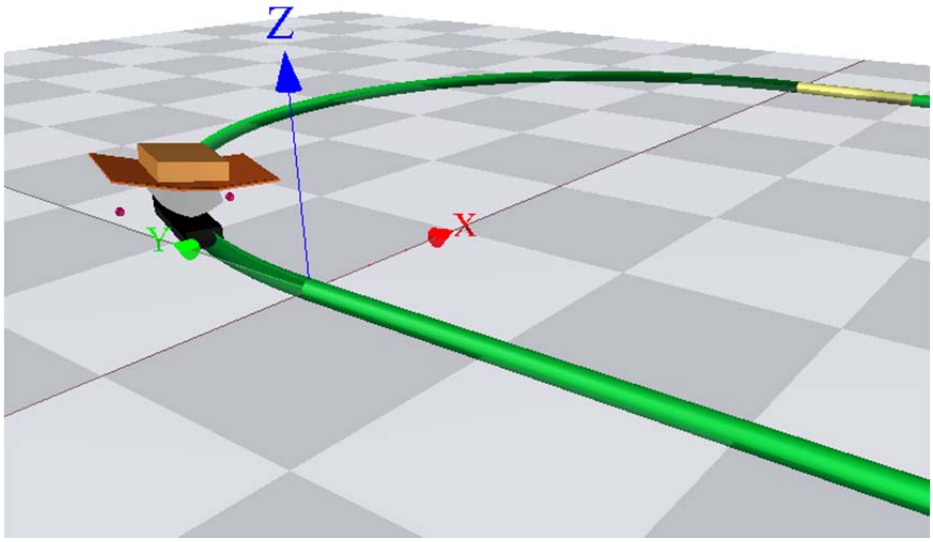
APPLYING CO-SIMULATION FOR AN INDUSTRIAL CONVEYOR SYSTEM

Electrical and Computer Engineering
Technical Report ECE-TR-5



AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

DATA SHEET

Title: Applying Co-Simulation for an Industrial Conveyor System
Subtitle: Electrical and Computer Engineering
Series title and no.: Technical Report ECE-TR-5

Authors: Kim Bjerge and Peter Gorm Larsen
Department of Engineering – Electrical and Computer Engineering,
Aarhus University

Internet version: The report is available in electronic format (pdf) at the Department of Engineering website http://www.eng.au.dk.

Publisher: Aarhus University©
URL: http://www.eng.au.dk

Year of publication: 2012 **Pages**: 39
Editing completed: October 2011

Abstract: This paper describes an industrial application of a new research technology enabling the co-simulation of models in continuous time and discrete event respectively. The application concerns modeling of a conveyor system with trolleys that has tilting capabilities that can be used to compensate for high speeds in curves in order to avoid parcels falling of the trolleys. The main challenge for this kind of physical system is that a system solution here requires both insight into the mechanical physics behavior as well as ways in which the system can be controlled discretely by a software based solution. This paper demonstrates how it is possible to bridge the gap between these two different disciplines in co-simulated models.

Keywords: VDM, SysML, co-simulation, industrial application, conveyor system

Financial support: Partially supported by the EU DESTECS project

Please cite as: Kim Bjerge & Larsen, P.G., 2012. Applying Co-Simulation for an Industrial Conveyor System. Department of Engineering, Aarhus University. Denmark. 39 pp. Technical Report ECE-TR-5

Cover image: Kim Bjerge

ISSN: 2245-2087

APPLYING CO-SIMULATION FOR AN INDUSTRIAL CONVEYOR SYSTEM

Kim Bjerge and Peter Gorm Larsen — Aarhus University, Department of Engineering

Abstract

This paper describes an industrial application of a new research technology enabling the cosimulation of models in continuous time and discrete event respectively. The application concerns modeling of a conveyor system with trolleys that has tilting capabilities that can be used to compensate for high speeds in curves in order to avoid parcels falling of the trolleys. The main challenge for this kind of physical system is that a system solution here requires both insight into the mechanical physics behaviour as well as ways in which the system can be controlled discretely by a software based solution. This paper demonstrates how it is possible to bridge the gap between these two different disciplines in co-simulated models.

Table of Contents

1

Introduction

When developing mechatronic control systems typically a *multidisciplinary* approach is advantageous. The embedded systems market is a rapidly evolving one, making it imperative that developers can conceive and evaluate designs quickly and with confidence. This is made all the more challenging by two factors. First, ever more demanding and interdependent requirements, including the need for reliability, fault tolerance, performance and interoperability. Second, the increasingly distributed character of embedded systems, which introduces a wider range of architectures – and potential or possible faults – for controllers. In the DESTECS[1] (Design Support and Tooling for Embedded Control Software) project an attempt for solving this challenge using co-simulation between co-models has been made. In this paper we describe how this new research technology can be applied to an industrial case study.

The concept explored in the DESTECS project contains a continuous-time model of a physical world that we would like to develop a controller for and a discrete event model of a software controller. The continuous time model is expressed using a tool called 20-sim with a semantic basis founded in Bond graphs. The discrete event model is expressed using the VDM (Vienna Development Method) [1, 2] notation with a semantic basis founded in set theory. These models are combined to form a co-model that can be simulated using co-simulation. For overviews of these co-models SysML [3] will be used. With the co-modelling approach it is possible to explore different design alternatives including adjusting physical design parameters and experimenting with alternative control strategies. This approach saves both time and money in creation of expensive physical models.

Because of tough competition in the market where the company this research have been conducted we are unfortunately not allowed to mention their name or the precise product that this work has been concerned with. However, we can say that it deals with a trolley conveyor system where it is possible to tilt the trolley. This functionality is used for example to move parcels on the trolleys out when they have reached their desired destination. However, in this work we are interested in the ability to tilt the trolleys in curves for the conveyor in order to prevent the centrifugal force to get parcels to slide off with high speeds. Thus the purpose of this research is to come up with a controls algorithm that can tilt the trolley to compensate for high speeds in curves and to be able to predict at what speed the parcels will start to fall of in a given configuration.

This paper is structured such that after this introduction Chapter 2 provides an introduction to the DESTECS technology used in this work. Afterwards Chapter 3 introduces the physical laws necessary to understand the subsequent model of the physical behaviour of the case study. This is

[1] www.destecs.org

followed by Chapters 4 and 5 which introduces the physical model and the controller model made for the case respectively. Then Chapter 6 explains how these two co-models are co-simulated and what conclusions that can be drawn from these. Afterwards Chapter 7 provides an overview of the related work. Finally Chapter 8 ends the paper with a collection of concluding remarks.

2

The DESTECS Technology

The DESTECS project [4] supports model-based approaches to the engineering of embedded control systems. Thus our basic concepts relate to models and their analysis by simulation. A model is a more or less abstract representation of a system or component of interest. For an embedded system, the model typically includes both the physical environment and its controller. The amount of abstraction is up to the modeller and depends on the purpose for which the model is being constructed. In this paper we build a model of a part of a physical conveyor system and the software that controls tilting the trolleys when they go through curves in order to avoid packages sliding on the trolleys. Here the purpose is to determine the best way of controlling such tilting of the trolleys under different conditions and this purpose is used to decide the abstractions that should be made in the modeling.

Models can be produced during requirements definition and analysis and during architectural design. Models persist throughout the development process. Once the design has been decided upon, the model of the plant forms a reference for the physical environment and plant. The model of the controller can be used as a specification for the controller and software. Therefore, these models play a role in validation and testing during the implementation of the design.

We build models in order to support various forms of analysis including static mathematical reasoning and simulation – the latter is our focus in DESTECS. A model that is capable of being run on a machine is termed executable, and an execution of an executable model is called a simulation. We focus on supporting models in which the controller and plant or environment are modelled using different specialised environments and tools. In particular, we support co-simulation by allowing the collaboration of two simulation engines in order to produce a coherent combined simulation of a model of a digital controller expressed in a discrete-event (DE) formalism and a model of the plant/environment expressed in a continuous-time (CT) notation.

Once models have been constructed, they can be evaluated by co-simulation. The comparative evaluation of models against criteria of interest to the developer, and hence the selection of the best candidate model for subsequent development as illustrated in Figure 2.1 termed design space exploration (DSE).

The design flow process starts with an initial concept and finishes with a realisation in hardware and software via a series of design choices. Each choice involves making a selection from alternatives on the basis of criteria that are important to the developer such as cost or performance. The alternative selected at each point constrains the range of design alternatives that may be viable next steps forward from the current position.

There are many design decisions as the design is detailed. Each decision adds detail to the design, restricts the design space and lowers the abstraction level. The variations between alter-

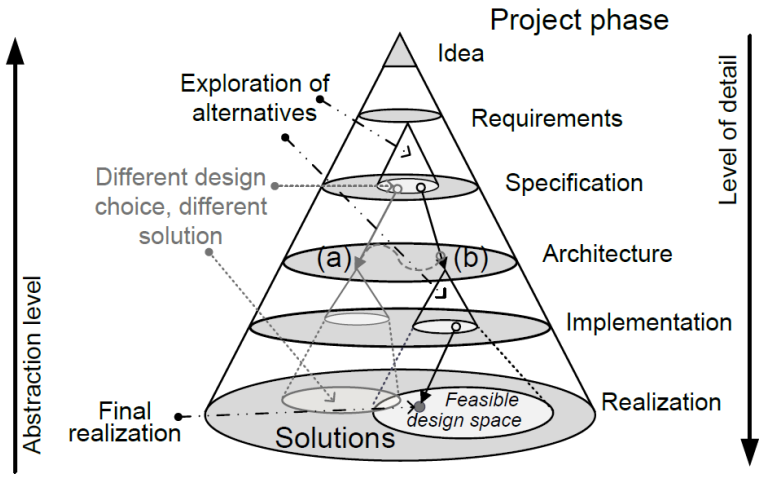


Figure 2.1: Design Cone with different abstraction levels

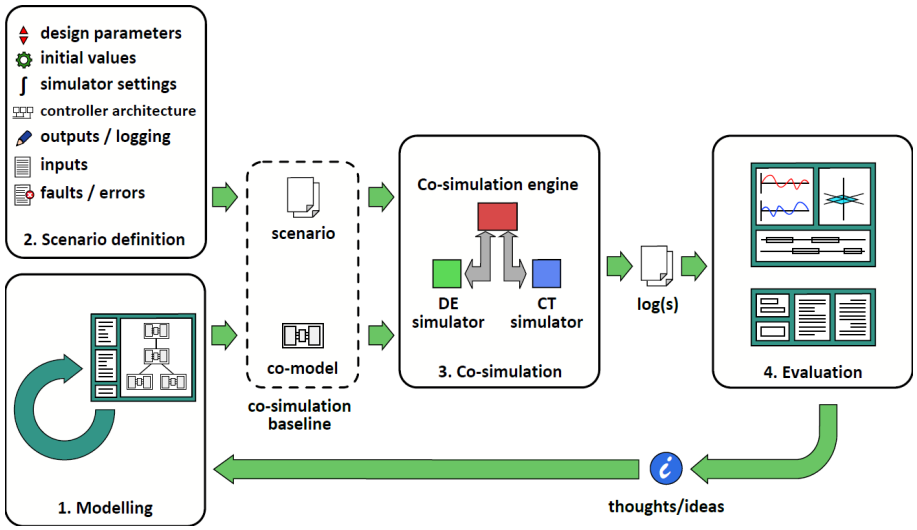


Figure 2.2: The process of design space exploration

native designs may be architectural or qualitative rather than limited to specific numeric design parameters.

The DSE process is illustrated in Figure 2.2. The first step of this (iterative) process is modelling. The engineer creates models reusing existing components and sub-models where possible. Inputs are defined allowing scenarios to be realised during co-simulation. The engineer may also model faulty behaviour, ideally supported by tools. Information about inputs and faults is added to the co-model interface making this behaviour configurable through scenarios. Contracts that defines variables and parameters to be exchanged during co-simulation are defined in forming co-models. These models and co-models populate the model base; model management techniques help the engineer to maintain this model base.

However, in this paper limited emphasis is put on the non-normative behaviour of the system since that was not a part of the main purpose of this investigation. However, this would be a natural next step combined with a more thougrough design space exploration.

2.1 Discrete Event Modelling

For the discrete event modelling side we use the *Vienna Development Method* (VDM) as the formalism for discrete-event models of controllers. VDM's origins lie in the definitions of semantics of programming languages, notably at IBM's Vienna Laboratory, in the 1970s [1, 5]. The basic modelling language (VDM-SL) and its denotational semantics have been standardised [6, 7]. A proof theory has been defined, based on the typed logic of partial functions [8, 9]. Extensions to the language have introduced object-oriented structuring and concurrency [10]. A further extension of VDM++ called VDM Real Time (VDM-RT) is the dialect used in this paper and in DESTECS in general.

A goal of VDM's development since the mid-1990s has been to develop modelling and analysis techniques that are accessible to the majority of systems and software engineers, and do not require deep understanding of the form of the underlying semantics. Emphasis has been laid on the development of robust and efficient tools for simulation of executable models, rather than proof, and on links to other less formal modelling frameworks, such as UML [11]. Current tool support for VDM includes the commercial VDMTools, and the more recent open-source tool Overture [12, 13]. The emphasis on simulation has led to the development of a very efficient interpreter for the executable subset of VDM [14]. As a consequence of this approach, there is an ongoing record of successful industry deployment [15, 16].

2.2 Continuous Time Modelling

In the area of continuous time modelling without doubt, Matlab/Simulink [17] has the largest user base in industry as well as in the academic world. The modelling and simulation part, Simulink, is built upon the Matlab environment and provides block diagram modelling. The base library of Simulink is limited to block diagrams. External libraries with physical components can be purchased. These libraries are comparable to what is offered in Modelica [18] and 20-sim [19], but not with the same level of sophistication. Moreover, the library models are closed source.

20-sim is a multi-domain modelling and simulation package for modelling complex physical systems. Allthough 20-sim is a commercial tool all model libraries are open source. The package supports mixed mode integration techniques to allow the modelling and simulation of computer controlled physical systems that contain continuous as well as discrete time parts. The package supports the connection of external software through dll-functions, both at modelling and simulation level (discrete-time, continuous-time or hybrid). 20-sim allows export to Matlab/Simulink but still not mature in all levels. 20-sim also has a 3D editor enabling physical visualization of all simulations which we will show illustrations from later.

2.3 Injecting Potential Faults

In the DESTECS project there is also a focus on ways in which we can model abnormal behaviour, whether caused by conventional faults or "malicious" users, and defences against these, including fault tolerance mechanisms that may protect against these. However, for the case study described in this paper we have not yet exploited these possibilities.

3

The Physics and Control of the Conveyor System

This chapter contains a description of the physics and mathematical models that are the fundament for the co-simulated model of the control of the tilting functionality in curves. This first model contains the modeling of the environment for the path of the conveyor-loop which is composed by straight lines and circle curve segments as illustrated in the figure below.

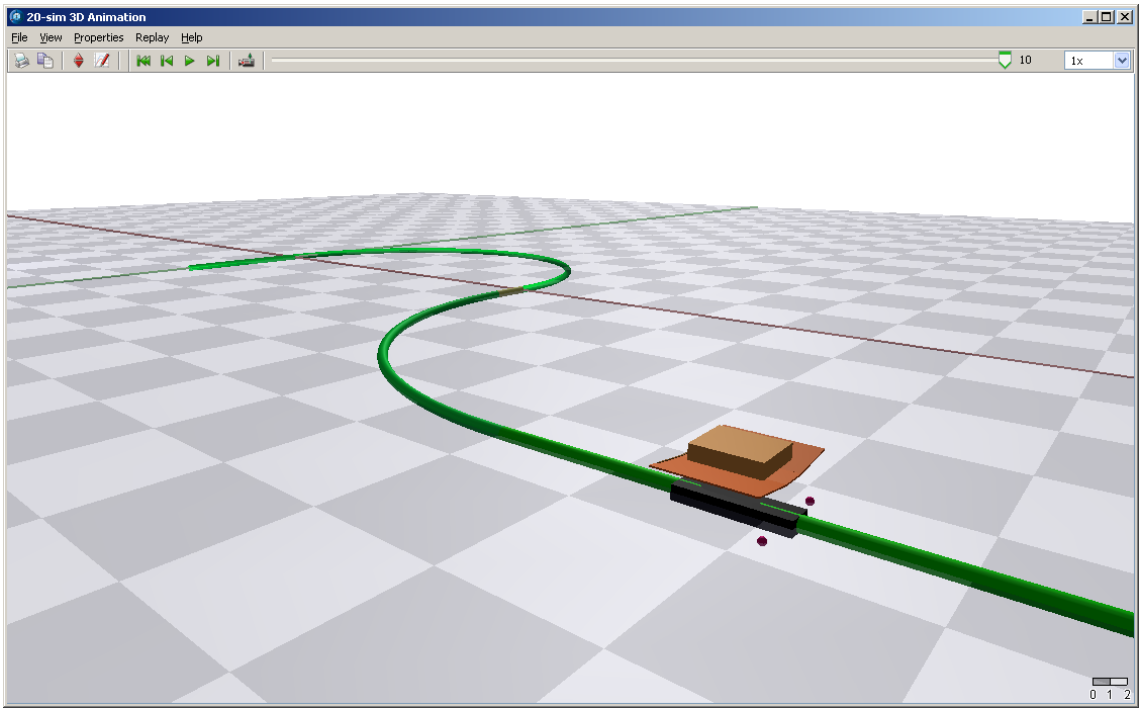


Figure 3.1: Path of the conveyor segment

On this conveyor path the parcel is transported on a trolley at a constant speed. Only one trolley is modeled with focus on transporting a single parcel trolley. The single trolley and parcel are modeled as a 3D Mechanical model of rigid bodies elements [20] composed by modelling body parts of the electrical tilting device, trolley and parcel.

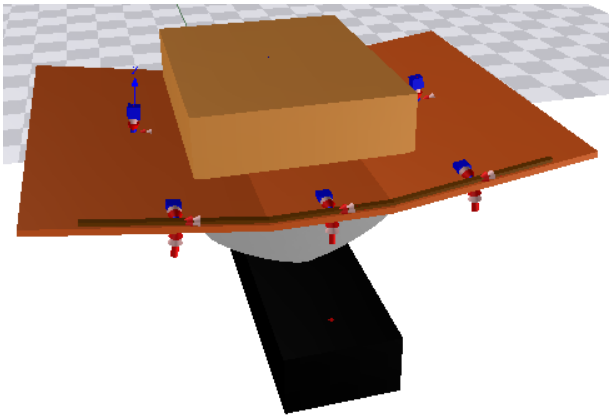


Figure 3.2: 3D Mechanical model of trolley and parcel

The 3D mechanical model is not described in details in this report since the major work is done by Frank Groen, Controllab Products B.V. in the Netherlands. The physical theory for the parcel trolley and edge contact and 3D position controller are neither covered by this report. The main focus on the work in this report is on controlling the tilt angle of the trolley moving in curves on the path of the conveyor segments. The physical and mathematical models behind the tilting control function are described in the following sections. The physics that are included covers the conveyor velocity, acceleration of the tilting trolley and forces like gravity, friction and the centrifugal force that has an influence on the safe position of parcels. The tilt angle of the trolley, tilting on and off motion curves, conveyor path and angular PID controller are described in detail in the following chapters.

3.1 Tilt Angle

In curves of the loop conveyor the tilting control function is activated to ensure that parcels are kept in the same position on the trolley. The tilting control function ensures that the angle of a fully tilted trolley must be sufficient to neutralize the centrifugal force applied to the parcel. This is done by tilting a trolley the angle as shown in figure 3.3. The resulting force vector (Fres), that is composed by the centrifugal and gravity force, must be perpendicular to the surface of the tilted trolley. The desired angle (α_d) can be calculated by the equation 3.3 based on the trigonometric of the two forces.

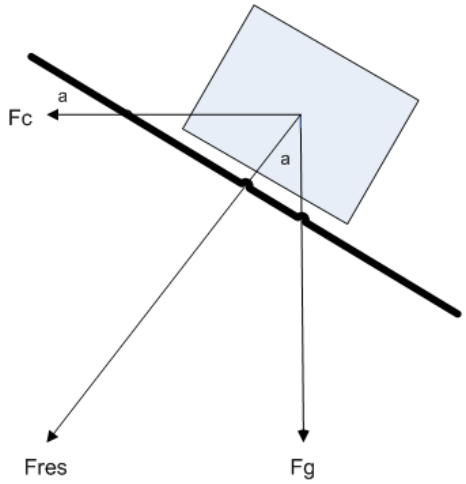


Figure 3.3: Tilt Angle

The centrifugal force is give by

$$Fc = \frac{mv^2}{r} \tag{3.1}$$

and the gravity force by

$$Fg = mg \tag{3.2}$$

the desired tilted angle (α_d) can be computed based on the two force vectors as

$$\alpha_d = arctan\frac{Fc}{Fg}$$

inserting equation for the forces we have

$$\alpha_d = arctan(\frac{v^2}{rg}) \tag{3.3}$$

3.2 Tilt Motion Curves

At certain locations at the entrance of a curve area an actuator is located at the conveyor to signal when to perform the tilt control function. The tilt on or off position must be reached in the limit of the tilt runtime (tp) starting from the offset position of the actuators. The actuators are controlling the motion of the trolley with a constant angular acceleration and deceleration. The time it takes to accelerate the angular velocity to a certain velocity is determined as a percentage (p) of the tilt runtime (tp). The tilt runtime includes the sum of the acceleration and deceleration time. The motion curve moving the trolley in tilt on and off position is illustrated in the figure below.

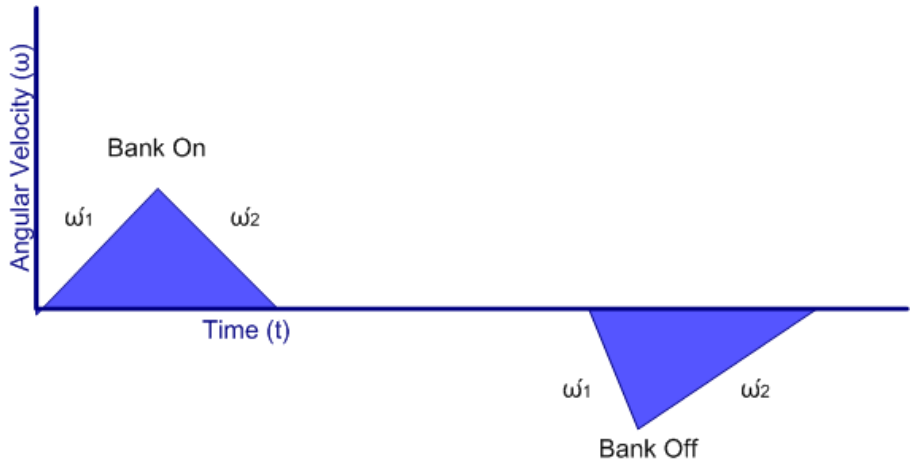


Figure 3.4: Angular velocity for tilt on and off

The desired tilt on angle (α_d) can be found using the equation 3.4 below where the angular acceleration ($\dot{\omega}_1$), deceleration($\dot{\omega}_2$), percentage (p) and tilt runtime (tp) is used as parameters.

$$\alpha_d = 1/2 \cdot \dot{\omega}_1 (p \cdot tp)^2 + 1/2 \cdot \dot{\omega}_2 ((1-p) \cdot tp)^2 \tag{3.4}$$

The maximum angle velocity must be equal for acceleration and deceleration according to figure 3.4 give by

$$\dot{\omega}_{max} = \dot{\omega}_1 \cdot p \cdot tp = \dot{\omega}_2 ((1-p) \cdot tp) \tag{3.5}$$

substituting and reduction equation 3.4 and 3.5, we obtain

$$\dot{\omega}_1 = \frac{2\alpha_d}{p \cdot tp^2} \qquad \dot{\omega}_2 = \frac{2\alpha_d}{(1-p) \cdot tp^2} \tag{3.6}$$

during angular acceleration the angle as function of time (t) can be written as

$$\alpha(t) = \frac{\dot{\omega}_1}{2} t^2 \tag{3.7}$$

during angular deceleration the angle can be found by finding the top point (Tp) in the second order polynomial written as

$$\alpha(t) = at^2 + bt + c, \qquad Tp = (-\frac{b}{2a}, -\frac{D}{4a}), \qquad D = b^2 - 4ac$$

the polynomial constant a, can be found based on the angular deceleration given by

$$a = -1/2 \cdot \dot{\omega}_2$$

the constants b and c can be found from the second order polynomial given that the top point (Tp) should be equal to

$$Tp = (tp, \alpha_d)$$

finally we find that during deceleration the angle $\alpha(t)$ can be calculated by the below equation

$$\alpha(t) = -\frac{\dot{\omega}_2}{2}t^2 + \dot{\omega}_2 \cdot tp \cdot t + \alpha_d - \frac{\dot{\omega}_2}{2}tp^2 \tag{3.8}$$

The acceleration 3.7 equation must be used when $t <= p \cdot tp$ and the deceleration 3.8 equation when $t > p \cdot tp$ for the tilt on motion curve. The resulting angular motion curves are verified in Matlab as illustrated in the figures below. The first curve illustrates the situation where the percentage of acceleration and deceleration is equal. (p = 0.5) The second curve illustrates the situation where 80% of the tilt runtime (tp) is used to accelerate the trolley in tilt position and the remaining 20% for decelerating. (p = 0.8)

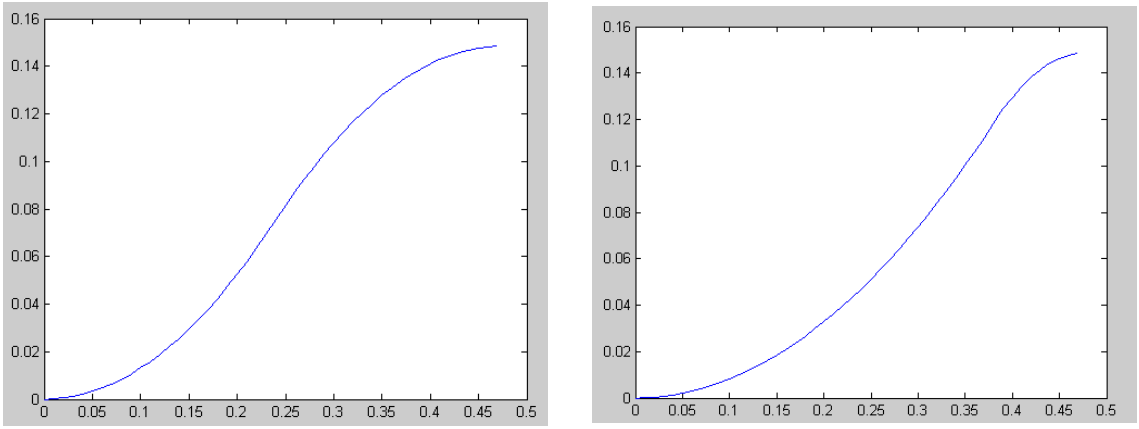


Figure 3.5: Motion curves for tilt angle $\alpha(t)$ as function of time for p = 0.5 and p = 0.8

In the same way the tilt off motion curves can be found given that the top point (Tp) should be equal to zero reaching the tilt runtime (tp).

$$Tp = (tp, 0)$$

The acceleration 3.9 equation must be used when $t <= p \cdot tp$ and the deceleration 3.10 equation when $t > p \cdot tp$ for the tilt off motion curve.

$$\alpha(t) = -\frac{\dot{\omega}_1}{2}t^2 + \alpha_d \tag{3.9}$$

$$\alpha(t) = \frac{\dot{\omega}_2}{2}t^2 - \dot{\omega}_2 \cdot tp \cdot t + \frac{\dot{\omega}_2}{2}tp^2 \tag{3.10}$$

3.3 PID Controller

Adjusting the trolley to the desired tilt angle is done by a PID controller. A proportional-integral-derivative (PID) controller calculates the "error" value as the difference between the measured angle of the trolley position and the desired tilt angle setpoint. The controller attempts to minimize the error by adjusting the setpoint based on the error. The PID algorithm involves three separate parameters: the proportional (P), the integral (I) and derivative (D) values. The proportional value determines the relation to the current error, the integral value determines the reaction based on the sum of recent errors, and the derivative value determines the reaction based on the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process of adjusting the desired tilt angle. The weight constants are denoted K_p, K_i and K_d. By tuning the three constants in the PID controller algorithm, the controller can provide control action designed for specific processing requirements. The response of the controller can be described in terms of the responsiveness of the controller to an error, the degree to which the controller overshoots the tilt angle setpoint and the degree of system oscillation. The generic PID controller is illustrated in the figure[1] below.

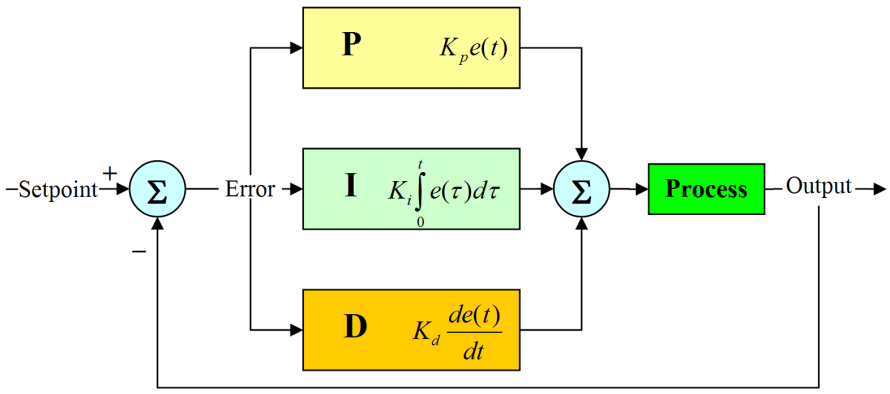


Figure 3.6: A PID Controller

In the trolley model the physical angular torque that moves the trolley in position is controlled by a PID controller. The PID controller is adjusted to minimize overshoots and oscillation, at the same time regulating to the desired angle as fast as possible. The model implementation of the PID controller uses the built in integral and derivative functions with respect to the time found in the 20-sim reference manual [21].

[1] http://en.wikipedia.org/wiki/PID_controllers

3.4 Conveyor Path

The conveyor path is composed of linear and circular segments. The trajectory of the trolley is modeled as moving in a 2D space computing the position and velocity vectors as the trolley is moving along the conveyor path. The conveyor path in this model is divided into 5 segments composed by linear and circular segments as illustrated in the figure below.

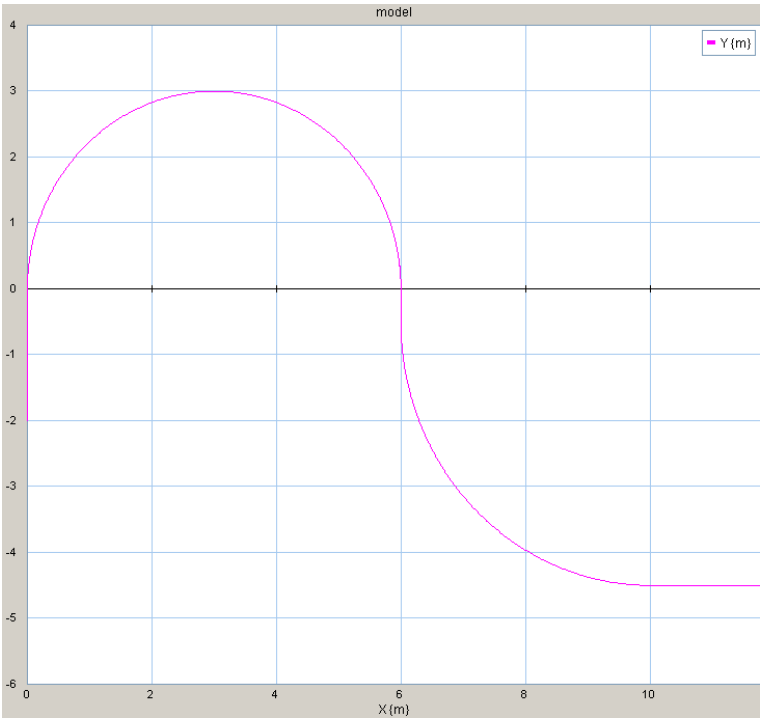


Figure 3.7: Path of the conveyor segment

The equations below are used for computing the position and velocity in a 2D space (x,y) on a straight line and circle segment. Position for a trolley moving on a linear segment in y or x direction given by the conveyor speed (v) in start position (x_s, y_s), we find

$$p_x(t) = v_x t + x_s \qquad p_y(t) = v_y t + y_s$$

The conveyor path model in figure 3.7 shows we are always either moving in the x or y direction and therefore either v_x or v_y will be zero. The angular velocity for the trolley moving with velocity (v) in a circular curve with radius (r), is found by

$$\omega = \frac{v}{r}$$

The position (p_x, p_y) for a trolley moving on a circular segment with the center (c_x, c_y) and radius (r), is given by

$$p_x(t) = r\omega cos(\omega t + \omega_{start}) + c_x \qquad p_y(t) = r\omega sin(\omega t + \omega_{start}) + c_y \tag{3.11}$$

The velocity (v_x, v_y) can be found as the derivative of the position vector

$$v_x(t) = p'_x(t) = -r\omega sin(\omega t + \omega_{start}) \qquad v_y(t) = p'_y(t) = r\omega cos(\omega t + \omega_{start}) \tag{3.12}$$

The level change can be modeled as described for the circular helix on page 377 in [22]. This is done by adding a third dimension to the 2D euclidean space model described above.

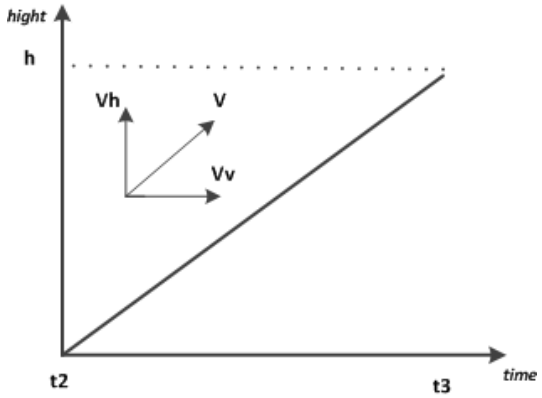


Figure 3.8: Slope for level change in circular helix curves

The horizontal and vertical velocity can be found as

$$v_h = \frac{h}{t3 - t2} \qquad v_v = \sqrt{v^2 - v_h^2} \tag{3.13}$$

3.5 Alternative Conveyor Loop Curves

This section describes an alternative of using circles in the conveyor loop curves and how to evaluate the use of piecewise clothoid curves as described in [23]. The paper describes how to create curves for Computer Graphics composed by clothoids, line and circular-arc segments as shown in the figure below.

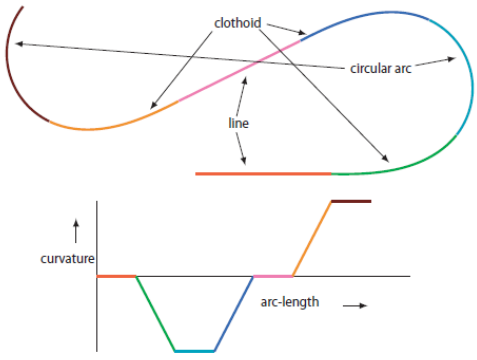


Figure 3.9: Curve composed of clothoids, line and circular-arc segments

A clothoid, also known as a Cornu or Euler Spiral, is defined as a curved line whose curvature varies linearly with its arch-length [24]. This has also been modelled and analysed as a part of our work but the details of this will be left out of this paper since this idea was not the primary scope for the company.

4

The Physical Model of the Conveyor System

This chapter contains the description of the continuous time (CT) and discrete event (DE) models in terms of SysML blocks [25]. The Systems Modeling Language (SysML) is a general-purpose diagram model language that supports the specification, design, analysis and verification of systems. These systems may include software, hardware and mechanical components. SysML is a graphical modeling language with a standardized semantic for representing requirements, behavior, structure and properties of the system and its components. In this work modeling structure with blocks is done by using the block definition diagram and internal block diagrams as described in [3]. The Block Definition Diagrams (BDDs) describes the Tilt Control Function Model composed of SysML blocks which gives an overview of the model decomposition of sub-modeling elements fulfilling different purpose and functionality. The model is divided into two sub-models concerning the conveyor environment in continuous time (CT) and the controller in discrete event (DE). The modeling blocks are described briefly in this chapter for sub-models of the conveyor and environment.

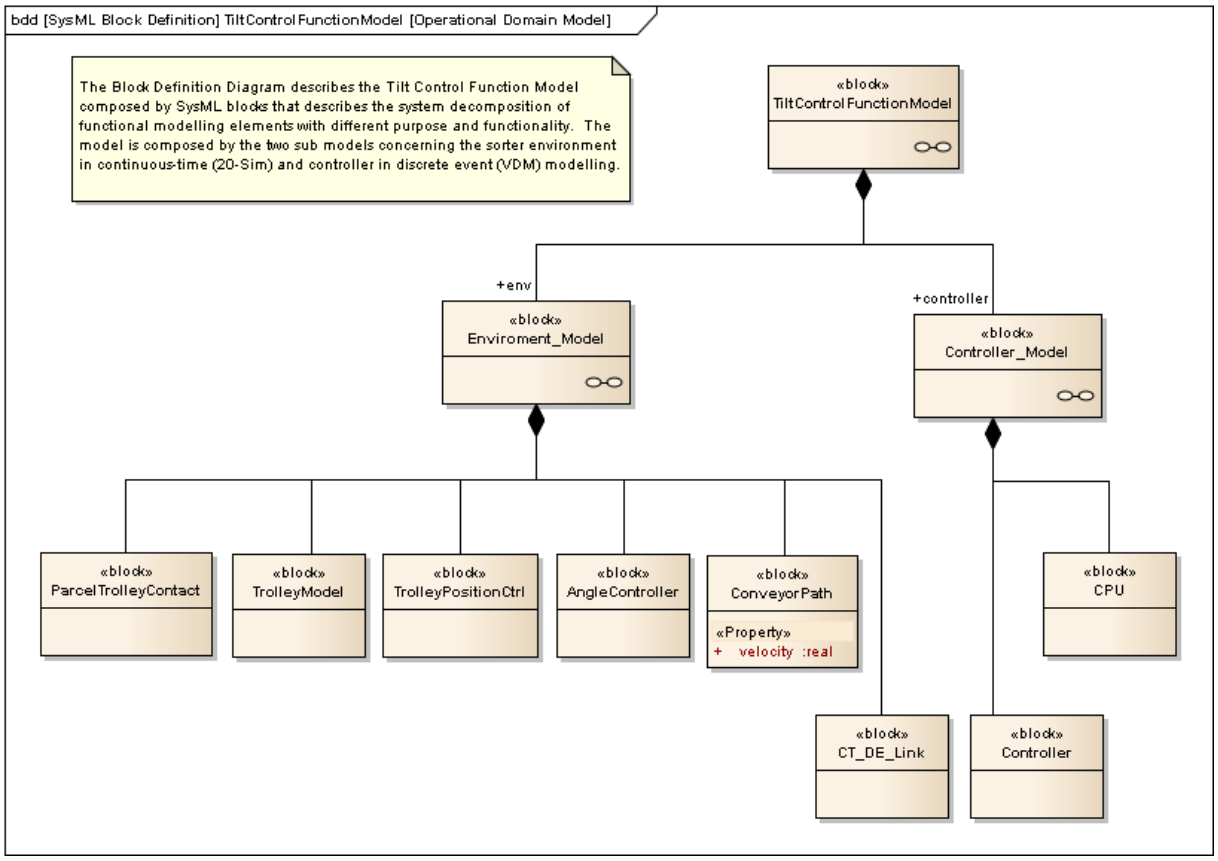


Figure 4.1: SysML Block Definition Diagram of Tilt Control Function Model

4.1 Model of Conveyor Environment (CT)

The continuous time model is composed of a number of modeling blocks that together models the environment including the conveyor loop, trolley and parcel.

The relation between the different modeling blocks (submodels) are illustrated in the internal block diagram below.

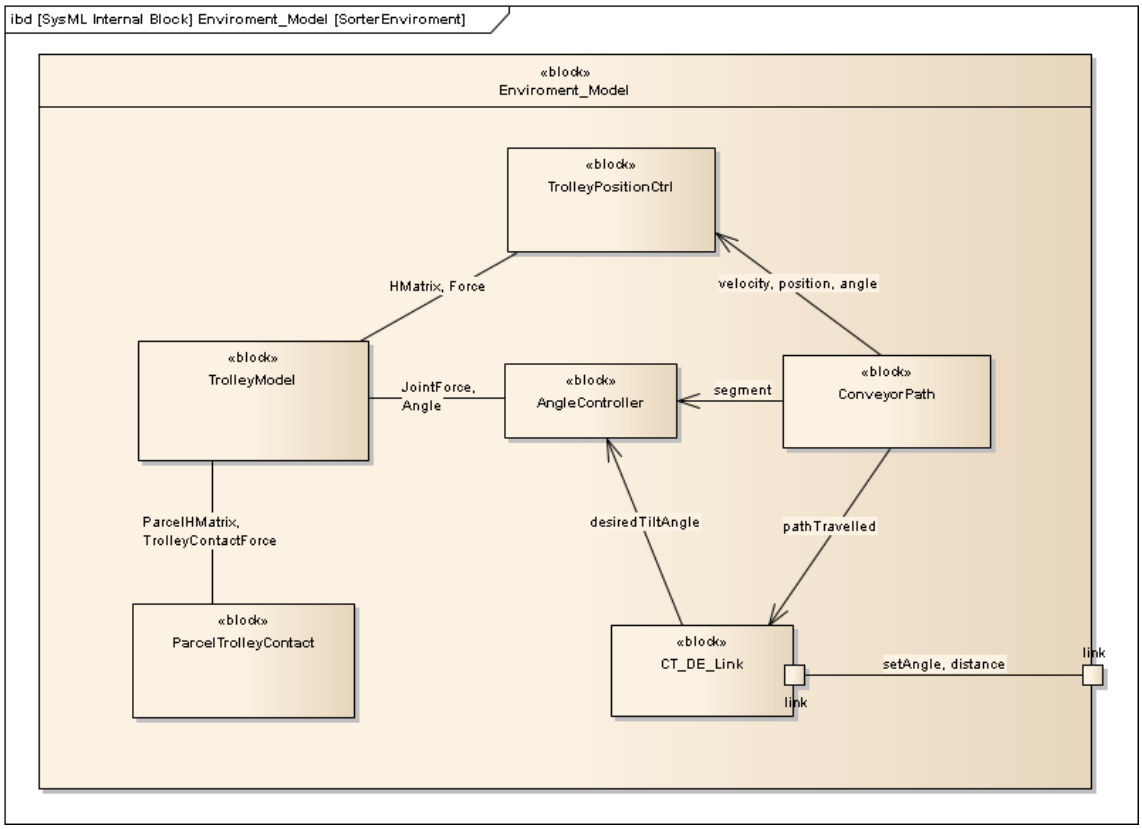


Figure 4.2: SysML Internal Block Diagram of continuous time part of model

A CT model consists of submodels connected by terms of signals and powerports. A signal port is either an input or output port and the signal has a direction from one submodel to another. In the SysML internal block diagram these signals are described with arrows on edges between the submodel blocks. The `desiredTiltAngle` in figure 4.2 is such a signal that simulates the current desired tilt angle.

A powerport is always characterized by two domain independent variables as described in [26] chapter 8. The product of these variables has the dimension of power in Watts. Therefore, they are called *power conjugated variables*. For a mechanical domain the variables are force and velocity, for an electrical domain the variables are voltage and current. In 20-sim iconic diagrams, like we use in this CT model, a powerport is a port where power can be exchanged between a component (submodel blocks) and its environment in terms of these variables. One of the variables will be an input and the other, an output. A powerport in the electrical domain is composed by an across variable voltage and a through variable current as described in the 20-sim manual section 2.4 and 4 [21]. Powerports are illustrated by edges between the SysML blocks without any arrow on the edge. The `JointForce` in figure 4.2 is an example of a powerport composed by a torque (T) and angular velocity (omega). The product of the torque and angular velocity has also the dimension

of power in Watts. In our case the torque has a direction from the `AngularController` to the TrolleyModel. The angular velocity has the opposite direction and gives the current physical angular speed of the tilting trolley implicit given by the `JointForce` powerport.

Conveyor Path is a model of the conveyor loop lay-out. The model consists of a number of segments composed by linear and circular pieces. The conveyor path models the trajectory of the trolleys motion on the conveyor loop. The position, angle and velocity is computed in 3D euclidean space as a function of the simulation time.

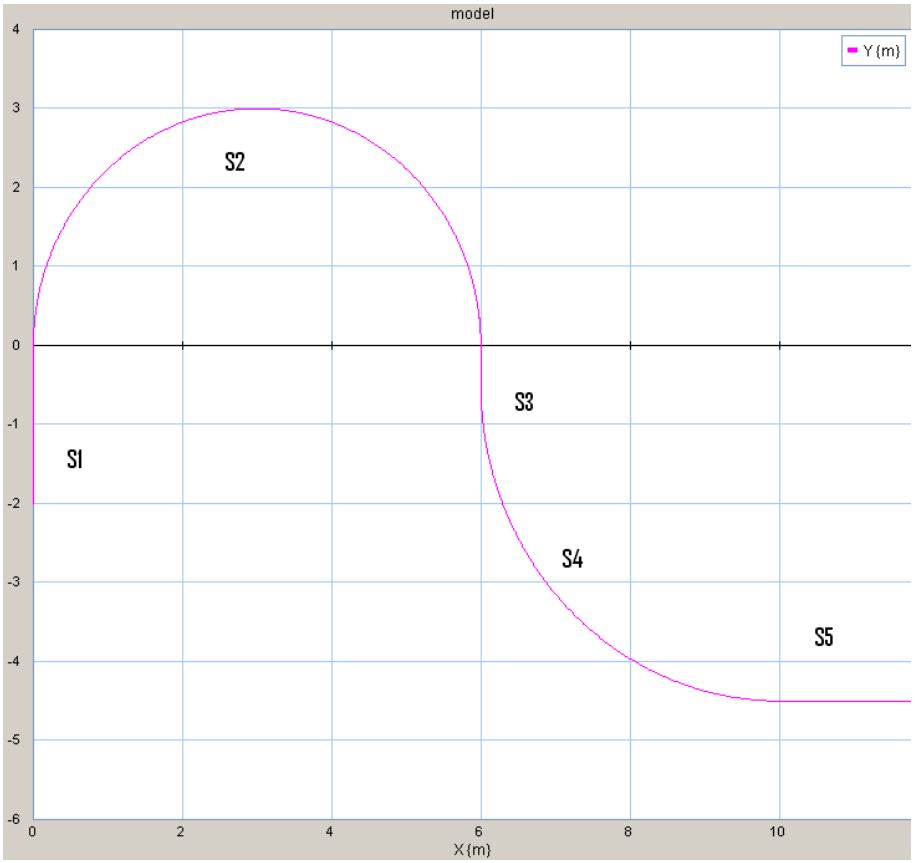


Figure 4.3: Conveyor Path and Segments

The conveyor path in this case is combined by the 5 segments listed below. (See figure 4.3)

- S1 is a straight segment in the y-direction
- S2 is a curve segment of 180 degrees clockwise, possible to have a level change in the z-direction
- S3 is a straight segment between 180 and 90 degrees curves
- S4 is a curve segment of 90 degrees counter clockwise
- S5 is a straight segment in x-direction

Trolley Model is a 3D model of the parcel and trolley made up by modeling systems of rigid bodies fully described in [20]. A rigid body is used to model a component as a system of particles where the distance between particles remain unchanged. Each particle in a rigid body is located by its constant position vector in a cartesian reference frame. The method

assumes an inertial coordinate system to describe the translational velocities and forces of the bodies and defines a body-fixed coordinate system, with their origin in the gravitational center of each rigid body. In the 20-sim 3D mechanics editor it is though possible to have the center of gravity (COG) at a different location than the body reference. This could be very useful for instance with cylinders, where you want to have the reference at the bottom of the cylinder. The velocity of each part of a rigid body is defined by the combination of translational and a rotational velocity. Bodies can be connected with either welding, rotation or translation joints.

For the trolley model welding joints are used to model the connected front edge composed by 3 body parts (center, left and right) to the trolley edge. The trolley is similarly assembled by 3 parts named: trolley 1 (center part), trolley 2 (right side) and trolley 3 (left side). A rotation joint is used to model the angle position of the trolley in relation to the slider on which the trolley i moving. For the 3D model of the parcel and trolley the bodies are defined as shown in figure 4.4 from the 20-sim 3D Mechanical editor below.

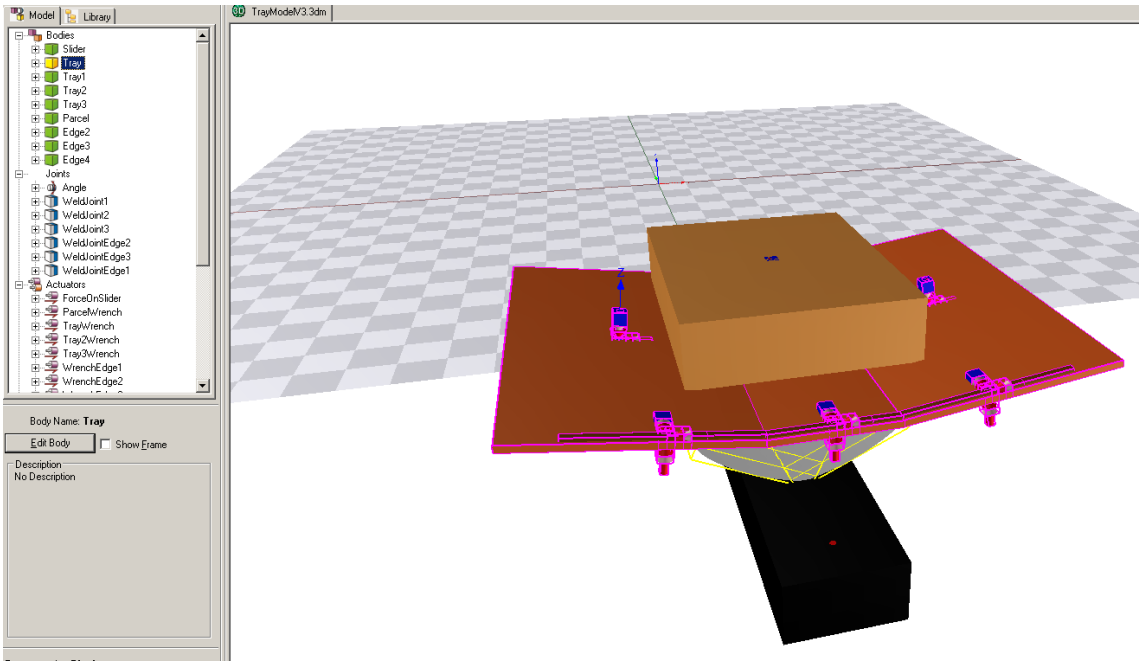


Figure 4.4: 3D Model of trolley composed by rigid bodies

The list of bodies and joints in the 3D mechanical model of the parcel, slider, trolley and front edge can be composed as:

Bodies

- Slider - Conveyor part on where trolley is moving
- Trolley - Mechanical tilting device that supports trolley and parcel
- Trolley 1 - Center part of trolley
- Trolley 2 - Right side of trolley
- Trolley 3 - Left side of trolley
- Parcel
- Edge 2 - Edge right side of trolley

- Edge 3 - Edge left side of trolley
- Edge 4 - Edge center part of trolley

Joints

- Angle - Angle of trolley with reference to slider
- WeldJoint1 - Welding joint for Edge 4
- WeldJoint2 - Welding joint for right side of trolley
- WeldJoint3 - Welding joint for left side of trolley
- WeldJointEdge1 - Welding joint for Edge 4
- WeldJointEdge2 - Welding joint for Edge 2
- WeldJointEdge3 - Welding joint for Edge 3

Sensors and actuators are model components used to interface with the 3D mechanical model. Sensors in this model are of the Jacobian matrix type. The Jacobian matrix is described in the basic concepts of planar kinematics in section 3.1 of [27]. The study of analysing the dynamic response of a system of interconnected bodies as a result of applied forces is done by formulating equations of motion. The Jacobian matrix is composed of first-order partial differential equations describing the orientation of a tangent plane to the function at a given point. The Jacobian matrix describes the orientation of the velocity and acceleration vectors in the 3D euclidean space. It plays a central role in the theory and numerical methods of kinematics and dynamics.

A H-Matrix is a 4x4 matrix that contains a combined position and orientation of the body. It is made up of a 3x3 matrix that describes orientation or rotation by three 3x1 vectors that are perpendicular to each other. An important property of the rotation matrix is that, it is an orhonormal matrix, so the transpose of the matrix is its inverse. The last part of the H-Matrix is a 3x1 vector for the position of the body (x,y,z) in the 3D euclidean space.The lower row is always [0,0,0,1] to make a squared 4x4 matrix.

The advantage is that by multiplying those matrices the relative position and orientation can be obtained for the objects joint together in 3D space. Actuators in the model uses power ports that includes a force and velocity vector in 3D space. The advantage of these matrices is that properties of bodies, like positions, forces and velocities can be transformed from one coordinate frame to another. For more details on actuators and sensors open the 3D mechanical model of the TrolleyModel in the 20-sim model.

Trolley Position Controller controls the trolley position and heading in 3D euclidean space as a function of time. The position controller converts the position, velocity and angle signals from the conveyor path modeling component to forces and velocity vectors that controls the actuators in the 3D mechanical model. Sensor inputs from the 3D mechanical model, in terms of three H-Matrices one for each sub part of the trolley (center, right and left), is used to adjust the current position and orientation of the trolley to the desired values. The desired H-Matrix is computed based on the position and velocity vectors from the conveyor path. The force needed to move the trolley into the desired position and orientation is computed using Twists, Wrenches and Adjoint matrices to transform the orientation of the trolley as described in [28].

Parcel Trolley Contact, contains the equations for the friction for each corner of the parcels contact with the trolley. The contact is calculated for each of the four corners of the parcel.

The contact model calculates contact between the flat surface (the trolley) and the spherical corner of the parcel. The trolley consists of 6 flat surfaces. Rotational and translational friction is part of the model. The static friction defined as the torque of force necessary to initiate motion from rest is modeled as the contact stiffness and damping. The coulomb friction (kinetic friction, dynamic friction) as the component that is only dependent of the direction of velocity is modeled for rotational and translational movements. The friction coefficient (E) can also be adjusted for the model.

Controller Link models the 4 positions of the actuators used to order the trolleys to tilt on or off. The details of these aspects are left out because of confidentiality reasons.

Angle Controller regulates the physical position of a tilted trolley. It regulates the angle of the tilted trolley to the desired angle send by the DE model. The regulation is performed by combining three actions proportional, derivative and integral (PID) control as described on page 185-186 in [26] and in section 3.3 of this paper. The PID-controller is the most widely used controller, in this case it can be adjusted so the error between the tilted trolley angle and desired angle is decreased as quickly as possible without overshoot and no instability. The PID controller can be tuned by adjusting the PID constant parameters K_p, K_d and K_i in the model.

This sub-model is an abstraction of the physical movement of the tilting device driven by the gear motor. It abstracts the electronic PID controller, H-bridge and gear motor that all are implemented as mechanical and electrical parts of the tilting device. Only the controlling part of motion curves will be moved to the discrete event (DE) model. The PID controller will be kept in the continuous time (CT) model since these parts today already are working in the tilting device.

The figure below illustrates the DE model in 20-sim including sub-models described in this chapter. "Globals" containing the common parameters for all sub-models concerning the parcel contact with trolley and edge divided in 3 parts for center, right and left part of trolley and edge.

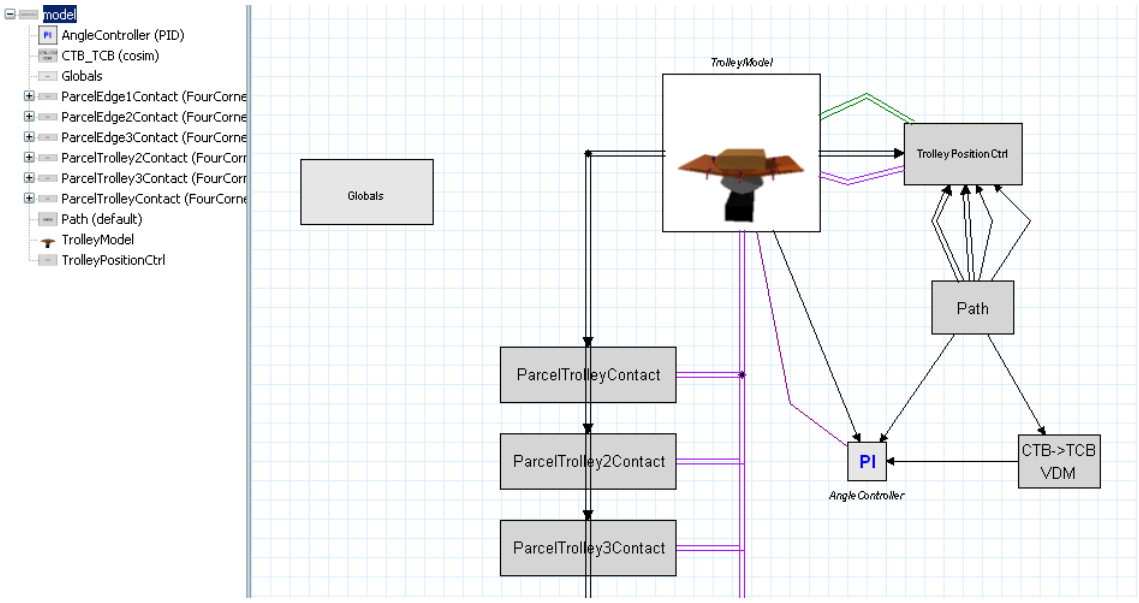


Figure 4.5: CT model overview in 20-sim

5

The Controller Model of the Conveyor System

This chapter describes the specification of the discrete event (DE) model in VDM-RT that models the controller software. The DE model is written in VDM++ which is an object-oriented formal design language. More details on the VDM++ language can be found in [10]. In the DESTECS project the overture version of VDM-RT is extended to handle co-simulation with 20-sim, more about the overture tool can be found in [13]. The DE model is linked to the continuous time model (CT) om terms of the co-model interface as described in the DESTECS methodological guidelines section 4.2.1 [29]. The contract between the CT and DE model is described in terms of SysML internal block diagram using SysML blocks and stereotypes defined for shared variables and events for the contract between the co-models. The VDM model is a model of the controller simulating setting the desired tilt angle in periodic timed steps. The DE model consist of an active thread allocated on a simulated CPU running at a certain speed. This is illustrated by the internal SysML block diagram in figure 5.1. Note that multiple cpus could be used if one wish to experiment for example with distributed controllers connected by buses.

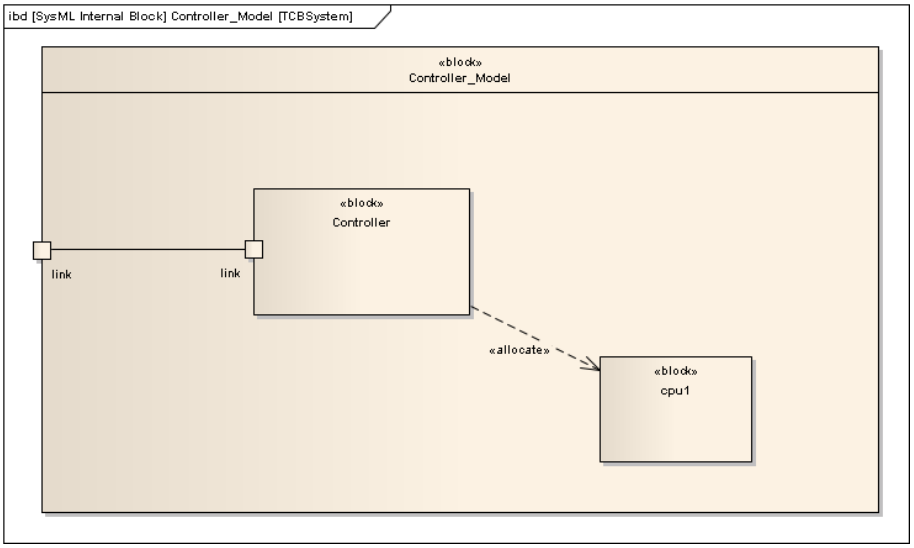


Figure 5.1: SysML Internal Block Diagram of DE part of DE Controller model

5.1 Model of Software Controller (DE)

In this section the VDM-RT specification is described in detail for the DE model of the controller.

```
system ControllerModel
instance variables

-- Architecture definition
public static Ctlr : [Controller] := nil;

-- CPU clock 100 MHz => 1 cycle = 10 ns
cpu1 : CPU := new CPU(<FP>, 100E9);

operations

public ControllerModel : () ==> ControllerModel
ControllerModel () ==
 (Ctlr := new Controller();
  cpu1.deploy(Ctlr, "Ctlr");
 );

end ControllerModel
```

Figure 5.2: VDM Controller Model that creates the controller deployed on a CPU

The system controller model instantiates the controller and deploy it on a simulated CPU with a specified clock speed. The controller contains the public static values and instant variables linked to the contract (See figure 6.1) of the shared parameters between the CT and DE models.

```
class Controller

instance variables
-- Distance travelled since simulation started
public distanceTravelled : real := 0.0;
public dist1 : real := 0.0;      -- Position of actuator 1
public dist2 : real := 0.0;      -- Position of actuator 2
public dist3 : real := 0.0;      -- Position of actuator 3
public dist4 : real := 0.0;      -- Position of actuator 4
public setAngle : real := 0.0;   -- Desired tilt angle
public nextAngle : real := 0.0; -- Desired tilt angle
private angle : real := 0.0;     -- Temp. computed tilt angle

-- Part of model left out

thread

 periodic (40E6, 0, 0, 0) (ControlStep);
```

Figure 5.3: Model of DE controller – instance variables

The "ControlStep" is the operation called from a periodic thread that computes the new desired tilt angle based on the distance travelled on the conveyor path. In this case the operation of the thread is called with a periodicity of 40 ms. That means sensors in the environment is sampled with a frequency of 25 Hz in this case the distance travelled on the conveyor path. With a conveyor speed of eg. 2.0 $\frac{m}{s}$ the tolerance of computing the tilt on and off position will be 40 ms * 2.0 $\frac{m}{s}$ = 8 cm. The DE model can be used to model the resolution of the controller and impact on the system behavior. In this case how accurate is necessary for the controller to sample the distance travelled ensuring safe tilt on and off functionality?

```
public ControlStep : () ==> ()
ControllStep() ==
(-- Updating previously computed desired tilt angle
 setAngle := nextAngle;

 -- First curve tilt on
 if (distanceTravelled >= dist1) and (distanceTravelled < dist2)
 then let t = (distanceTravelled - dist1)/v
      in
        tiltRightCurve(t, r2)

 -- First curve tilt off
 elseif (distanceTravelled >= dist2) and
        (distanceTravelled < dist3)
 then let t = (distanceTravelled - dist2)/v
      in
        tiltOffRightCurve(t, r2)

  -- Second curve tilt on
 elseif (distanceTravelled >= dist3) and
        (distanceTravelled < dist4)
 then let t = (distanceTravelled - dist3)/v
      in
        tiltLeftCurve(t, r4)

 -- Second curve tilt off
 elseif (distanceTravelled >= dist4)
 then let t = (distanceTravelled - dist4)/v
      in
        tiltOffLeftCurve(t, r4)
          --setAngle := tiltOff() -- Without motion curves
 else nextAngle := tiltOff();
);
```

Figure 5.4: Model of DE controller – periodic thread

Every time the distance travelled is equal to the position of the conveyor path to perform a tilt function the tilt motion curves of the conveyor path are computed by calling the tilt right and left curve operations. The time and curve radius is used as parameters to compute the desired tilt angle of the trolley. The setAngle is a shared variable that is used to exchange the desired tilt angle between the DE and CT models.

```
functions

-- Converts from radians to degrees
public radToDeg: real -> real
radToDeg(rad)==
    (rad/MATH`pi)*180;

-- Computate desired tilt angle
public tiltOnAngle: real -> real
tiltOnAngle (r) ==
    radToDeg( MATH`atan( (v * v / r ) / 9.82 ))
pre r > 0;

-- Right tilt angle
public tiltRight: real -> real
tiltRight(r) ==
    tiltOnAngle(r)
pre r > 0;

-- Left tilt angle
public tiltLeft: real -> real
tiltLeft(r) ==
    -tiltOnAngle(r)
pre r > 0;

-- Tilt off
public tiltOff: () -> real
tiltOff() == 0;
```

Figure 5.5: Model of DE controller – functions

Functions used to compute the desired tilt on angle for tilting left and right using the tilt angle equation 3.3.

```
-- Motion curve for right tilt-on
public tiltRightCurve: real * real ==> ()
tiltRightCurve(t, r) ==
  nextAngle := tiltOnCurve(t, r);

-- Motion curve for left tilt-on
public tiltLeftCurve: real * real ==> ()
tiltLeftCurve(t, r) ==
  nextAngle := -tiltOnCurve(t, r);

-- Tilt on angle using acceleration/decelleration motion curves
public tiltOnCurve: real * real ==> real
tiltOnCurve(t, r) ==
 let a = tiltOnAngle(r),
     tp = trolleyPitch/v, -- Trolley pitch in time
     wa1 = 2*a/(tp*tp*p), -- Acceleration
     wa2 = 2*a/(tp*tp*(1 - p)) -- Decelleration
 in
   (angle := if t < tp*p
             then (wa1/2)*(t*t) -- Acceleration curve, Tilt On
             elseif t < tp
             then -(wa2/2)*(t*t) + wa2*tp*t + a - (wa2/2)*(tp*tp)
                  -- Decelleration curve, Tilt On
             else a;

    return angle)
pre t >= 0 and r > 0 and r < r_max;
```

Figure 5.6: Model of DE controller - operations for tilt-on motion curves

Tilt on left and right operations to compute the tilt-on motion curve according to equations 3.7 and 3.8. The tilt-off operations are similar.

6

Co-Simulation of the Conveyor System

This chapter describes the co-simulated contract as described in the DESTECS methodological guidelines section 4.2 [29]. The co-simulation framework consist of the key elements centered around the co-model containing the DE and CT submodels. Each model has a model interface, which defines the part of the model which can be accessed externally. The shared model interface is defined primarily as shared design parameter and variables defined in the co-simulation contract. (See figure 6.1)

Design parameters are used to configure the model and are not changed during co-simulation. For the tilt control function model design parameters could be the conveyor speed or trolley length. Shared variables are changed during co-simulation and used to exchange information between the CT and DE submodels. Variables always have a direction in terms of information flow. Variables are either monitored or controlled by the controller in the DE model, hence their names are called monitored or controlled variables. Besides shared design parameters and variables the contract defines events that are possible to generate in the CT submodel. Events can be generated when a variable reaches a certain value like a certain position on the conveyor path.

The completed co-simulation contract for the tilt control function is defined and described in this chapter.

6.1 Co-simulation Contract

The contract between the DE and CT models is specified as shared design parameters (sdp), monitored and controlled variables (monitored, controlled). This contract specification is illustrated in the SysML internal block diagram below.

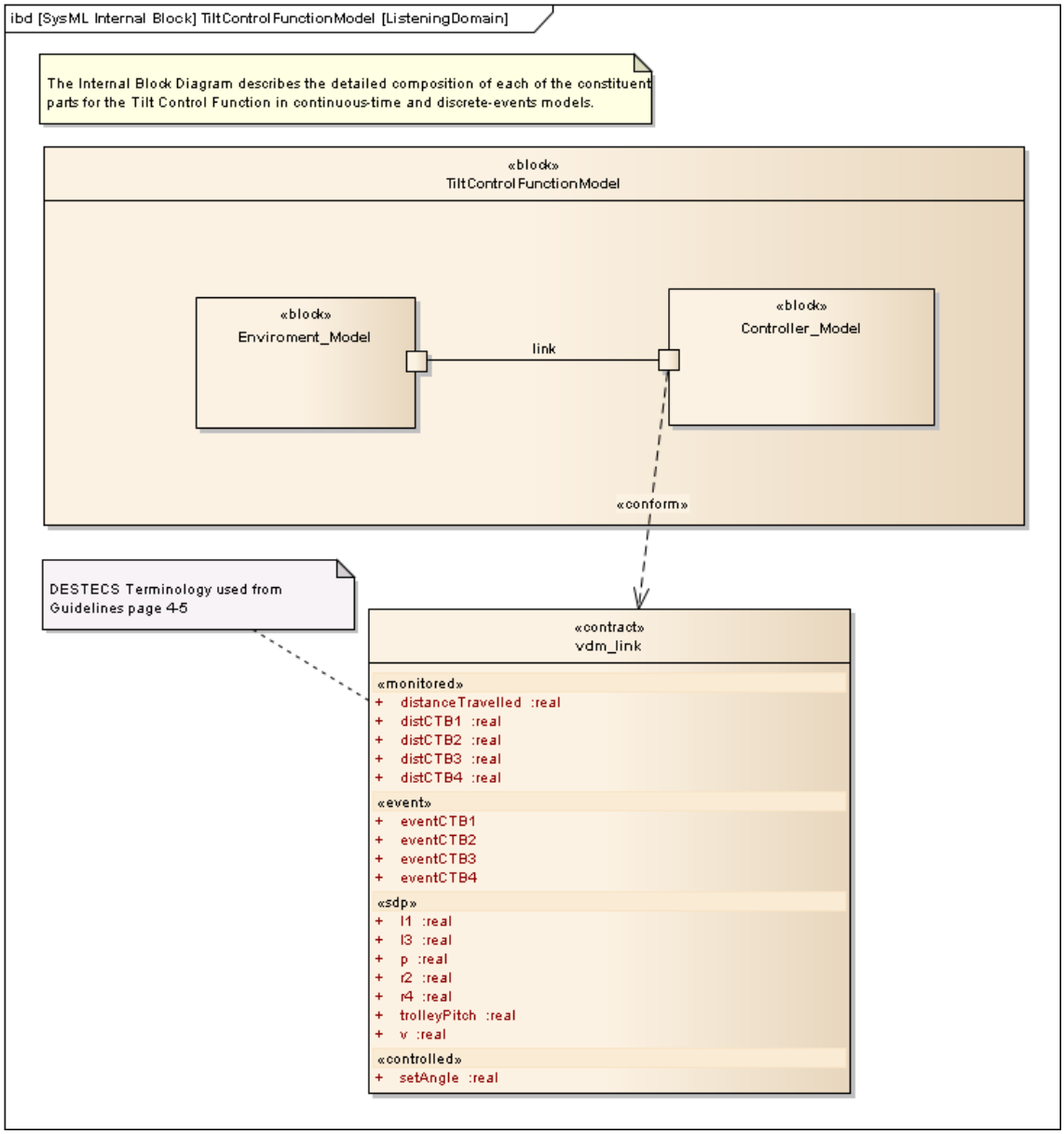


Figure 6.1: SysML Internal Block Diagram for interface part of models

The shared design parameters concerns the configuration of the conveyor path segments 1-4, trolley pitch and conveyor speed (v). The monitored variables are used by the DE model to compute the time for activating the tilt on and off motion curves. The controlled variable setAngle is the desired tilt angle computed by the VDM model of the controller. The set angle is computed in discrete event steps based on the periodic rate (default 20 ms) for scheduling of the controller.

The contract parameters linked to the VDM model is defined in separate files as shown below.

```
sdp v=Controller.v;
sdp r2=Controller.r2;
sdp r4=Controller.r4;
sdp l1=Controller.l1;
sdp l3=Controller.l3;
sdp trolleyPitch=Controller.trolleyPitch;
sdp p=Controller.p;

output setAngle=TCBCtlr.setAngle;

input distanceTravelled=TCBCtlr.distanceTravelled;
input distCTB1=TCBCtlr.distCTB1;
input distCTB2=TCBCtlr.distCTB2;
input distCTB3=TCBCtlr.distCTB3;
input distCTB4=TCBCtlr.distCTB4;

event eventCTB1=TCBCtlr.eventCTB1;
event eventCTB2=TCBCtlr.eventCTB2;
event eventCTB3=TCBCtlr.eventCTB3;
event eventCTB4=TCBCtlr.eventCTB4;
```

Figure 6.2: VDM Link and contract

The shared variables and parameters for co-simulation are listed below:

Shared design parameters (CT <-> DE).

- l1: real [m] // Length of conveyor path linear segment 1
- r2: real [m] // Radius of conveyor path circular segment 2
- l3: real [m] // Length of conveyor path linear segment 3
- r4: real [m] // Radius of conveyor path circular segment 4
- trolleyPitch: real [m] // Length of trolley
- v: real [m/s] // Conveyor speed

Monitored design variables (CT -> DE):

- distanceTravelled: real [m] // Distance traveled on conveyor path
- dist1: real [m] // Position on conveyor path for CTB1 - computed by CT model

- dist2: real [m] // Position on conveyor path for CTB2 - computed by CT model
- dist3: real [m] // Position on conveyor path for CTB3 - computed by CT model
- dist4: real [m] // Position on conveyor path for CTB4 - computed by CT model

Controlled design variables (DE -> CT):

- setAngle: real [deg] // Desired trolley set tilt angle

Below is listed the CT sub-model of the interface in 20-sim to the controller modeled in VDM. The shared design parameters are marked 'shared'. The monitored and controlled parameters seen from the DE model are exported or imported by the CT model. Events are generated (eventCTBx) with the "eventup" a zero crossing detection function. These events can be used in the DE model to trigger when the tilt on and off actuators are passed on the conveyor path.

```
/* Submodel/block
CTB_TCB Link: co-simulation interface to DE model
*/
variables
   boolean global eventCTB1 ('eventup');
   boolean global eventCTB2 ('eventup');
   real angle {deg};
parameters
   real global trolleyPitch ('shared') {m};
   real global v  ('shared') {m/s}; // Conveyor speed
   real global r2 ('shared') {m}; // Radius of the first curve
   real global p ('shared'); // Percentage of runtime acc.
   real global dir2; // Direction of first curve
   real global l1, l2 {m}; // Length of linear segments
externals
   real global export distanceTravelled {m};
   real global export distCTB1 {m}; // Position of CTB1
   real global export distCTB2 {m}; // Position of CTB2
   real global import setAngle {deg};
equations
   // Distance for position of CTB on conveyor path
   distCTB1 = l1 - offsetCTB1;
   distCTB2 = l1 + l2 - offsetCTB2;
   // Generate events when path traveled equal to CTB positions
   eventCTB1 = eventup(pathTravelled-distCTB1);
   eventCTB2 = eventup(pathTravelled-distCTB2);
   // Monitored and controlled shared variables
   distanceTravelled = pathTravelled;
   desiredTiltAngle = setAngle;
```

Figure 6.3: 20-sim CT sub-model of link to the DE model

6.2 Co-simulation Results

During simulation, curve graphs are displayed for the conveyor path segments (segment), trolley velocity (X,Y) and heading angle, physical (physicalTiltAngle) and controlled tilt angle (desiredTiltAngle). Finally the parcel corner position on trolley in X and Y direction are displayed. (See figure 6.4)

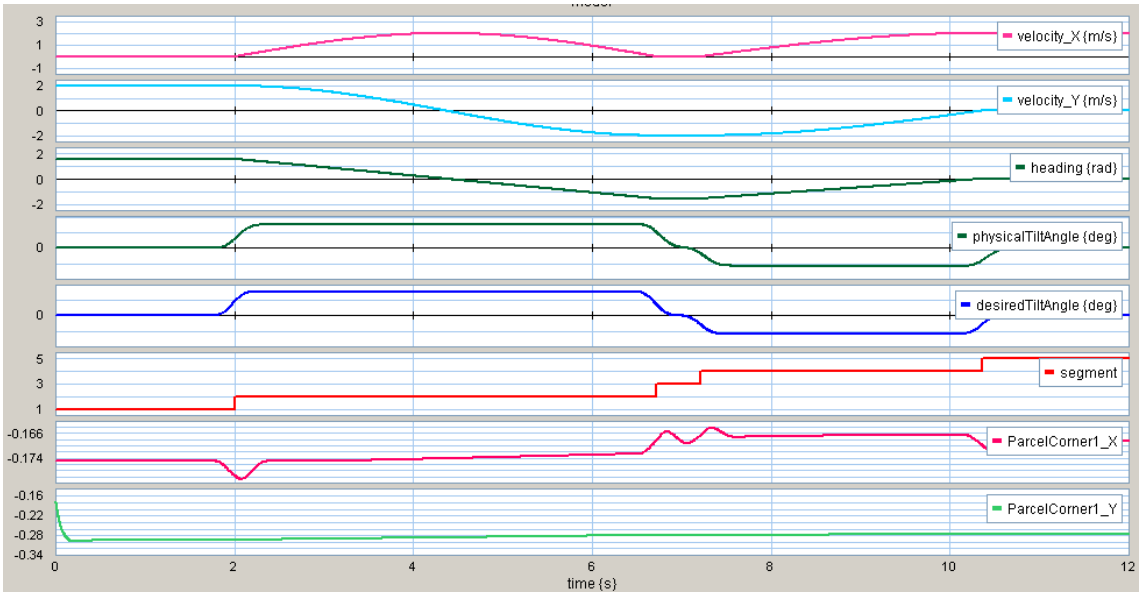


Figure 6.4: Co-Simulated result of tilt motion curves in 20Sim

The best way to investigate the impact of a model parameter change is to adjust the parcel dimensions, mass and friction so the parcel will be slightly sliding on the trolley during motion. In the following test a parcel with dimension (100*450*350 mm, 2 kg) is selected. We would like to investigate the impact of the new percentage (p) parameter. Does it have an influence for the safe position of the parcel on the trolley? If we try to simulate controlling the tilt angle using different motion curves by adjusting this percentage (p) how would it impact the parcel sliding position (X) on the trolley? This is done by comparing the curve graphs of the parcel corner positions in the X direction after each simulation. The result after these different simulations are listed in the curve graphs below for p = 0.2, 0.5 and 0.8.

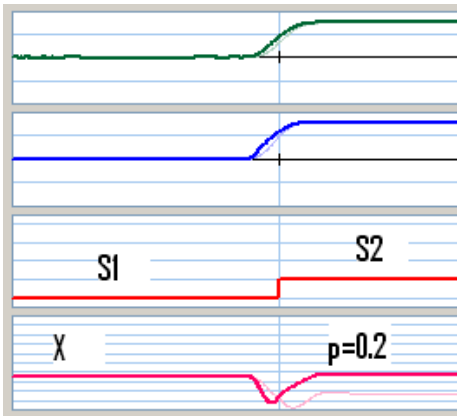


Figure 6.5: Tilt on motion curve, adjusting p from 0.5 to 0.2 (bold)

Comparing the two simulation results it turns out that adjusting the percentage parameter to 0.2 is better than 0.5 and 0.8. That means accelerating the tilt angle fast shortly gives a safer parcel positioning on the trolley. The forces on the tilting device mechanics will be increased and could have an impact on the mechanical construction. This observation would be interesting to verify on a real conveyor with the new tilt function being able to adjust the tilt motion curves by the percentage (p) parameter.

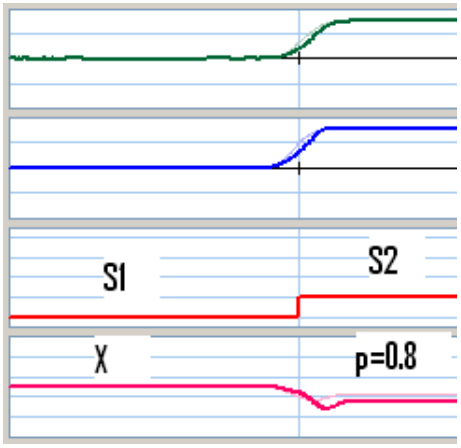


Figure 6.6: Tilt on motion curve, adjusting p from 0.5 to 0.8 (bold)

The model allows the mechanical designer to explore different design alternatives by adjusting model parameters for different design scenarios. If a model parameter is adjusted to a value that makes it impossible to simulate the physical behavior, or the parcel will be sliding off the trolley, the simulation will stop automatically. If the parcel is sliding off the trolley it will also be visible in the 3D animation as shown in figure 6.7. In the co-simulated model only the corners have contact with the trolley, so the corners falls of the trolley, and thus in the simulation the parcel goes partly through the tray as illustrated in the figure.

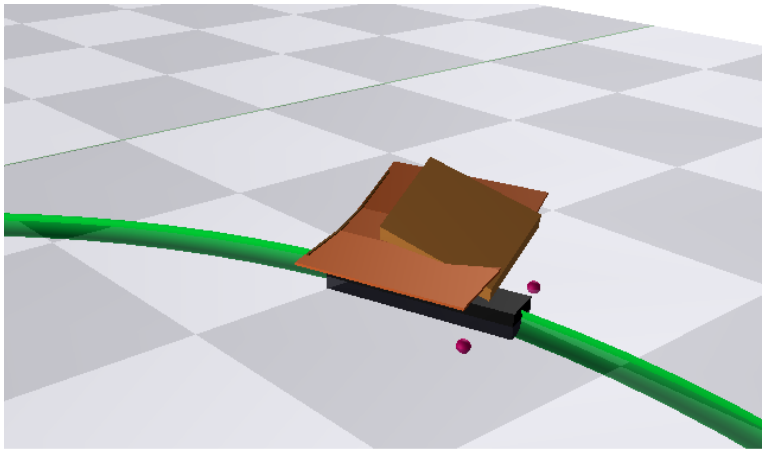


Figure 6.7: Parcel sliding off at conveyor speed 3.0 m/s and d = 0.5 (Translation friction)

7

Related Work

The goal of the DESTECS tool is to support modeling of embedded control software in relation to the environment. A model is a simplification of a system entity under design. The model contains characteristics and properties of the system entity that are relevant for a given goal. A model is minimal with respect to the given task and goal of the model. The goal for the DESTECS tool and methodology is to combine continuous-time models with discrete-event controller models [30]. The DESTECS tool combines the tools 20-sim and VDM++ for combined co-simulated models. Co-simulation of these models allows developers to use multi-domain fault injection tests to build more dependable real-time embedded systems [31].

The Mathwoks, Inc. provides a well-known commercial tool suite (MATLAB/Simulink) for modelling of continuous-time systems targeting different domains. Simulink is available to combined with toolboxes for domains like control engineering and signal processing. Simulink combined with the stateflow toolbox is frequently used in control engineering by the industry. It is a modeling and simulation tool based on mathematical models. Simulink is similar to 20-sim in being a continuous-time graphical modeling tool. Simulink is mainly graphical orientated in its model based approach, where 20-sim combines parameters, equations in a graphical representation of the model. The graphical representation for both tools is intuitive and allows control engineers to focus on the control function, without caring about the detailed coding and implementation platform. Simulink is based on synchronous data flow graphs (SDF) and supports modeling of multi-rate systems. The stateflow toolbox combined with Simulink makes it possible to model state based controllers in combination with a continuous-time model of the environment.

Tool suppliers have attempted to bridge the gab between continous-time and discrete-event models by coupling UML tool for software engineering with Simulink. This is good step forward, the coupling is based on generated source code from the Simulink and IBM Rhapsody (UML) toolkits. It requires a lot of implementation details for the descrete-event part of the UML model before a co-simulation is possible. The DESTECS tool integrates the co-simulated models at a higher abstraction level that enables the developer to generate models faster focusing on the modelling functionality.

The Ptolemy [32] project focuses on modeling, simulation and design of heterogeneous systems. Emphasis is on embedded systems that mix different technologies and, accordingly, also Model of Computation (MoC). Ptolemy supports different types of applications, including signal processing and control application like being the target for DESTECS. Ptolemy supports different MoCs and corresponding domains like continuous time for mechanical systems and discrete event models. The tool do not focus on fault injection as part of the DESTECS project neither is it based on a formal modelling language like VDM.

8

Conclusions and Future Work

In this paper we have demonstrated a CT-first approach to the construction of co-models. We have show how co-simulation between a continuous time and a discrete event model can be used in order to gather a deeper understanding across disciplinary boundaries. This have been demonstrated on an industrial case on a part of a trolley-based conveyor system.

The project did succeed to create a co-simulated model containing a continuous-time (CT) model in 20-sim co-simulated with a discrete-event (DE) model in VDM-RT with the first versions of the DESTECS tool. The project has proven to be very promising based on the feedback from the company. The experience gained from the DESTECS tool and methodology is good, even though the tool currently is in a development phase, it was possible to create and demonstrate a working co-simulated model. The simulation speed is still slow for a commercial use of the tool especially if many different designs alternatives should be investigated.

The contact-model between parcel and trolley has not been verified against a real system. Measurements should be done on a real system to obtain the proper parameters for the contact-model. There is a risk that no correct parameters can be found for the current model, since the current contact-model does not describe non-linearities like stiction or deformation of the parcel.

Design space exploration and fault injection planned to be covered by the tool has not been evaluated in this project. Design space exploration by executing scenarios with variable design parameters between automated simulations would be a great feature for this project. The company could use such a functionality to specify a number of test scenarios that could be verified by the tool. This would be the case in exploring handling different parcels (dimensions, frictions and mass) as a final verification of a a control for the tilting of the trolleys in curves.

The capabilities demonstrated in this work was very well received by the company when it was demonstrated to them. This have clearly resulted in their increased interest in analysis of precise models in their future work. If you have a good model describing the real thing, it is possible to do all kinds of different scenario's. Even with different parcels, trajectories, speeds etcetera.

The next step in our work will be to verify the simulated model against a real conveyor system using the controller developed in this research. It is very important with such a validation to ensure confidence in the fidelity of the model within the company management and engineers.

We sincerely hope that others will also be able to benefit from the DESTECS methodology and tool support used on this case study. We believe that this technology will be beneficial in many cases where a system-wide multidisciplinary approach can be applied.

Acknowledgements

We would like to thank all the employees at the company involved in this study as well as all the stakeholders from the DESTECS project enabling this technology becoming a reality. Special thanks to Frank Groen from Controllab Products B.V. as he contributed major work on the 20-sim and the 3D mechanical models. Support for this work has partially been provided by funding from the EU FP7 DESTECS project. Finally we would like to especially thank Sune Wolff for reading earlier drafts of this paper.

Bibliography

[1] D. Bjørner and C. Jones, eds., *The Vienna Development Method: The Meta-Language*, vol. 61 of *Lecture Notes in Computer Science*. Springer-Verlag, 1978.

[2] J. S. Fitzgerald, P. G. Larsen, and M. Verhoef, "Vienna Development Method," *Wiley Encyclopedia of Computer Science and Engineering*, 2008. edited by Benjamin Wah, John Wiley & Sons, Inc.

[3] R. S. Sandford Friedenthal, Alan Moore, *A Prictical Guide to SysML*. Friendenthal, Sanford: Morgan Kaufman OMG Press, First ed., 2008. ISBN 978-0-12-374379-4.

[4] J. F. Broenink, P. G. Larsen, M. Verhoef, C. Kleijn, D. Jovanovic, K. Pierce, and W. F., "Design support and tooling for dependable embedded control software," in *Proceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems*, pp. 77–82, ACM, April 2010.

[5] C. B. Jones, "Scientific Decisions which Characterize VDM," in *FM'99 - Formal Methods* (J. Wing, J. Woodcock, and J. Davies, eds.), pp. 28–47, Springer-Verlag, 1999. Lecture Notes in Computer Science 1708.

[6] ISO, "Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language," December 1996.

[7] P. G. Larsen and W. Pawłowski, "The Formal Semantics of ISO VDM-SL," *Computer Standards and Interfaces*, vol. 17, pp. 585–602, September 1995.

[8] C. B. Jones and K. Middelburg, "A typed logic of partial functions reconstructed classically," Tech. Rep. 89, Department of Philosophy, Utrecht University, April 1993.

[9] J. Bicarregui, J. Fitzgerald, P. Lindsay, R. Moore, and B. Ritchie, *Proof in VDM: A Practitioner's Guide*. FACIT, Springer-Verlag, 1994. ISBN 3-540-19813-X.

[10] J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef, *Validated Designs for Object–oriented Systems*. Springer, New York, 2005.

[11] OMG, "Omg unified modeling languagetm (omg uml), superstructure specification ver. 2.3." *http://www.uml.org/spec/UML/2.3/Superstructure*, 2010.

[12] J. Fitzgerald, P. G. Larsen, and S. Sahara, "VDMTools: Advances in Support for Formal Modeling in VDM," *ACM Sigplan Notices*, vol. 43, pp. 3–11, February 2008.

[13] P. G. Larsen, N. Battle, M. Ferreira, J. Fitzgerald, K. Lausdahl, and M. Verhoef, "The Overture Initiative – Integrating Tools for VDM," *ACM Software Engineering Notes*, vol. 35, January 2010.

[14] K. Lausdahl, P. G. Larsen, and N. Battle, "A Deterministic Interpreter Simulating a Distributed Real Time System using VDM," in *ICFEM 2011*, October 2011.

[15] P. G. Larsen and J. Fitzgerald, "Recent Industrial Applications of VDM in Japan," in *FACS 2007 Christmas Workshop: Formal Methods in Industry* (J. B. Paul Boca and P. G. Larsen, eds.), Electronic Workshops in Computing, British Computer Society, December 2007.

[16] T. Kurita and Y. Nakatsugawa, "The Application of VDM++ to the Development of Firmware for a Smart Card IC Chip," *Intl. Journal of Software and Informatics*, vol. 3, October 2009.

[17] "Simulink - Simulation and Model-Based Design." http://www.mathworks.com/products/simulink/, 2009.

[18] M. Wetter, "Modelica-based Modelling and Simulation to support Research and Development in Building Energy and Control Systems," *Journal of Building Performance Simulation*, vol. 2, pp. 143–161, June 2009.

[19] C. Kleijn, "Modelling and Simulation of Fluid Power Systems with 20-sim," *Intl. Journal of Fluid Power*, vol. 7, November 2006.

[20] A. Bos, *Modelling multibody systems in terms of multibond graphs – with application to a motorcycle*. PhD thesis, Faculty of Electrical Engineering, University of Twente, Enschede, Netherlands, December 1986.

[21] C. Kleijn, *20-sim 4.1 Reference Manual*. Enschede: Controllab Products B.V., First ed., 2009. ISBN 978-90-79499-05-2.

[22] E. Kreyszig, *Advanced Engineering Mathematics*. New York, 605 Third Avenue: John Wiley & Sons, Inc., Fifth ed., 1983. ISBN 0-471-88941-5.

[23] J. McCrae and K. Singh, "Sketching piecewise clothoid curves," *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*, 2008.

[24] H. Makino, "Clothoidal interpolation – a new tool for high-speed continuous path control," *Annals of the CIRP - Maufacturing Technology*, vol. 37, no. 1, pp. 25–28, 1988.

[25] "Systems modeling language (sysml) specification," Tech. Rep. Version 1.0, SysML Modelling team, November 2005. http://www.sysml.org/artifacts/specs/SysMLv1.0a-051114R1.pdf.

[26] J. van Amerongen, *Dynamical Systems for Creative Technology*. Controllab Products, Enschede, Netherlands, 2010.

[27] E. J. Haug, *Kinematics and Dynamics of Mechanical Systems*. Needham Heights, Massachusetts 02194: Allyn and Beacon, First ed., 1989. ISBN 0-205-11669-8.

[28] S. Stramigioli, "Geometric modelling of mechanical systems for interactive control," *Lecture from Delft Univerity of Technology, The Netherlands*, 2005.

[29] J. F. Broenink, J. Fitzgerald, K. Pierce, Y. Ni, and X. Zhang, "Methodological guidelines 1," tech. rep., The DESTECS Project (INFSO-ICT-248134), December 2010.

[30] J. Fitzgerald, P. G. Larsen, K. Pierce, and M. Verhoef, "A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems," *To appear in Mathematical Structures in Computer Science*, September 2011.

[31] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, pp. 75–82, April 1997.

[32] P. Marwedel, *Embedded System Design*. Springer, 2003.

Kim Bjerge and Peter Gorm Larsen: Applying Co-Simulation for an Industrial Conveyor System, 2012

Department of Engineering
Aarhus University
Edison, Finlandsgade 22
8200 Aarhus N
Denmark

Tel.: +45 4189 3000