, Curve_I0_O2701.blockInterfaceIn)
annotation(Line(points = {{33, 74}, {18, 74}}));
  connect(blockInterfaceIn1, straightFix1.blockInterfaceIn)
annotation(Line(points = {{80, 74}, {59, 74}}));
```

```
  connect(Curve_I0_O901.blockInterfaceOut, straightFix5.blockInterfaceIn)
annotation(Line(points = {{4, -55}, {4, -42}}));
  connect(straightFix2.blockInterfaceOut, Curve_I0_O901.blockInterfaceIn)
annotation(Line(points = {{35, -66}, {15, -66}}));
  connect(blockInterfaceIn2, straightFix2.blockInterfaceIn)
annotation(Line(points = {{80, -66}, {61, -66}}));
  connect(straightFix3.blockInterfaceOut, blockInterfaceOut)
annotation(Line(points = {{-55, 4}, {-82, 4}}));
  connect(MergeSQR_I90_I270_O1801.blockInterfaceOut,
straightFix3.blockInterfaceIn) annotation(Line(points = {{-4, 4}, {-29, 4}}));
  connect(straightFix5.blockInterfaceOut,
MergeSQR_I90_I270_O1801.blockInterfaceIn2) annotation(Line(points = {{4, -14},
{4, -14}, {4, -4}, {4, -4}}));
  connect(straightFix4.blockInterfaceOut,
MergeSQR_I90_I270_O1801.blockInterfaceIn1) annotation(Line(points = {{4, 22},
{4, 22}, {4, 12}, {4, 12}}));
  connect(Curve_I0_O2701.blockInterfaceOut, straightFix4.blockInterfaceIn)
annotation(Line(points = {{4, 60}, {4, 60}, {4, 50}, {4, 50}}));

   annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
   Icon(coordinateSystem(initialScale = 0.1), graphics = {
    Line(points = {{90, 80}, {0, 80}, {0, 0}, {-90, 0}}, color = {0, 0, 255},
thickness = 10),
    Line(points = {{90, -80}, {0, -80}, {0, 0}}, color = {0, 0, 255},
thickness = 10),
    Text(origin = {140, -28}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-90, 41}, {-26, 99}}, textString = "1"),
    Text(origin = {76, -5}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-15, -1}, {29, -61}}, textString = "2")}));

end Merge_Leftwards;

//Merge system downwards direction (y axis)

model Merge_Downwards
  blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-64, 66}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {-80, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {66, 68}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {4, -74}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  StraightConveyor.StraightFix straightFix1(Length_straight = 1,
angle_straight = -90)  annotation(Placement(visible = true,
transformation(origin = {-64, 44}, extent = {{-10, -10}, {20, 10}}, rotation =
-90)));
  StraightConveyor.StraightFix straightFix2(Length_straight = 1,
angle_straight = -90) annotation(Placement(visible = true,
transformation(origin = {66, 46}, extent = {{-10, -10}, {20, 10}}, rotation =
-90)));
  StraightConveyor.StraightFix straightFix3(Length_straight = 1,
angle_straight = -90) annotation(Placement(visible = true,
transformation(origin = {4, -34}, extent = {{-10, -10}, {20, 10}}, rotation =
-90)));
  CurveConveyor.Curve_I90_O0 Curve_I90_O01 annotation(Placement(visible =
true, transformation(origin = {-64, 0}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  CurveConveyor.Curve_I90_O180 Curve_I90_O1801 annotation(Placement(visible =
true, transformation(origin = {66, 0}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
```

```
  StraightConveyor.StraightFix straightFix4(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {-42, 0}, extent
= {{-10, -10}, {20, 10}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix5(Length_straight = 1,
angle_straight = 180)  annotation(Placement(visible = true,
transformation(origin = {46, 0}, extent = {{-10, -10}, {20, 10}}, rotation =
180)));
  MergeSquare.MergeSQR_I0_I180_O270 MergeSQR_I0_I180_O2701
annotation(Placement(visible = true, transformation(origin = {4, -2}, extent =
{{-10, -10}, {10, 10}}, rotation = 0)));
equation
  connect(straightFix5.blockInterfaceOut,
MergeSQR_I0_I180_O2701.blockInterfaceIn2) annotation(Line(points = {{22, 0},
{12, 0}, {12, -2}, {12, -2}}));
  connect(Curve_I90_O1801.blockInterfaceOut, straightFix5.blockInterfaceIn)
annotation(Line(points = {{58, 0}, {48, 0}, {48, 0}, {50, 0}}));
  connect(straightFix2.blockInterfaceOut, Curve_I90_O1801.blockInterfaceIn)
annotation(Line(points = {{66, 24}, {66, 24}, {66, 8}, {66, 8}}));
  connect(blockInterfaceIn2, straightFix2.blockInterfaceIn)
annotation(Line(points = {{66, 68}, {66, 68}, {66, 50}, {66, 50}}));
  connect(straightFix3.blockInterfaceOut, blockInterfaceOut)
annotation(Line(points = {{4, -56}, {6, -56}, {6, -74}, {4, -74}}));
  connect(MergeSQR_I0_I180_O2701.blockInterfaceOut,
straightFix3.blockInterfaceIn) annotation(Line(points = {{4, -10}, {4, -10},
{4, -30}, {4, -30}}));
  connect(straightFix4.blockInterfaceOut,
MergeSQR_I0_I180_O2701.blockInterfaceIn1) annotation(Line(points = {{-18, 0},
{-4, 0}, {-4, -2}, {-4, -2}}));
  connect(Curve_I90_O01.blockInterfaceOut, straightFix4.blockInterfaceIn)
annotation(Line(points = {{-56, 0}, {-44, 0}, {-44, 0}, {-46, 0}}));
  connect(straightFix1.blockInterfaceOut, Curve_I90_O01.blockInterfaceIn)
annotation(Line(points = {{-64, 20}, {-64, 20}, {-64, 8}, {-64, 8}}));
  connect(blockInterfaceIn1, straightFix1.blockInterfaceIn)
annotation(Line(points = {{-64, 66}, {-62, 66}, {-62, 48}, {-64, 48}}));

  annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
  Icon(coordinateSystem(initialScale = 0.1), graphics = {
    Line(points = {{0, -90}, {0, 0}, {-80, 0}, {-80, 90}}, color = {0, 0,
255}, thickness = 10),
    Line(points = {{0, 0}, {80, 0}, {80, 80}}, color = {0, 0, 255}, thickness
= 10),
    Text(origin = {12, 12}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-90, 41}, {-26, 99}}, textString = "1"),
    Text(origin = {40, 111}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-15, -1}, {29, -61}}, textString = "2")}));

end Merge_Downwards;

//Merge system upwards direction (y axis)

model Merge_Upwards
  blockInterface blockInterfaceOut annotation(Placement(visible = true,
transformation(origin = {-2, 70}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {0, 80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceIn1 annotation(Placement(visible = true,
transformation(origin = {-76, -70}, extent = {{-10, -10}, {10, 10}}, rotation
= 0), iconTransformation(origin = {-80, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  blockInterface blockInterfaceIn2 annotation(Placement(visible = true,
transformation(origin = {72, -72}, extent = {{-10, -10}, {10, 10}}, rotation =
0), iconTransformation(origin = {80, -80}, extent = {{-20, -20}, {20, 20}},
rotation = 0)));
  StraightConveyor.StraightFix straightFix1(Length_straight = 1,
angle_straight = 90) annotation(Placement(visible = true,
transformation(origin = {-76, -50}, extent = {{-10, -10}, {20, 10}}, rotation
= 90)));
```

```
  StraightConveyor.StraightFix straightFix2(Length_straight = 1,
angle_straight = 90) annotation(Placement(visible = true,
transformation(origin = {72, -50}, extent = {{-10, -10}, {20, 10}}, rotation =
90)));
  StraightConveyor.StraightFix straightFix3(Length_straight = 1,
angle_straight = 90) annotation(Placement(visible = true,
transformation(origin = {-2, 24}, extent = {{-10, -10}, {20, 10}}, rotation =
90)));
  CurveConveyor.Curve_I270_O0 Curve_I270_O01 annotation(Placement(visible =
true, transformation(origin = {-77, -3}, extent = {{-15, -15}, {15, 15}},
rotation = 0)));
  CurveConveyor.Curve_I270_O180 Curve_I270_O1801 annotation(Placement(visible
= true, transformation(origin = {71, -3}, extent = {{-15, -15}, {15, 15}},
rotation = 0)));
  StraightConveyor.StraightFix straightFix4(Length_straight = 1,
angle_straight = 0) annotation(Placement(visible = true, transformation(origin
= {-48, -4}, extent = {{-10, -10}, {20, 10}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix5(Length_straight = 1,
angle_straight = 180) annotation(Placement(visible = true,
transformation(origin = {44, -4}, extent = {{-10, -10}, {20, 10}}, rotation =
180)));
  MergeSquare.MergeSQR_I0_I180_O90 MergeSQR_I0_I180_O901
annotation(Placement(visible = true, transformation(origin = {-2, -4}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
equation
  connect(straightFix3.blockInterfaceOut, blockInterfaceOut)
annotation(Line(points = {{-2, 48}, {0, 48}, {0, 70}, {-2, 70}}));
  connect(MergeSQR_I0_I180_O901.blockInterfaceOut,
straightFix3.blockInterfaceIn) annotation(Line(points = {{-2, 4}, {-2, 4}, {-
2, 22}, {-2, 22}}));
  connect(straightFix5.blockInterfaceOut,
MergeSQR_I0_I180_O901.blockInterfaceIn2) annotation(Line(points = {{20, -4},
{4, -4}, {4, -4}, {6, -4}}));
  connect(Curve_I270_O1801.blockInterfaceOut, straightFix5.blockInterfaceIn)
annotation(Line(points = {{58, -2}, {48, -2}, {48, -4}, {48, -4}}));
  connect(straightFix2.blockInterfaceOut, Curve_I270_O1801.blockInterfaceIn)
annotation(Line(points = {{72, -26}, {72, -26}, {72, -16}, {72, -16}}));
  connect(blockInterfaceIn2, straightFix2.blockInterfaceIn)
annotation(Line(points = {{72, -72}, {72, -72}, {72, -52}, {72, -52}}));
  connect(straightFix4.blockInterfaceOut,
MergeSQR_I0_I180_O901.blockInterfaceIn1) annotation(Line(points = {{-24, -4},
{-10, -4}, {-10, -4}, {-10, -4}}));
  connect(Curve_I270_O01.blockInterfaceOut, straightFix4.blockInterfaceIn)
annotation(Line(points = {{-64, -4}, {-50, -4}, {-50, -4}, {-52, -4}}));
  connect(straightFix1.blockInterfaceOut, Curve_I270_O01.blockInterfaceIn)
annotation(Line(points = {{-76, -26}, {-78, -26}, {-78, -16}, {-78, -16}}));
  connect(blockInterfaceIn1, straightFix1.blockInterfaceIn)
annotation(Line(points = {{-76, -70}, {-76, -70}, {-76, -54}, {-76, -54}}));
 annotation(Diagram(coordinateSystem(extent = {{-100, -100}, {100, 100}},
preserveAspectRatio = true, initialScale = 0.1, grid = {2, 2})),
 Icon(coordinateSystem(initialScale = 0.1), graphics = {
  Line(points = {{0, 90}, {0, 0}, {-80, 0}, {-80, -90}}, color = {0, 0, 255},
thickness = 10),
  Line(points = {{0, 0}, {80, 0}, {80, -80}}, color = {0, 0, 255}, thickness =
10),
  Text(origin = {16, -148}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-90, 41}, {-26, 99}}, textString = "1"),
  Text(origin = {36, -45}, fillColor = {255, 255, 255}, fillPattern =
FillPattern.Solid, extent = {{-15, -1}, {29, -61}}, textString = "2")}));
  end Merge_Upwards;end MergeSystem;
```

Position1 to position 2 standards

```
package Pos1_Pos2

  //From point 1 to 2 with an L shape

model L_Form
  Sink.FixedSink fixedSink1(SinkPosX = 100, SinkPosY = -90)
annotation(Placement(visible = true, transformation(origin = {72, -44}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  CurveConveyor.Curve_I90_O0 Curve_I90_O01 annotation(Placement(visible =
true, transformation(origin = {-68, -44}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  Source.FixSource fixSource1 annotation(Placement(visible = true,
transformation(origin = {-68, 16}, extent = {{-10, -10}, {10, 10}}, rotation =
-90)));
  StraightConveyor.Straight_VariableLength
straight_VariableLength1(angle_straight = -90)  annotation(Placement(visible =
true, transformation(origin = {-68, -12}, extent = {{-10, -10}, {10, 10}},
rotation = -90)));
  StraightConveyor.Straight_VariableLength straight_VariableLength2
annotation(Placement(visible = true, transformation(origin = {4, -46}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  equation
    connect(straight_VariableLength2.blockInterfaceOut,
fixedSink1.blockInterface1) annotation(Line(points = {{12, -46}, {72, -46},
{72, -44}, {72, -44}}));
    connect(Curve_I90_O01.blockInterfaceOut,
straight_VariableLength2.blockInterfaceIn) annotation(Line(points = {{-60, -
44}, {-4, -44}, {-4, -46}, {-4, -46}}));
    connect(straight_VariableLength1.blockInterfaceOut,
Curve_I90_O01.blockInterfaceIn) annotation(Line(points = {{-68, -20}, {-68, -
20}, {-68, -36}, {-68, -36}}));
    connect(fixSource1.interfaceOut,
straight_VariableLength1.blockInterfaceIn) annotation(Line(points = {{-68,
16}, {-68, 16}, {-68, -4}, {-68, -4}}));

end L_Form;

  //From point 1 to 2 with an reverse L shape

model Inverted_L_Form
  Source.FixSource fixSource1 annotation(Placement(visible = true,
transformation(origin = {-70, -36}, extent = {{-10, -10}, {10, 10}}, rotation
= 0)));
  Sink.FixedSink fixedSink1(SinkPosX = 100, SinkPosY = 100)
annotation(Placement(visible = true, transformation(origin = {36, 36}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.Straight_VariableLength
straight_VariableLength1(angle_straight = 90)  annotation(Placement(visible =
true, transformation(origin = {-70, 0}, extent = {{-10, -10}, {10, 10}},
rotation = 90)));
  CurveConveyor.Curve_I270_O0 Curve_I270_O01 annotation(Placement(visible =
true, transformation(origin = {-68, 38}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  StraightConveyor.Straight_VariableLength straight_VariableLength2
annotation(Placement(visible = true, transformation(origin = {-14, 38}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
equation
  connect(straight_VariableLength2.blockInterfaceOut,
fixedSink1.blockInterface1) annotation(Line(points = {{-6, 38}, {36, 38}, {36,
36}, {36, 36}}));
  connect(Curve_I270_O01.blockInterfaceOut,
straight_VariableLength2.blockInterfaceIn) annotation(Line(points = {{-60,
38}, {-22, 38}, {-22, 38}, {-22, 38}}));
  connect(straight_VariableLength1.blockInterfaceOut,
Curve_I270_O01.blockInterfaceIn) annotation(Line(points = {{-70, 8}, {-68, 8},
{-68, 30}, {-68, 30}}));
```

```
  connect(fixSource1.interfaceOut, straight_VariableLength1.blockInterfaceIn)
annotation(Line(points = {{-70, -36}, {-70, -36}, {-70, -8}, {-70, -8}}));

end Inverted_L_Form;

  //From point 1 to 2 with an S shape

model S_Form
  Sink.FixedSink fixedSink1(SinkPosY = 100)  annotation(Placement(visible =
true, transformation(origin = {66, 48}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  CurveConveyor.Curve_I270_O0 Curve_I270_O01 annotation(Placement(visible =
true, transformation(origin = {-10, 46}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  StraightConveyor.StraightFix straightFix1(Length_straight = 50)
annotation(Placement(visible = true, transformation(origin = {28, 46}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));

  Source.FixSource fixSource1 annotation(Placement(visible = true,
transformation(origin = {-76, -24}, extent = {{-10, -10}, {10, 10}}, rotation
= 0)));
  StraightConveyor.Straight_VariableLength straight_VariableLength1
annotation(Placement(visible = true, transformation(origin = {-44, -24},
extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.Straight_VariableLength
straight_VariableLength2(angle_straight = 90)  annotation(Placement(visible =
true, transformation(origin = {-10, 12}, extent = {{-10, -10}, {10, 10}},
rotation = 90)));
  CurveConveyor.Curve_I180_O90 curve_I180_O901 annotation(Placement(visible =
true, transformation(origin = {-12, -24}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  equation
    connect(straight_VariableLength1.blockInterfaceOut,
curve_I180_O901.blockInterfaceIn) annotation(Line(points = {{-36, -24}, {-20,
-24}, {-20, -24}, {-20, -24}}));
    connect(curve_I180_O901.blockInterfaceOut,
straight_VariableLength2.blockInterfaceIn) annotation(Line(points = {{-12, -
16}, {-10, -16}, {-10, 4}, {-10, 4}}));

  connect(straight_VariableLength2.blockInterfaceOut,
Curve_I270_O01.blockInterfaceIn) annotation(Line(points = {{-10, 20}, {-10,
38}}));
    connect(Curve_I270_O01.blockInterfaceOut, straightFix1.blockInterfaceIn)
annotation(Line(points = {{-2, 46}, {20, 46}}));

    connect(fixSource1.interfaceOut,
straight_VariableLength1.blockInterfaceIn) annotation(Line(points = {{-76, -
24}, {-52, -24}, {-52, -24}, {-52, -24}}));
    connect(straightFix1.blockInterfaceOut, fixedSink1.blockInterface1)
annotation(Line(points = {{36, 46}, {58, 46}, {58, 48}, {58, 48}}));
end S_Form;


  //From point 1 to 2 with an reversed L shape

model Inverted_S_Form
  Source.FixSource fixSource1 annotation(Placement(visible = true,
transformation(origin = {-76, -64}, extent = {{-10, -10}, {10, 10}}, rotation
= 0)));
  Sink.FixedSink fixedSink1(SinkPosY = 100)  annotation(Placement(visible =
true, transformation(origin = {76, 80}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  StraightConveyor.Straight_VariableLength
straight_VariableLength1(angle_straight = 90)  annotation(Placement(visible =
true, transformation(origin = {-76, -28}, extent = {{-10, -10}, {10, 10}},
rotation = 90)));
```

```
  CurveConveyor.Curve_I270_O0 Curve_I270_O01 annotation(Placement(visible =
true, transformation(origin = {-76, 8}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  StraightConveyor.Straight_VariableLength straight_VariableLength2
annotation(Placement(visible = true, transformation(origin = {2, 6}, extent =
{{-10, -10}, {10, 10}}, rotation = 0)));

  StraightConveyor.StraightFix straightFix1(Length_straight = 50,
angle_straight = 90)  annotation(Placement(visible = true,
transformation(origin = {76, 32}, extent = {{-10, -10}, {20, 10}}, rotation =
90)));
  CurveConveyor.Curve_I180_O90 curve_I180_O901 annotation(Placement(visible =
true, transformation(origin = {74, 4}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));

equation
    connect(curve_I180_O901.blockInterfaceOut, straightFix1.blockInterfaceIn)
annotation(Line(points = {{74, 12}, {76, 12}, {76, 30}, {76, 30}}));
    connect(straight_VariableLength2.blockInterfaceOut,
curve_I180_O901.blockInterfaceIn) annotation(Line(points = {{10, 6}, {66, 6},
{66, 4}, {66, 4}}));
  connect(fixSource1.interfaceOut, straight_VariableLength1.blockInterfaceIn)
annotation(Line(points = {{-76, -64}, {-76, -36}}));
  connect(straightFix1.blockInterfaceOut, fixedSink1.blockInterface1)
annotation(Line(points = {{76, 56}, {76, 56}, {76, 80}, {76, 80}}));


  connect(Curve_I270_O01.blockInterfaceOut,
straight_VariableLength2.blockInterfaceIn) annotation(Line(points = {{-68, 8},
{-6, 8}, {-6, 6}, {-6, 6}}));
  connect(straight_VariableLength1.blockInterfaceOut,
Curve_I270_O01.blockInterfaceIn) annotation(Line(points = {{-76, -20}, {-78, -
20}, {-78, 0}, {-76, 0}}));

end Inverted_S_Form;


  //From point 1 to 2 with a straight line

model DirectConnection_Form
  Sink.FixedSink fixedSink1(SinkPosY = 100)  annotation(Placement(visible =
true, transformation(origin = {86, 48}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  Source.FixSource fixSource1 annotation(Placement(visible = true,
transformation(origin = {-80, -18}, extent = {{-10, -10}, {10, 10}}, rotation
= 0)));
  StraightConveyor.straightConveyor_Variable straightConveyor_Variable1
annotation(Placement(visible = true, transformation(origin = {4, 8}, extent =
{{-10, -10}, {10, 10}}, rotation = 0)));
  equation
    connect(straightConveyor_Variable1.blockInterfaceOut,
fixedSink1.blockInterface1) annotation(Line(points = {{12, 8}, {86, 8}, {86,
48}}));
    connect(fixSource1.interfaceOut,
straightConveyor_Variable1.blockInterfaceIn) annotation(Line(points = {{-80, -
18}, {-4, -18}, {-4, 8}, {-4, 8}}));
end DirectConnection_Form;

end Pos1_Pos2;
```

Simple System

```
model SimpleSystem
  Source.FixSource fixSource1 annotation(Placement(visible = true,
transformation(origin = {-88, -8}, extent = {{-10, -10}, {10, 10}}, rotation =
0)));
  StraightConveyor.StraightFix Entrance(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {-66, -8}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  BranchSystem.Branch branch1 annotation(Placement(visible = true,
transformation(origin = {-17, -9}, extent = {{-31, -31}, {31, 31}}, rotation =
0)));
  StraightConveyor.StraightFix straightFix2(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {34, 16}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix3(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {34, -34}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.StraightFix Exit(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {132, -10},
extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  Sink.VariableSink variableSink1 annotation(Placement(visible = true,
transformation(origin = {160, -10}, extent = {{-10, -10}, {10, 10}}, rotation
= 0)));
  MergeSystem.Merge_Rightwards merge_Rightwards1 annotation(Placement(visible
= true, transformation(origin = {88, -10}, extent = {{-32, -32}, {32, 32}},
rotation = 0)));
equation
  connect(Exit.blockInterfaceOut, variableSink1.blockInterface1)
annotation(Line(points = {{144, -10}, {160, -10}}));
  connect(merge_Rightwards1.blockInterfaceOut, Exit.blockInterfaceIn)
annotation(Line(points = {{114, -10}, {127, -10}}));
  connect(fixSource1.interfaceOut, Entrance.blockInterfaceIn)
annotation(Line(points = {{-80, -8}, {-71, -8}}));
  connect(Entrance.blockInterfaceOut, branch1.blockInterfaceIn)
annotation(Line(points = {{-54, -8}, {-42, -8}}));
  connect(branch1.blockInterfaceOut2, straightFix3.blockInterfaceIn)
annotation(Line(points = {{8, -34}, {29, -34}}));
  connect(straightFix3.blockInterfaceOut, merge_Rightwards1.blockInterfaceIn2)
annotation(Line(points = {{46, -34}, {62, -34}, {62, -36}}));
  connect(branch1.blockInterfaceOut1, straightFix2.blockInterfaceIn)
annotation(Line(points = {{8, 16}, {29, 16}}));
  connect(straightFix2.blockInterfaceOut, merge_Rightwards1.blockInterfaceIn1)
annotation(Line(points = {{46, 16}, {62, 16}}));
  annotation(Diagram(coordinateSystem(initialScale = 0, extent = {{-100, -
100}, {200, 100}})), Icon(coordinateSystem(initialScale = 0, extent = {{-100,
-100}, {200, 100}})), version = "", uses);
end SimpleSystem;
```

System: two sources, two sinks

```
model TwoSources_TwoSinks
  Source.FixSource fixSource2(paraThroughPut = 200)
annotation(Placement(visible = true, transformation(origin = {-133, -35},
extent = {{-17, -17}, {17, 17}}, rotation = 0)));
  BranchSystem.Branch branch1 annotation(Placement(visible = true,
transformation(origin = {-66, -36}, extent = {{-42, -42}, {42, 42}}, rotation
= 0)));
  StraightConveyor.StraightFix straightFix2(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {100, 30}, extent
= {{-20, -20}, {20, 20}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix3(Length_straight = 1)
annotation(Placement(visible = true, transformation(origin = {5, -69}, extent
= {{-21, -21}, {21, 21}}, rotation = 0)));
```

```
  BranchSystem.Branch branch2 annotation(Placement(visible = true,
transformation(origin = {80, -68}, extent = {{-42, -42}, {42, 42}}, rotation =
0)));
  CurveConveyor.Curve_I180_O270 Curve_I180_O2701 annotation(Placement(visible
= true, transformation(origin = {250, -2}, extent = {{-28, -28}, {28, 28}},
rotation = 0)));
  Source.FixSource fixSource1(paraPosY = 7.5)  annotation(Placement(visible =
true, transformation(origin = {-129, 61}, extent = {{-19, -19}, {19, 19}},
rotation = 0)));
  StraightConveyor.StraightFix straightFix1(Length_straight = 3)
annotation(Placement(visible = true, transformation(origin = {-73.6271,
62.2203}, extent = {{-9.48021, -14.2203}, {18.9604, 14.2203}}, rotation =
0)));
  Sink.VariableSink variableSink1 annotation(Placement(visible = true,
transformation(origin = {151, -103}, extent = {{-19, -19}, {19, 19}}, rotation
= 0)));
  Sink.VariableSink variableSink2 annotation(Placement(visible = true,
transformation(origin = {249, -67}, extent = {{-19, -19}, {19, 19}}, rotation
= 0)));
  MergeSystem.Merge_Rightwards merge_Rightwards1 annotation(Placement(visible
= true, transformation(origin = {32, 28}, extent = {{-42, -42}, {42, 42}},
rotation = 0)));
  MergeSystem.Merge_Rightwards merge_Rightwards2 annotation(Placement(visible
= true, transformation(origin = {172, -4}, extent = {{-42, -42}, {42, 42}},
rotation = 0)));
equation
  connect(fixSource1.interfaceOut, straightFix1.blockInterfaceIn)
annotation(Line(points = {{-129, 61}, {-95, 61}, {-95, 62}, {-76, 62}}));
  connect(straightFix1.blockInterfaceOut, merge_Rightwards1.blockInterfaceIn1)
annotation(Line(points = {{-52, 62}, {-2, 62}}));
  connect(branch1.blockInterfaceOut1, merge_Rightwards1.blockInterfaceIn2)
annotation(Line(points = {{-32, -2}, {-4, -2}, {-4, -6}, {-2, -6}}));
  connect(merge_Rightwards2.blockInterfaceOut,
Curve_I180_O2701.blockInterfaceIn) annotation(Line(points = {{206, -4}, {228,
-4}, {228, -2}, {228, -2}}));
  connect(branch2.blockInterfaceOut1, merge_Rightwards2.blockInterfaceIn2)
annotation(Line(points = {{114, -34}, {138, -34}, {138, -38}, {138, -38}}));
  connect(straightFix2.blockInterfaceOut, merge_Rightwards2.blockInterfaceIn1)
annotation(Line(points = {{124, 30}, {138, 30}, {138, 30}, {138, 30}}));
  connect(merge_Rightwards1.blockInterfaceOut, straightFix2.blockInterfaceIn)
annotation(Line(points = {{66, 28}, {90, 28}, {90, 30}, {90, 30}}));
  connect(fixSource2.interfaceOut, branch1.blockInterfaceIn)
annotation(Line(points = {{-133, -35}, {-119, -35}, {-119, -36}, {-100, -
36}}));
  connect(branch1.blockInterfaceOut2, straightFix3.blockInterfaceIn)
annotation(Line(points = {{-32, -70}, {-21, -70}, {-21, -69}, {-12, -69}}));
  connect(Curve_I180_O2701.blockInterfaceOut, variableSink2.blockInterface1)
annotation(Line(points = {{250, -24}, {248, -24}, {248, -68}, {250, -68}}));
  connect(branch2.blockInterfaceOut2, variableSink1.blockInterface1)
annotation(Line(points = {{114, -102}, {152, -102}, {152, -104}, {152, -
104}}));
  connect(straightFix3.blockInterfaceOut, branch2.blockInterfaceIn)
annotation(Line(points = {{22, -69}, {37, -69}, {37, -68}, {46, -68}}));
  annotation(Diagram(coordinateSystem(extent = {{-150, -150}, {350, 100}})),
Icon(coordinateSystem(extent = {{-150, -150}, {350, 100}})), version = "",
uses);
end TwoSources_TwoSinks;
```

Warehouse

```
model Warehouse
  Source.FixSource fixSource1(paraPosY = 100) annotation(Placement(visible =
true, transformation(origin = {-90, 68}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  Source.FixSource fixSource2(paraPosY = 50) annotation(Placement(visible =
true, transformation(origin = {-90, 0}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
```

```
  Source.FixSource fixSource3 annotation(Placement(visible = true,
transformation(origin = {-90, -76}, extent = {{-10, -10}, {10, 10}}, rotation
= 0)));
  Sink.FixedSink fixedSink1(SinkPosX = 50) annotation(Placement(visible =
true, transformation(origin = {90, -76}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  Sink.FixedSink fixedSink2(SinkPosX = 50, SinkPosY = 50)
annotation(Placement(visible = true, transformation(origin = {90, -2}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  Sink.FixedSink fixedSink3(SinkPosX = 50, SinkPosY = 100)
annotation(Placement(visible = true, transformation(origin = {90, 66}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix1(Length_straight = 5)
annotation(Placement(visible = true, transformation(origin = {-70, 68}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix2(Length_straight = 5)
annotation(Placement(visible = true, transformation(origin = {-72, 0}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix3(Length_straight = 5)
annotation(Placement(visible = true, transformation(origin = {-70, -76},
extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  MergeSquare.MergeSQR_I180_I90_O270 MergeSQR_I180_I90_O2701
annotation(Placement(visible = true, transformation(origin = {-44, 0}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  MergeSquare.MergeSQR_I90_I180_O0 MergeSQR_I90_I180_O01
annotation(Placement(visible = true, transformation(origin = {-44, -76},
extent = {{-10, -10}, {10, 10}}, rotation = 0)));
  BranchSquare.BranchSQR_I180_O0_O90 BranchSQR_I180_O0_O901
annotation(Placement(visible = true, transformation(origin = {38, -76}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  BranchSquare.BranchSQR_I270_O0_O90 BranchSQR_I270_O0_O901
annotation(Placement(visible = true, transformation(origin = {38, -2}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  CurveConveyor.Curve_I180_O270 Curve_I180_O2701 annotation(Placement(visible
= true, transformation(origin = {-44, 68}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  CurveConveyor.Curve_I270_O0 Curve_I270_O01 annotation(Placement(visible =
true, transformation(origin = {38, 66}, extent = {{-10, -10}, {10, 10}},
rotation = 0)));
  StraightConveyor.StraightFix straightFix7(Length_straight = 48.5,
angle_straight = -90) annotation(Placement(visible = true,
transformation(origin = {-44, 36}, extent = {{-10, -10}, {10, 10}}, rotation =
-90)));
  StraightConveyor.StraightFix straightFix8(Length_straight = 49,
angle_straight = -90) annotation(Placement(visible = true,
transformation(origin = {-44, -40}, extent = {{-10, -10}, {10, 10}}, rotation
= -90)));
  StraightConveyor.StraightFix straightFix9(Length_straight = 40)
annotation(Placement(visible = true, transformation(origin = {-6, -76}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.StraightFix straightFix10(Length_straight = 49,
angle_straight = 90) annotation(Placement(visible = true,
transformation(origin = {38, -38}, extent = {{-10, -10}, {10, 10}}, rotation =
90)));
  StraightConveyor.StraightFix straightFix11(Length_straight = 48.5,
angle_straight = 90) annotation(Placement(visible = true,
transformation(origin = {38, 28}, extent = {{-10, -10}, {10, 10}}, rotation =
90)));
  StraightConveyor.straightConveyor_Variable straightConveyor_Variable1
annotation(Placement(visible = true, transformation(origin = {64, -76}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.straightConveyor_Variable straightConveyor_Variable2
annotation(Placement(visible = true, transformation(origin = {64, -2}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
  StraightConveyor.straightConveyor_Variable straightConveyor_Variable3
annotation(Placement(visible = true, transformation(origin = {64, 66}, extent
= {{-10, -10}, {10, 10}}, rotation = 0)));
equation
```

```
  connect(straightConveyor_Variable3.blockInterfaceOut,
fixedSink3.blockInterface1) annotation(Line(points = {{72, 66}, {90, 66}}));
  connect(Curve_I270_O01.blockInterfaceOut,
straightConveyor_Variable3.blockInterfaceIn) annotation(Line(points = {{46,
66}, {56, 66}}));
  connect(straightConveyor_Variable2.blockInterfaceOut,
fixedSink2.blockInterface1) annotation(Line(points = {{72, -2}, {92, -2}, {92,
-2}, {90, -2}}));
  connect(BranchSQR_I270_O0_O901.blockInterfaceOut1,
straightConveyor_Variable2.blockInterfaceIn) annotation(Line(points = {{46, -
2}, {56, -2}, {56, -2}, {56, -2}}));
  connect(straightConveyor_Variable1.blockInterfaceOut,
fixedSink1.blockInterface1) annotation(Line(points = {{72, -76}, {92, -76},
{92, -76}, {90, -76}}));
  connect(BranchSQR_I180_O0_O901.blockInterfaceOut2,
straightConveyor_Variable1.blockInterfaceIn) annotation(Line(points = {{46, -
76}, {58, -76}, {58, -76}, {56, -76}}));
  connect(Curve_I180_O2701.blockInterfaceOut, straightFix7.blockInterfaceIn)
annotation(Line(points = {{-44, 60}, {-44, 41}}));
  connect(straightFix7.blockInterfaceOut,
MergeSQR_I180_I90_O2701.blockInterfaceIn2) annotation(Line(points = {{-44,
24}, {-44, 8}}));
  connect(BranchSQR_I270_O0_O901.blockInterfaceOut2,
straightFix11.blockInterfaceIn) annotation(Line(points = {{38, 6}, {38,
23}}));
  connect(straightFix11.blockInterfaceOut, Curve_I270_O01.blockInterfaceIn)
annotation(Line(points = {{38, 40}, {38, 58}}));
  connect(fixSource2.interfaceOut, straightFix2.blockInterfaceIn)
annotation(Line(points = {{-82, 0}, {-77, 0}}));
  connect(straightFix2.blockInterfaceOut,
MergeSQR_I180_I90_O2701.blockInterfaceIn1) annotation(Line(points = {{-60, 0},
{-52, 0}}));
  connect(fixSource3.interfaceOut, straightFix3.blockInterfaceIn)
annotation(Line(points = {{-82, -76}, {-75, -76}}));
  connect(straightFix3.blockInterfaceOut,
MergeSQR_I90_I180_O01.blockInterfaceIn2) annotation(Line(points = {{-58, -76},
{-52, -76}}));
  connect(fixSource1.interfaceOut, straightFix1.blockInterfaceIn)
annotation(Line(points = {{-82, 68}, {-75, 68}}));
  connect(straightFix1.blockInterfaceOut, Curve_I180_O2701.blockInterfaceIn)
annotation(Line(points = {{-58, 68}, {-52, 68}}));
  connect(straightFix10.blockInterfaceOut,
BranchSQR_I270_O0_O901.blockInterfaceIn) annotation(Line(points = {{38, -30},
{38, -30}, {38, -10}, {38, -10}}));
  connect(BranchSQR_I180_O0_O901.blockInterfaceOut1,
straightFix10.blockInterfaceIn) annotation(Line(points = {{38, -68}, {38, -
68}, {38, -46}, {38, -46}}));
  connect(straightFix9.blockInterfaceOut,
BranchSQR_I180_O0_O901.blockInterfaceIn) annotation(Line(points = {{2, -76},
{30, -76}, {30, -76}, {30, -76}}));
  connect(MergeSQR_I90_I180_O01.blockInterfaceOut,
straightFix9.blockInterfaceIn) annotation(Line(points = {{-36, -76}, {-14, -
76}, {-14, -76}, {-14, -76}}));
  connect(straightFix8.blockInterfaceOut,
MergeSQR_I90_I180_O01.blockInterfaceIn1) annotation(Line(points = {{-44, -48},
{-44, -48}, {-44, -68}, {-44, -68}}));
  connect(MergeSQR_I180_I90_O2701.blockInterfaceOut,
straightFix8.blockInterfaceIn) annotation(Line(points = {{-44, -8}, {-44, -8},
{-44, -32}, {-44, -32}}));
end Warehouse;
```

REFERENCES

[1] Material Handling Systems, Hong Kong University of Science and technology.

[2] College-Industry Council on Material handling Education.

[3] Single Row Layout Models, Keller, 2015.

[4] Continuous Conveyors. Schrage conveying systems, (http://www.schrage.de/en/schrage-informs/glossary/continuous-conveyor.html)

[5] Continuous conveyor glossary, Schrage conveying systems, (http://www.schrage.de/en/schrage-informs/glossary/continuous-conveyor.html)

[6] Basic queuing theory, Dr. János Sztrik.

[7] (Basic queuing theor;, Dr. János Sztrik, p. 12.

[8] Basic queuing theory, Dr. János Sztrik, p. 13.

[9] Material flow planning and calculation Part 6; Jodin, Trummer, Tinello; 2014, p 13-18

[10] Basic queuing theory, Dr. János Sztrik, p. 14

[11] Holste, scdigest, 2013, (http://www.scdigest.com/experts/Holste_13-06-19.php?cid=7162)

[12] Holste, scdigest, 2013, (http://www.scdigest.com/experts/Holste_13-06-26.php?cid=7186)

[13] Availability and Maintainability in Engineering Design, Rudolph Frederick Stapelberg, 2009, p. 296

[14] Zhang, RNA automation, 2014, (http://www.rnaautomation.com/blog/measure-effectiveness-production-line/)

[15] Belt Conveyors for Bulk Materials Calculations by CEMA 5th Edition, Kulinowski, p. 5.

[16] Types of conveyors. (https://es.slideshare.net/Thimarl1977/types-of-conveyors)

[17] Drive Solutions: Mechatronics for Production and Logistics, E. Kiel, 2008

[18] Parts and functions of a conveyor system, Dr. B. C. Paul, 2012.

[19] Budde Fördertechnik (http://www.budde-foerdertechnik.de/en/products/telescopic-conveyor-range.html)

[20] Conveyor Equipment Manufacturers Association

[21] FSI industries curve conveyor illustration http://www.fsindustries.com/more_info/gravity_roller_curve_conveyor_25in/gravity_roller_curve_conveyor_25in.shtml

[22] Schäfer loading/unloading area picture (http://www.manutencionyalmacenaje.com/es/notices/2014/09/almacen-automatico-a-gran-altura-de-ssi-schaefer-para-congelados-37394.php#.WNuFWW_yjIV)

[23] Eroski Logistic center picture (http://www.inbisanoticias.com/inbisabyco.php?id=227)

[24] Criteria for the selection of a programming language for introductory courses; Parker, Ottaway. 2006

[25] Building Logistics Networks Using Model-Based Reasoning Techniques. p 414