MCAD2Sim: Towards Automatic Kinematic Joints Recognition

Suthida Thongnuch[1] and Alexander Fay[2]

Institute of Automation Technology, Helmut Schmidt University, Hamburg, Germany
[1]suthida.thongnuch@hsu-hh.de
[2]alexander.fay@hsu-hh.de

Corresponding author: Suthida Thongnuch, suthida.thongnuch@hsu-hh.de

Abstract. Developing production machines involves engineering processes to transform customer requirements into real machines. Commissioning, which is a bottleneck of the engineering processes, must be better addressed to improve the machine development. To do so, with the help of virtual machines, virtual commissioning (VC) can be performed before the real machine construction with several promising benefits. VC, however, seeks a method to automatically generate detailed virtual machines. The automatic model generation makes VC benefits valid. This paper presents a practical and automatic VC model generation method by applying the constraint-based algorithm to MCAD models. As a result, the static geometry in the models governs kinematic joints and parameters. The proposed MCAD2Sim workflow produces executable kinematic models in the COLLADA format, which is a part of AutomationML and widely used in the industrial automation domain. Furthermore, in this paper, the application of the entire workflow on a mechanical assembly is demonstrated. The results serve as a preliminary solution to the automatic VC model generation for more sophisticated real-world applications.

Keywords: virtual commissioning, kinematics model, COLLADA, mechanical computer-aided design.
DOI: https://doi.org/10.14733/cadaps.2020.44-60

1 INTRODUCTION

Developing special-purpose production machines transforms customer requirements into real physical machines through engineering phases of a project development life cycle. The conventional development life cycle starts with planning, followed by the design (mechanical design, electrical design, and control code creation), construction, and commissioning. Based on the mechanical and electrical design, mechanical parts are manufactured and assembled, and electrical wiring is installed in the construction phase. After that, commissioning is performed on the machine to validate control code before it is in operation. Commissioning, as a transition between the design and the operation, is considered as a crucial stage and a bottleneck. Delay in

commissioning directly impacts the project lead time. In practice, commissioning covers not only control code validation, but the overall machine functions test including electrical wiring check, sensors calibration, actuator tuning, etc. Control code bugs and other design flaws are discovered and rectified in the commissioning phase. More than 85% of the faults and errors found during commissioning come from previous phases [31]. Additionally, the quality of the machine must be ensured at an acceptable commissioning cost. As a consequence, commissioning is at risk of time delay, over budget, and poor quality.

In order to reduce the commissioning risks, the idea of an early validation using virtual commissioning (VC) has emerged since the last decade [11]. VC, in comparison to the real physical commissioning mentioned in the previous paragraph, adopts models instead of a real machine. Different types of VC models are discussed in section 2. VC is commissioning that tests individual components and functions of the system during the project development using simulation methods and models [30]. VC offers several promising benefits regarding time, cost, and quality. With the help of VC models, the control code is tested thoroughly in every test case scenario. The quantitative benefits of VC are also reported; for instance, commissioning time is cut by 25% to 75% [10][24][36]. As a result, the on-site commissioning expenses such as man-hour costs and test material consumption are reduced. However, VC is seldom used in practice despite plenty of VC-related software tools. At present, the effort to set up VC models is much greater than the returned benefits. VC requires a substantial modeling effort [33] and engineering effort [22] because the modeling process is still a manual, error-prone, and time-consuming [4][5][12][15][32]. VC is, therefore, considered as an additional workload nowadays.

The motivation of this work is to encourage the automatic VC model generation. Rather than modeling from scratch, building it on readily available data or models is preferable. Models such as those from mechanical computer-aided design (MCAD) systems, for example, are usually at hand in the early development process as a result of the mechanical design. However, MCAD models cannot be directly used to perform VC as VC models. They are usually imported into VC simulation tools for visualization purposes. Behavior descriptions such as motions are missing; therefore, the descriptions must be manually added using provided functions in the tools. In the context of VC models creation, MCAD models contain a lot of information [2][17]. Geometry and kinematic relations can be extracted from MCAD models and utilized in VC models besides installed actuators and sensors [17]. Exploiting such information can reduce the effort to set up VC models. Inspired by this fact and constraint-based MCAD systems, the theory and algorithm in mechanical assembly design are applied in this paper to automatically develop VC models from MCAD models. The methodology bridges two different worlds of MCAD systems and VC simulation tools by the MCAD2Sim workflow. Here, the behavior is automatically recognized and extracted from the MCAD models to reduce the effort.

The remaining part of this article is structured as follows. Section 2 reviews state of the art in two aspects: behavior modeling in the context of VC and the MCAD2Sim workflow in research and tools. Section 3 explains the MCAD2Sim approach and the relevant theoretical background. Section 4 demonstrates and exemplifies the approach implementation in detail. Section 5 compares the results with other approaches. Finally, section 6 concludes the result and discusses the advantages and disadvantages of the approach as well as the improvement in future work.

2 LITERATURE REVIEW

A simulation model in the context of VC is not merely a mathematical model. Instead, a geometry model and a behavior model constitute an essential part of a VC model [14][25]. Behavior modeling depends on many factors such as application scenarios and modeling techniques available in the tools. As a result, VC behavior models are rich in diversity. Recently, the Association of German Engineers (VDI) classified four types of VC models in ascending order of model fidelity: event-based, kinematics, kinematics with 3D models, and dynamics model [30]. As the name suggests, event-based models (e.g., Petri nets and state machines) provide binary responses to

events. Building VC models at the event-based level requires knowledge in such kind of formalism (e.g. Petri nets) and a considerable effort to integrate it with geometry. Similarly, a strong background in the laws of physics is required to build dynamics models. Strahilov and Damrath [27] model the linear movement of a pneumatic cylinder using the laws of fluid mechanics. The difference of compressed air pressure in the cylinder chamber determines the force acting on the piston. As a result, at the end of the simulation cycle, a new piston position is calculated. Related formulas and parameters such as the piston area, the chamber volume, friction in the chamber must be provided to physics engines for calculation. The dynamic model of the cylinder is achieved (the ramp up and ramp down of the piston is realized without the constant-speed assumption). However, the required effort, knowledge and input data are considerable. From the economic point of view, dynamic models are not always the optimum model because of their complexity and computational expense.

Simulation at the kinematic level, on the other hand, gives the impression of how systems work. The kinematics and kinematics with 3D model require little knowledge to predefine a movement path. They are closely related to the visualization and particularly useful to validate motion and detect a collision. It is, therefore, reasonable to build the kinematics with the 3D model because the effort spent is relatively low to achieve this high fidelity. Kinematic models serve as the foundation of the dynamic model because motions produced by dynamic models must be kinematically feasible as well [3].

This paragraph reviews the kinematic models as a part of 3D models in research and tools and identifies a research gap. As an essential part of a virtual machine, 3D MCAD models are usually imported into simulation tools and are then manually elaborated with kinematic information. In *Process Simulate*, users can manually configure a kinematic chain of an MCAD model using the *Kinematics Editor* tool. Guerrero et al. [8] demonstrate the application of *Process Simulate* to perform VC of a pick and place system. They manually configure and parametrize a kinematic chain of the 3D MCAD model of the pick and place system using *Kinematics Editor*. The process is cumbersome and tedious. Hoffmann et al. [12] propose a workflow transforming MCAD to a VC model. Their workflow begins with the import of a simplified MCAD model into the robot simulation tool called *Ciros*, then manually classify/structure components in the model into stationary parts, moving parts (i.e. actuators) and sensors, and assign *Ciros* functions to actuators (e.g. translation and rotation) and sensors (e.g. ultrasonic sensor). The resulted simulation model must be further refined with function parameters such as translation stroke and speed. They also conclude that it is impossible or partially possible to transfer MCAD with kinematics from CAD systems to the simulation environment; therefore it is necessary to attach kinematics to geometry manually. It is evident that kinematic behavior generation is still a manual process, and the automatic MCAD-to-simulation workflow is missing.

3 MCAD2SIM APPROACH

This paper proposes the MCAD2Sim approach aiming to reduce the manual creation of kinematics with 3D models generation. As the name suggests, the approach produces simulation models from MCAD models as depicted in Figure 1. An MCAD model designed in an MCAD system is converted into a 3D kinematics model represented by a kinematic chain. The algorithm adopted in this paper translates assembly constraints to kinematic joints which is a part of the chain. The kinematic model as an executable simulation model is in the COLLADA Kinematics format. The following subsections explain the related foundation applied in this approach.

3.1 Assembly Modeling in MCAD Systems

Products designed in MCAD systems are mechanical assemblies. In MCAD systems, it is necessary to model individual parts and put them together as an assembly for several reasons such as separating or reducing materials, allowing disassembly or repair, visualizing relative motions and

spatial relationship between parts, and creating a part list [18]. Assembling parts in MCAD systems is commonly fulfilled by assembly constraints.

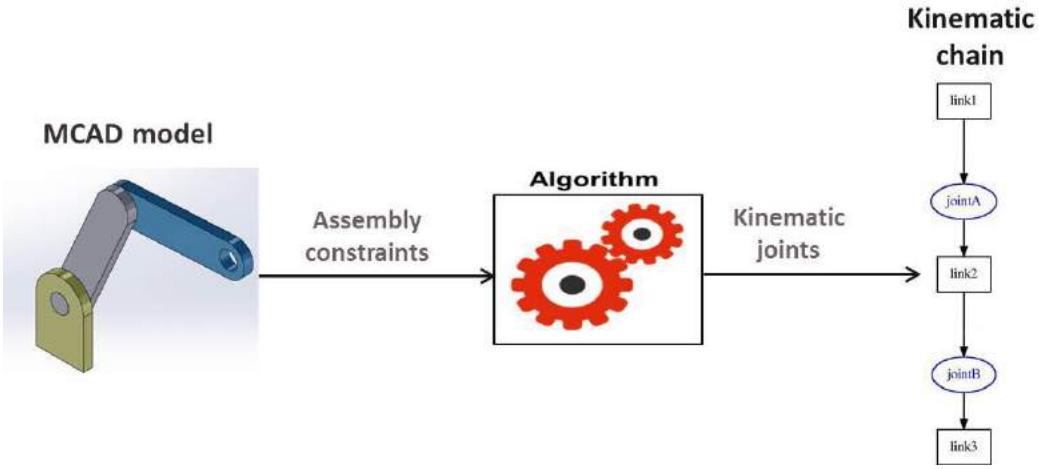


Figure 1: The MCAD2Sim workflow.

The assembly constraints define relationships between geometric elements of joined parts such as points, planes, axes, and surfaces. For example, as shown in Figure 2, the arm of the assembly shown in Figure 1 is joined to the base by two assembly constraints. The concentric constraint connects the cylindrical surfaces so that they share the center axis. The coincident constraint brings the two flat surfaces into contact. Without these constraints, the arm can move freely in the 3D space, i.e., six DoFs (three translations and three rotations along/around the X-, Y-, and Z-axis). The concentric and coincident constraints allow the arm to rotate around the base. Thus, the mobility is reduced to one DoF. Commercial MCAD systems offer several types of assembly constraints.

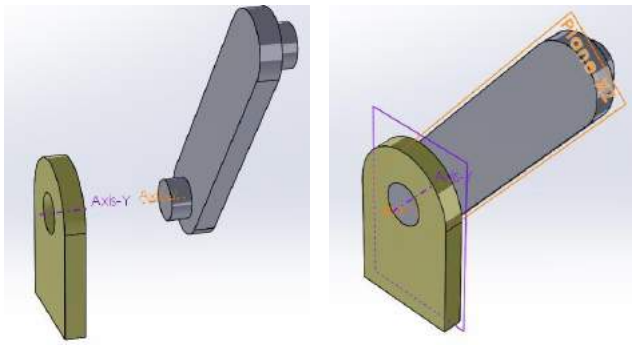


Figure 2: The line-line and plane-plane coincident assembly constraints.

3.2 Kinematic Joint Types

As shown in the left side of Figure 1, the assembly consists of three parts (or links) and two joints (or kinematic pairs). A combination of links and joints forms a kinematic chain (see the right side of Figure 1). A kinematic chain is a collection of links and joints interconnected to provide the output motion corresponding to the given input motion [21]. Generally, links are related to geometry and are, therefore, visually seen. Between the links, there is a joint which allows or restricts some relative motions of the links. As depicted in Figure 2, a revolute joint is a result of constraining the arm to its parent link. Joints are not obviously seen in comparison to links.

3.3 Mapping Constraints to Kinematic Joints

This subsection explains the algorithm that associates joint types with assembly constraints. MCAD models contain abundant information including the implicit kinematic information [2][17]. Assembly constraints can be mapped to kinematic joints. Kim et al. [16] establish a rule to detect kinematic joints based on two key geometry characteristics: IPV (Independent Principal Vectors) and IMG (Intersection of Mating Geometries). The number of IPVs and types of IMG determine the rotational and translational DoFs, respectively. Their combination results in the total number of DoF and kinematic joint type. These characteristics are the result of applied assembly constraints. As illustrated in subsection 3.1, when applying a constraint, geometric elements (e.g., lines, planes) must be used. Their intersections are the IMGs. The assembly shown in Figure 1 uses the axis and plane as the geometric elements. Their intersection or IMG is the point as shown in Figure 3. A point IMG allows no translation. Direction vectors of the axis and plane determine the IPV which is the number of independent direction vectors. As depicted in Figure 3, *vector1* is the direction vector of the axis. Similarly, *vector2* which is normal to the plane is the direction vector. IPV is, therefore, equal to one because both vectors point in the same direction. One IPV allows one rotation. Therefore, one IPV and the point IMG result in the revolute joint as shown in Table 1.

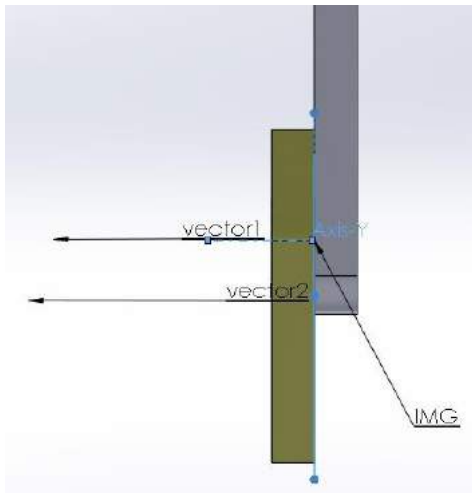


Figure 3: The side view of the assembly in Figure 1 shows the IMG and direction vectors resulting from assembly constraints.

Also, as seen in Figure 3, the resulting IMG and IPV can be mapped to a combination of constraints: the line-line coincident, the plane-plane coincident, which are perpendicular. The mapping of constraints to the kinematic joints is proposed by Chang [6]. Table 1 summarizes the mapping. Besides the joints in Table 1, bolted joints are ubiquitous in mechanical assemblies. Bolts, screws, and nuts are used in these joints to fix parts together. However, they must be removed from simulation models because they are irrelevant and contain too much detail for the simulation [12][26]. Therefore, this study replaces them with equivalent assembly constraints that lock movement in all directions resulting in fixed joints.

4 MCAD2SIM IMPLEMENTATION

This section exemplifies the MCAD2Sim approach using an example shown in subsection 4.1. The assembly constraints used in the prototype are explained here. Subsection 4.2 explains how the

approach is implemented to recognize kinematic joints and how the results are formulated for the simulation.

Joint types	*Number of IPVs*	*IMG*	*A possible combination of constraints [6]*
Prismatic	2	Line	line-line coincident || plane-plane coincident
Revolute	1	Point	line-line coincident ⊥ plane-plane coincident
Planar	1	Plane	a plane-plane coincident
Cylindrical	1	Line	a line-line coincident
Spherical	0	Point	a point-point coincident

Table 1: The mapping of assembly constraints and kinematic joints.

4.1 Prototype

This study adopts the XZ-Cartesian robot shown in Figure 4 as the case study. It is assembled in the MCAD system using the bottom-up approach, i.e., each component or sub-assembly is modeled or obtained from component manufacturers and is joined together by assembly constraints. MCAD models of every component except component (4) are products of *Festo* and are downloaded directly from the company website. The product codes and part numbers are indicated as shown in the feature tree of Figure 4.

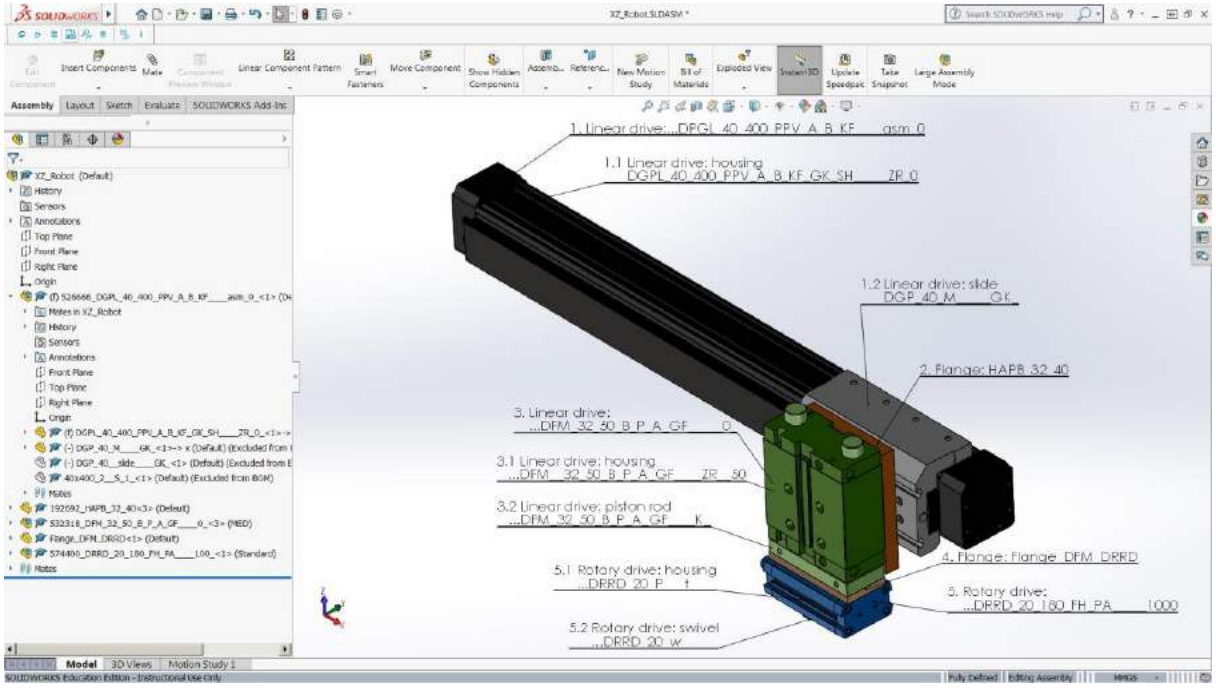


Figure 4: The XZ robot.

Table 2 shows how components in the prototype are connected. Component (1) is a pneumatic-driven linear drive consisting of the housing (1.1) and the slide (1.2). The slide (1.2) is assembled to the housing (1.1) using the line-line and plane-plane coincident as shown in Table 2 (see item 1). Since they are parallel, they result in a prismatic joint. The item 2 of Table 2 shows the flange (2) connecting to the slide (1.2) using the plane-plane coincident and the lock constraint. These

constraints replace bolts and slot nuts that fasten the flange to the slide. Similarly, as shown in the item 3 of Table 2, another side of the flange (2) is fixed to the linear drive (3) using the plane-plane coincident and the concentric with lock rotation. The latter constraint replaces four through-hole bolts locking the linear drive (3) to the flange (2).

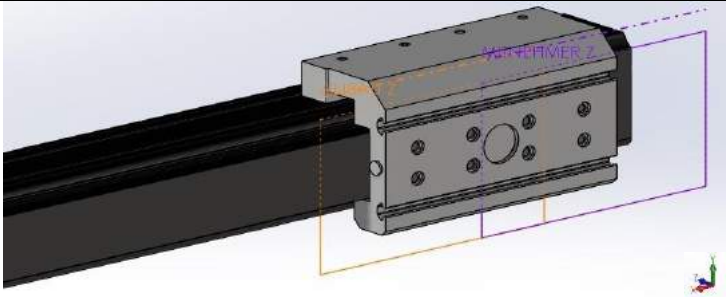
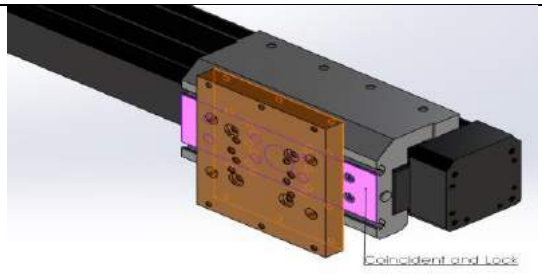
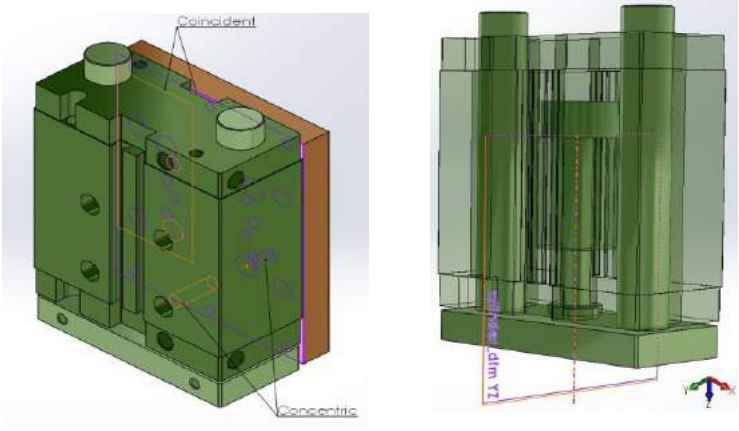
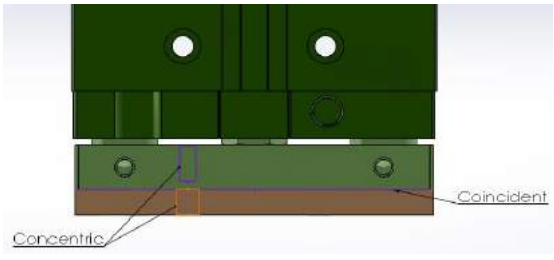
No.	*Components*	*Constraints*	*Joints*
1		line-line coincident || plane-plane coincident	prismatic
2		plane-plane coincident & lock constraint	fixed
3		(left) plane-plane coincident & concentric constraint with lock (right) line-line coincident || plane-plane coincident	(left) fixed (right) prismatic
4		plane-plane coincident & concentric constraint with lock	fixed

Table 2: The list of assembly constraints in the XZ robot.

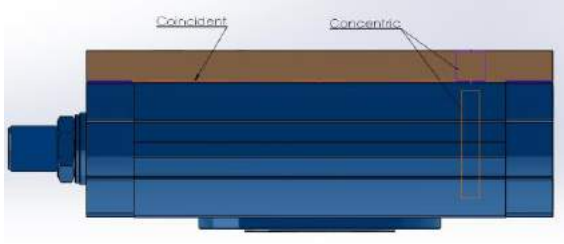
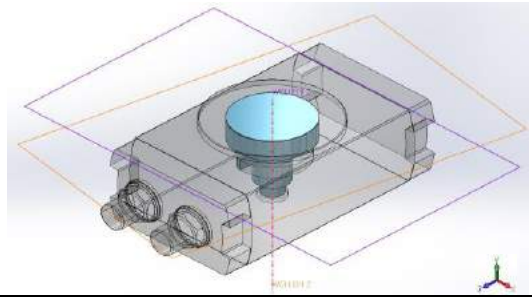
No.	Components	Constraints	Joints
5		plane-plane coincident & concentric constraint with lock	fixed
6		line-line coincident ⊥ plane-plane coincident	revolute

Table 2: The list of assembly constraints in the XZ robot (continued).

4.2 Methodology

Figure 5 presents the implementation process to recognize kinematic joints and determine joints parameters from an assembly in MCAD systems. The process starts from the root component or the assembly's ground which is fully constrained (or fixed). The root of this assembly is the component (1). For the sake of simplicity, if a component is a sub-assembly, it is assumed that it consists of two internal parts. Two end stops of (1) are, therefore, combined with (1.1). Also, in each sub-assembly, one component must be fixed, and another one is free to move. This defines the parent-child role. Constraints of the parent and child entity are retrieved and analysed. After the analysis, the result is written to the output file. The process is repeated until the end of the chain is found.

In case constraints consist of the plane-plane and line-line coincident, vectors of the plane and line are used to determine the angle between them using the dot product as shown in Equation (4.1):

$$\mathbf{V}_l \bullet \mathbf{V}_p = v_{lx} \times v_{px} + v_{ly} \times v_{py} + v_{lz} \times v_{pz} = \left|\mathbf{V}_l\right| \times \left|\mathbf{V}_p\right| \times \cos\theta \tag{4.1}$$

, where $\mathbf{V}_l = \begin{bmatrix} v_{lx} & v_{ly} & v_{lz} \end{bmatrix}$ is the vector of the line, $\mathbf{V}_\mathbf{p} = \begin{bmatrix} v_{px} & v_{py} & v_{pz} \end{bmatrix}$ is the normal vector of the plane, and θ is the angle between the line and plane. The angle θ, therefore, decides on the rotation or translation process. The line vector $\mathbf{V}_\mathbf{l}$ is passed to these processes as the rotation or translation axis as well as the in the transformation matrix **M** of the child entity as shown in Equation (4.2).

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{T} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \tag{4.2}$$

The transformation matrix **M** consists of the rotation matrix **R** and the translation vector **T** indicating the orientation and position of an object in the 3D space. As shown in Figure 6, after the rotation axis is determined, the mobility of the child entity is determined by incrementally turning it one degree around the axis until the collision is detected. The rotation around the X-, Y-, and Z-axis is specified using the rotation matrices shown in Equation (4.3). Similarly, the translation moving range is determined by incrementally move the entity 1mm along the translation axis.

$$\mathbf{R}_x(\alpha)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\alpha & \sin\alpha\\ 0 & -\sin\alpha & \cos\alpha\end{bmatrix};\mathbf{R}_y(\beta)=\begin{bmatrix}\cos\beta & 0 & -\sin\beta\\ 0 & 1 & 0\\ \sin\beta & 0 & \cos\beta\end{bmatrix};\mathbf{R}_z(\gamma)=\begin{bmatrix}\cos\gamma & \sin\gamma & 0\\ -\sin\gamma & \cos\gamma & 0\\ 0 & 0 & 1\end{bmatrix} \tag{4.3}$$

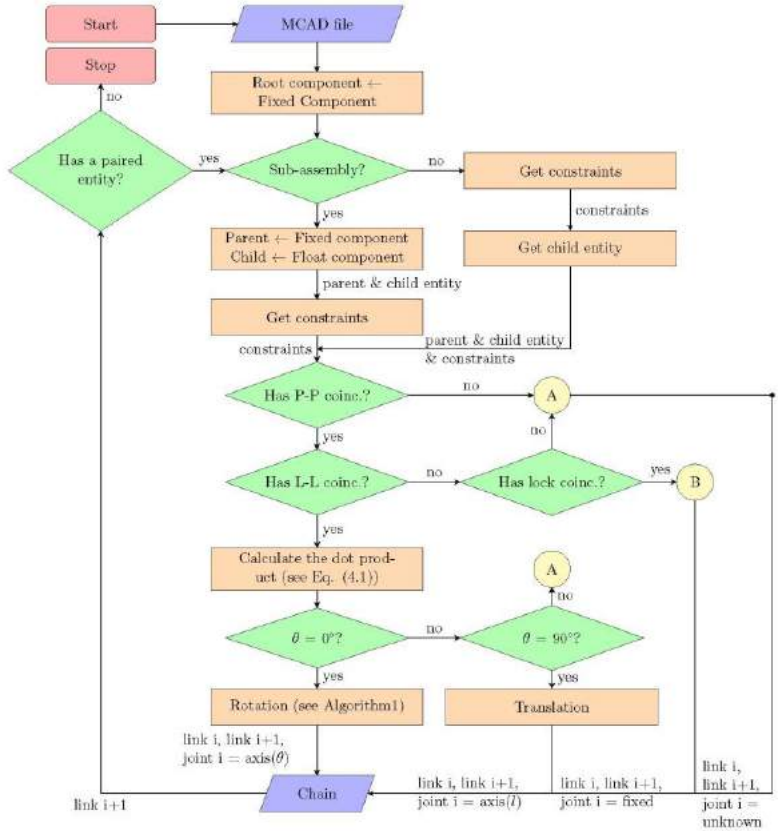


Figure 5: The flow chart shows the implementation process.

The internal structure and geometry of components allow and restrict some movements; therefore, they play an important role in collision detection. As a result, the slide (1.2) of the component (1) moves along the positive X-axis until it hits another end stop (or, in practice, shock absorbers). Similarly, the piston rod (3.2) moves along the positive Z-axis until it hits another end of the chamber. The swivel (5.2) rotates around the Y-axis in the CCW direction until it hits another end of the groove.

After the collision is detected, the transformation matrix is updated. The transformation matrices before and after rotation are converted into the quaternion (see Equation 4.4) and axis-angle representation (see Equation 4.5) for the swept angle calculation. The orientation before and

after the rotation are as follows: $\mathbf{R}_1 = \begin{bmatrix} -0.1736 & 0 & 0.9848 \\ 0 & 1 & 0 \\ -0.9848 & 0 & -0.1736 \end{bmatrix}; \theta_1 \mathbf{e}_1 = \begin{bmatrix} 0 \\ -100 \\ 0 \end{bmatrix}$ and $\mathbf{R}_2 = \begin{bmatrix} -0.1736 & 0 & -0.9848 \\ 0 & 1 & 0 \\ 0.9848 & 0 & -0.1736 \end{bmatrix}; \theta_2 \mathbf{e}_2 = \begin{bmatrix} 0 \\ 100 \\ 0 \end{bmatrix}$, resulting in the swept angle of 200 degrees. However, the sliding distance calculation is more straightforward. The subtraction of the translation vector **T** after and before the movement results in the distance travelled. The maximum allowable distance of 466 mm of the component (1) is calculated from the difference of [582.53 0 0] and [116.53 0 0]. The 49-mm stroke of the component (3) is calculated in the similar way.

Algorithm 1 Determine the rotation axis **e** and rotation angle θ

Input: the unit vector of the line-line coincidence $\mathbf{V_l}$ and the transformation matrix of the child entity before rotation in the local and world coordinate system (LCS and WCS) $\mathbf{M_1}$ and $\mathbf{M_{wcs,1}}$

Output: the rotation axis (in the WCS) **e** and the swivel angle θ

$\mathbf{M_2} \leftarrow$ new matrix 4x4 ▷ new transf. matrix (after rotation)
if $\mathbf{V_l} = [\pm 1 \quad 0 \quad 0]$ **then** ▷ determine the rot. axis
  while clash is off **do** rotate $(\mathbf{R_x}(\pm \mathbf{1}^\circ))$
  end while ▷ rotate child entity using rotation matrix (Eq. 4.3)
else if $\mathbf{V_l} = [0 \quad \pm 1 \quad 0]$ **then**
  while clash is off **do** rotate$(\mathbf{R_y}(\pm 1^\circ))$
  end while
else
  while clash is off **do** rotate$(\mathbf{R_z}(\pm 1^\circ))$
  end while
end if
$\mathbf{M_2} \leftarrow$ the current position of the child entity ▷ after rotation
▷ rot. matrix -> quaternion (Eq.(4.4))
$\mathbf{q_1} \leftarrow \mathbf{R_1}.Quaternion$ and $\mathbf{q_2} \leftarrow \mathbf{R_2}.Quaternion$
▷ quaternion -> axis-angle (Eq. (4.5))
$\theta_1 \mathbf{e_1} \leftarrow \mathbf{q_1}.AxisAngle$ and $\theta_2 \mathbf{e_2} \leftarrow \mathbf{q_2}.AxisAngle$
$\begin{bmatrix} i \\ j \\ k \end{bmatrix} \leftarrow \theta_2 \mathbf{e_2} - \theta_1 \mathbf{e_1}$ ▷ calculate the axis-angle difference
$\theta \leftarrow \sqrt{i^2 + j^2 + k^2}$ ▷ magnitude of the resulted vector
$\mathbf{e} \leftarrow \frac{1}{\theta} \begin{bmatrix} i \\ j \\ k \end{bmatrix}$ ▷ direction of the resulted vector
▷ If LCS $\neq$ WCS, calculate the rot. matrix
if $\mathbf{R_1} \neq \mathbf{R_{wcs,1}}$ **then** $\mathbf{R_{lcs->wcs}} \leftarrow \mathbf{R_{wcs,1}} \cdot \mathbf{R_1^{-1}}$
  $\mathbf{q} \leftarrow \mathbf{R_{lcs->wcs}}.Quaternion$ ▷ rot. matrix -> quaternion
  $\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \leftarrow \mathbf{q}.EulerAngles$ ▷ quaternion -> Euler angles (Eq. (4.6))
  $\mathbf{e} \leftarrow \mathbf{e^T} \cdot \mathbf{R_x}(\alpha) \cdot \mathbf{R_y}(\beta) \cdot \mathbf{R_z}(\gamma)$
end if

Figure 6: The algorithm determines the rotation axis and angle.

$$\mathbf{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}} \\ \frac{1}{4q_w}(r_{32} - r_{23}) \\ \frac{1}{4q_w}(r_{13} - r_{31}) \\ \frac{1}{4q_w}(r_{21} - r_{12}) \end{bmatrix} \tag{4.4}$$

$$\theta\mathbf{e} = \theta \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} = 2\cos^{-1}(q_w) \bullet \frac{180}{\pi} \begin{bmatrix} \frac{q_x}{\sqrt{1 - q_w^2}} \\ \frac{q_y}{\sqrt{1 - q_w^2}} \\ \frac{q_z}{\sqrt{1 - q_w^2}} \end{bmatrix} \tag{4.5}$$

, where θ is a scalar value of the rotation angle and $\mathbf{e}$ is a unit vector.

Since it is commonplace that components in an assembly have different coordinate systems, it is absolutely necessary to check if components' coordinate systems (LCS) are equivalent to the world coordinate system (WCS) (cf. the item 1, 3, and 6 in Table 2 and Figure 4). The resulted rotation matrix is converted to the Euler angles (see Equation (4.6)). The rotation axis is rotated to the WCS by rotation angles in X-, Y-, and Z-axis are applied to the rotation axis. The Y-up coordinate system of the component (1) is converted to the Z-up orientation by rotating the X- and Z-axis 90 and 180 degrees, respectively. Therefore, the positive X-axis translation axis is multiplied with $\mathbf{R_x}(90^o)\bullet\mathbf{R_z}(180^o)$ resulting in negative X-axis in the WCS. The positive Z-axis of component (3) is transformed using $\mathbf{R_x}(180^o)\bullet\mathbf{R_z}(90^o)$ resulting in the negative Z-axis. The positive Y-axis as the rotation axis is rotated to the WCS using $\mathbf{R_x}(-90^o)\bullet\mathbf{R_z}(90^o)$ resulted in the negative Z-axis. Every joint is then referred to the same coordinate system.

After the joint types and parameters are determined, the resulted kinematic pair is written to the "Chain" as the output of the process. The process proceeds to the next paired link. It is iterated until there is no connected links to consider. The generated output is formulated into a standard file exchange presented in the next subsection.

The presented workflow and algorithm is implemented in the stand-alone C# application. The application interacts with *SolidWorks* via *SolidWorks API*. The matrix **M** of each component is retrieved by the "Transform2". The command "getMates()" is used to read out the assembly constraints. For example, the line-line coincident is recognized by the command "swMateCOINCIDENT" and "swMateEntity2ReferenceType_Line". The component movement is performed using the "drag()" method with "CollisionDetectionEnabled" whose parameter is the incremental translation or rotation displacement.

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} \arctan 2(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2)) \\ \arcsin(2(q_w q_y - q_z q_x)) \\ \arctan 2(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2)) \end{bmatrix} \tag{4.6}$$

4.2.1 Compile the results into COLLADA

The generated output (produced in Figure 5) is formulated into COLLADA 1.5 (*.dae) according to its specification [13]. COLLADA is the XML-based standard data format (IEC 62714-3) to exchange geometry and kinematics. It is under the collective body of standard, called AutomationML which is the standard data exchange format (IEC 62714) in industrial automation.

Figure 7 and 8 show the generated kinematic chain in COLLADA format. Figure 7 shows the collection of joints stored in the <library_joints> as well as the COLLADA structure. Joint 2, 3, 5, and 6 are the fixed joints, but are labeled with the <revolute> tag whose <axis> value is null because the COLLADA syntax supports only prismatic and revolute joint. Figure 8 presents the hierarchy of kinematic chain (up to link3 – due to the space limitation) stored in the <kinematics_model>. The hierarchy saves only references to links and joints. Joints are instantiated and linked to their definition in the <library_joints>. The geometry of links stored in the <library_geometries> and <library_visual_scene> is referred to by the attributed "sid". These libraries are the visualization part of COLLADA. The visualization part, also known as COLLADA 1.4, is ideally exported from MCAD systems. In reality, however, COLLADA 1.4 is not supported by MCAD systems [28]. Therefore, each component of the XZ robot as a link is exported as the STL file. Each STL file is then converted into COLLADA 1.4 using the *MeshLabServer* command of *Meshlab* [7].

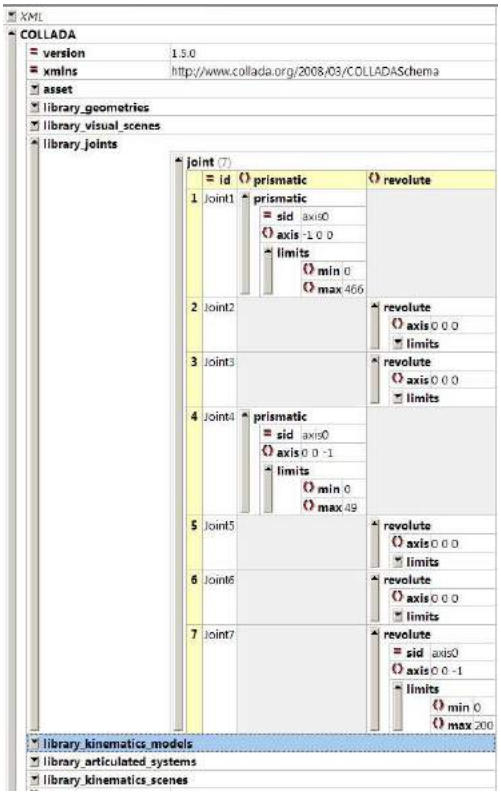


Figure 7: A snippet of the generated COLLADA: library_joints.

4.2.2 Import COLLADA into simulation tools

To present the usability of the result, this subsection demonstrates the import of COLLADA 1.5 into a simulation tool such as *RobotStudio*. The generated COLLADA is imported into *RobotStudio* as a part of AutomationML using the "AutomationML Explorer" add-in developed by Thongnuch et al. [29] as shown in Figure 9. According to AutomationML, the XZ robot, as one of the plant asset, is

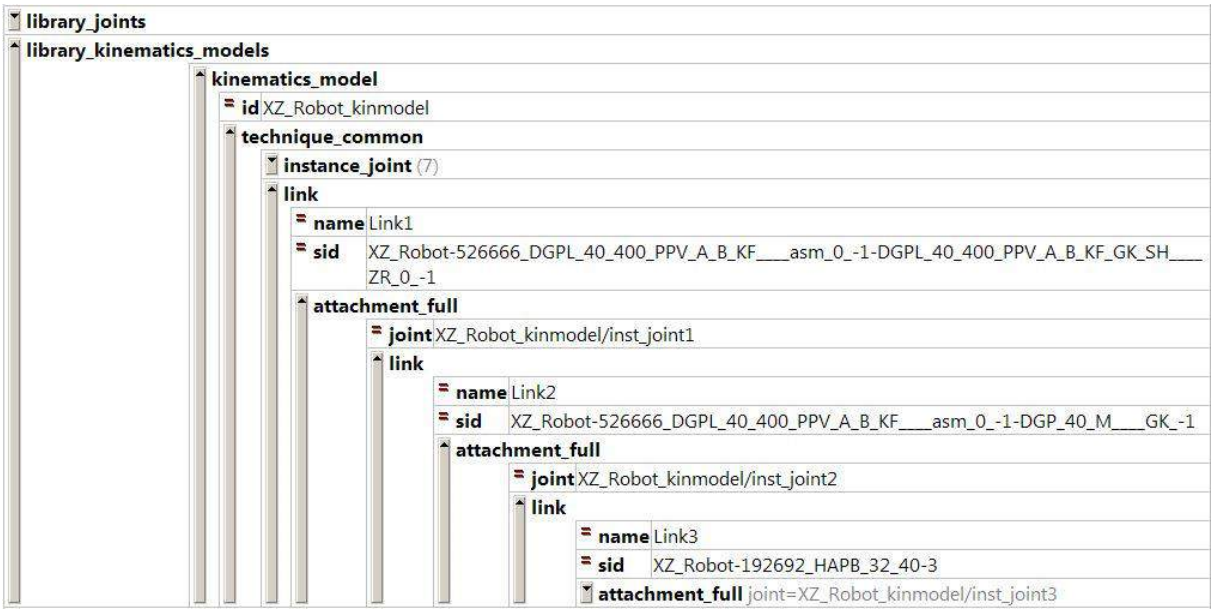


Figure 8: A snippet of the generated COLLADA: library_kinematic_models.

referred to as an IE (InternalElement) whose geometry and kinematics are described by COLLADA (XZ_Robot.dae).

The geometry and kinematics information in COLLADA is compiled to a mechanism (of a device type) in *RobotStudio*. The *RobotStudio* mechanism is a group of graphic components consisting of links and joints [1]. The geometry part (<library_visual_scenes> and <library_geometries>) is mapped to links as shown in the "Layout" tab. Then, the kinematic part (<library_joints> and <library_kinematics_models>) defines how the links are connected (prismatic, revolute, or fixed). The result of the compiled kinematic chain is shown in Figure 9. Users can move the kinematic chain within the permissible range. For example, joint 1 and 2 are moved by 346mm and 48mm, respectively. The distance travelled can be verified by the position of links as seen in Figure 9.

5 COMPARATIVE APPROACHES

This section compares the results with recent related works as shown in Table 3. We categorize the existing works based on three aspects: level of automation, model completeness, and output. The manual kinematic-type VC model creation such as in [8] is the lowest level of automation. Since the workflow presented in [8] is in PLM, the output remains usable within the PLM. The higher and desirable level of VC model generation is (semi-) automatic. Neugebauer and Schob [20] explain the concept in general to transform MCAD models and electrical circuits to VC models. However, no algorithm to extract kinematics from MCAD models is mentioned.

To some degree, our approach is comparable to [23]. The constraint-joint mapping also relies on similar assembly constraints. A similar mechanical assembly in [28] is translated into a kinematic chain using [23]. The result shows that revolute joints are correctly detected while other joint types need correction. Moreover, joints in sub-assemblies are not detected because the algorithm takes only constraints at the top-level assembly into consideration. The resulted mapping produces the kinematic chain in the Simulink block diagram in which joint direction and limits must be manually specified. Specifying joint directions by base and follower frames in the 2-D block diagram seems complicated.

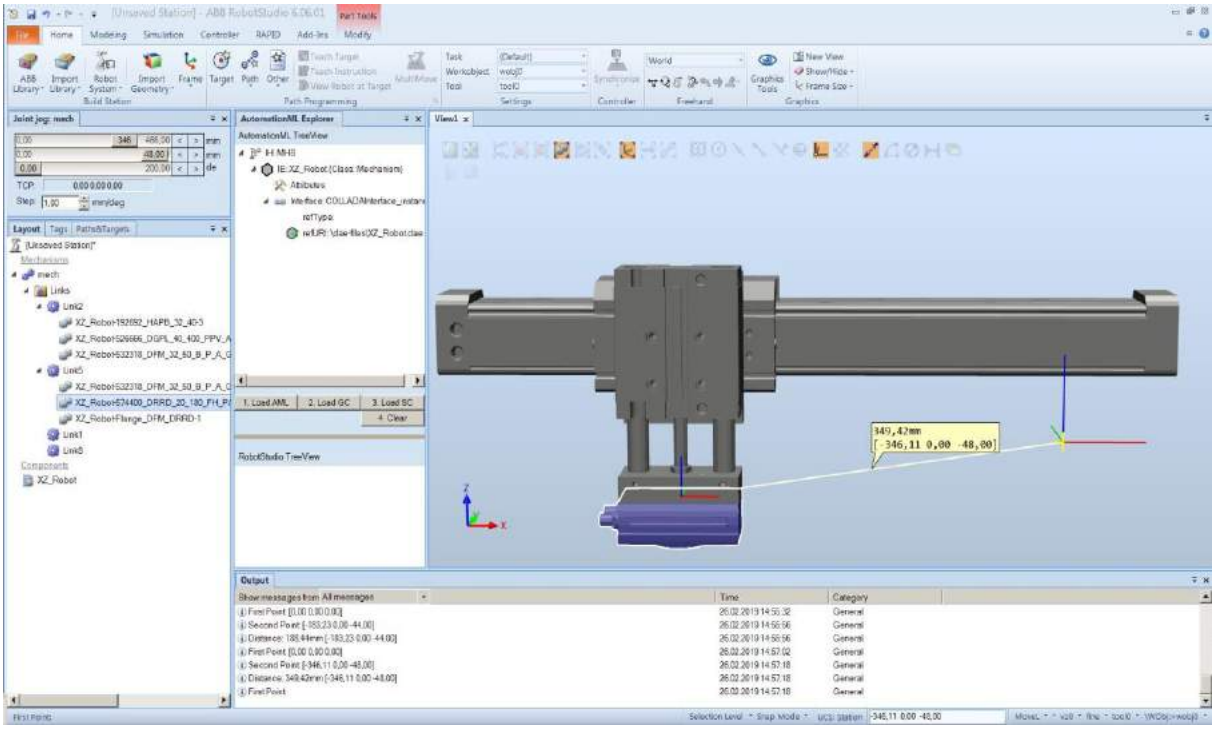


Figure 9: The resulted VC model as a mechanism in *RobotStudio*.

6 CONCLUSION AND FUTURE WORK

This paper addresses the high modeling effort that hinders the usage of VC in industry. The result demonstrates that applying the MCAD2Sim methodology to MCAD models can automatically produce kinematic-type VC models. The methodology applies the rule-based algorithm to basic constraints to extract joint types and determine joint parameters from the geometry. The described approach covers the entire workflow to executable VC models in the standard format. The generated kinematic model serves as the basis for the dynamic model. The generated kinematic model can be supplemented by dynamic behavior (e.g. a ramp-up and -down behavior and spring and damping of joints) and is driven by connected output signals.

The algorithm as the core of the methodology still has room for improvement. The current applied algorithm is straightforward, but it relies solely on explicit and basic constraints. Compliance with the mapping rule, which requires good design practices, is a must. To cover with more variations of constraints in MCAD models, future work takes geometry into account to deal with non-conforming constraints. The screw theory [34] and slippage motion analysis [35] could improve the algorithm. Additionally, the applied algorithm can detect only lower kinematic pairs (as listed in Table 1) in which the parent and child link share the surface contacts. Extending the algorithm to cover higher kinematic pairs (e.g., gears, cams) whose link contacts are point and line as proposed by Mitra et al. [19] is also an option.

Approaches	Manual	(Semi) Automatic				VC models in standard format
		Concept	Implementation		Applied algorithm	
			Kin. joint without param.	Kin. joint with param.		
Guerrero et al. [8]	x					no
Neugebauer and Schob [20]		x				
SimScape [23]			x		constraint-joint mapping	no
This study				x	constraint-joint mapping	yes

Table 3: A summary of the recent related works in terms of the level of automation, completeness, and output of the workflow to generate kinematic-type VC models.

ORCID

Suthida Thongnuch, https://orcid.org/0000-0003-4805-0827
Alexander Fay, http://orcid.org/0000-0002-1922-654X

REFERENCES

[1] ABB. (2017). Operating manual - RobotStudio 6.06 Software.
[2] AVANTI Consortium.: AVANTI: Test methodology for virtual commissioning based on behavior simulation of production systems, 2014. http://www.avanti-project.de/files/AVANTI_D1%201_State_of_the_Art_public.pdf
[3] Bender, J.; Erleben, K.; Trinkle, J.: Interactive Simulation of Rigid Body Dynamics in Computer Graphics, Computer Graphics Forum, 33(1), 2014, 246–270. https://doi.org/10.1111/cgf.12272
[4] Bergert, M.; Kiefer, J.: Mechatronic Data Models in Production Engineering. IFAC Proceedings Volumes, 43(4), 2010, 60-65. https://doi.org/10.3182/20100701-2-PT-4011.00012
[5] Botaschanjan, J.; Hummel, B.; Hensel, T.; Lindworsky, A.: Integrated Behavior Models for Factory Automation Systems. In Proceedings of the 2009 IEEE Conference on Emerging Technologies & Factory Automation (ETFA), 2009, 1 - 8. https://doi.org/10.1109/ETFA.2009.5347021
[6] Chang, K.-H.: Product Design Modeling using CAD/CAE: The Computer Aided Engineering Design Series. The computer aided engineering design series: Elsevier, 2014.
[7] Cignoni, P.; Callieri, M.; Corsini, M.; Dellepiane, M.; Ganovelli, F.; Ranzuglia, G.: MeshLab: an Open-Source Mesh Processing Tool. In Proceedings of the 2008 Eurographics Italian Chapter Conference, 2008, 129-136. https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136
[8] Guerrero, L. V.; López, V. V.; Mejía, J. E.: Virtual Commissioning with Process Simulation (Tecnomatix). Computer-Aided Design and Applications, 11(S1), 2014, 11-19. https://doi.org/10.1080/16864360.2014.914400
[9] Haller, K.; Lee-St.John, A.; Sitharam, M.; Streinu, I.; White, N.: Body-and-cad geometric constraint systems. Computational Geometry, 45(8), 2012, 385–405. https://doi.org/10.1016/j.comgeo.2010.06.003

[10] Fält, J.; Halmsjö, J.: Emulation of a production cell: Developing a Virtual Commissioning model in a concurrent environment, Master Thesis. Chalmers University of Technology, Gothenburg, Sweden. 2016.
[11] Hoffmann, P.; Schumann, R.; Maksoud, T. M.A.; Premier, G. C.: Virtual commissioning of manufacturing systems: A review and approaches for simplification. In Proceedings of the 24th European Conference on Modelling and Simulation, 2010, 175–181.
[12] Hoffmann, P.; Schumann, R.; Maksoud, T. M.A.; Premier, G. C.: Research on simplified modelling strategy for virtual commissioning. In Proceedings of the 24th European Modeling and Simulation Symposium (EMSS), 2012, 294–302.
[13] IEC 62714-3: Engineering data exchange format for use in industrial automation systems engineering -Automation Markup Language - Part3: Geometry and kinematics. 2016.
[14] Kiefer, J.; Ollinger, L.; Bergert, M.: Virtuelle Inbetriebnahme – Standardisierte Verhaltensmodellierung mechatronischer Betriebsmittel im automobilen Karosserierohbau. atp Edition - Automatisierungstechnische Praxis, 2009 (7), 2009, 40-46. https://doi.org/10.17560/atp.v51i07.92
[15] Kiefer, J.; Bergert, M.; Rossdeutscher, M.: Mechatronic Objects in Production Engineering. atp Edition - Automatisierungstechnische Praxis, 2010 (12), 2010, 36–45.
[16] Kim, J.; Kim, K.; Lee, J.; Jeong, J.: Generation of assembly models from kinematic constraints. The International Journal of Advanced Manufacturing Technology, 26(1-2), 2005, 131–137. https://doi.org/10.1007/s00170-004-2231-3
[17] Lindworsky, A.: Teilautomatische Generierung von Simulationsmodellen für den entwicklungsbegleitenden Steuerungstest. Ph.D. Thesis, Technical University of Munich, Munich. 2011.
[18] Lombard, M.: SolidWorks 2013 Bible: The Comprehensive Tutorial Resource. Indianapolis, Wiley. 2013.
[19] Mitra, N. J.; Yang, Y.-L.; Yan, D.-M.; Li, W.; Agrawala, M.: Illustrating how mechanical assemblies work. Communications of the ACM, 56(1), 2013, 106–114. https://doi.org/10.1145/2398356.2398379
[20] Neugebauer, R.; Schob, U.: Reducing the Model Generation Effort for the Virtual Commissioning of Control Programs. Production Engineering Research and Development, 5(5), 2011, 539–547. https://doi.org/10.1007/s11740-011-0317-y
[21] Norton, R. L.: Design of Machinery: An introduction to the synthesis and analysis of mechanisms and machines (2nd): McGraw-Hill Education. 1999.
[22] Oppelt, M.; Barth, M.; Urbas, L.: The Role of Simulation within the Life-Cycle of a Process Plant: Results of a global online survey, 2015. https://doi.org/10.13140/2.1.2620.7523
[23] MathWorks.: SimScape: Mates and Joints. 2017. https://de.mathworks.com/help/physmod/smlink/ref/mates-and-joints.html?requestedDomain=www.mathworks.com.
[24] Seidel, S.; Donath, U.; Haufe, J.: Towards an integrated simulation and virtual commissioning environment for controls of material handling systems. In Proceedings of the 2012 Winter Simulation Conference, 2012, 1-12. https://doi.org/10.1109/WSC.2012.6465081
[25] Spitzweg, M.: Methode und Konzept für den Einsatz eines physikalischen Modells in der Entwicklung von Produktionsanlagen. Ph.D. Thesis, Technical University of Munich, Munich. 2009.
[26] Strahilov, A.; Mrkonjic, M.; Kiefer, J.: Development of 3D CAD simulation models for virtual commissioning. In Proceedings of the 2012 Tools and Methods of Competitive Engineering (TMCE), 2012, 1281-1288.
[27] Strahilov, A.; Damrath, F.: Simulation of the behavior of pneumatic drives for virtual commissioning of automated assembly systems. Robotics and Computer-Integrated Manufacturing, 36, 2015, 101–108. https://doi.org/10.1016/j.rcim.2015.01.001

[28] Thongnuch, S.; Fay, A.: A Practical Simulation Model Generation for Virtual Commissioning. In Proceedings of the 2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), 2017, 1077-1082. https://doi.org/10.1109/AIM.2017.8014162
[29] Thongnuch, S.; Fay, A.; Drath, R.: Semi-automatic generation of a virtual representation of a production cell: combining 3D CAD and VDI-2860 behavior models by means of AutomationML. at - Automatisierungstechnik, 66(5), 2018, 372-384. https://doi.org/10.1515/auto-2017-0108
[30] VDI-Guideline. VDI 3693 – 1: Virtual commissioning – Part 1: Model types and glossary. (VDI). Berlin: Beuth Verlag GmbH. 2016.
[31] Weber, K. H.: Inbetriebnahme verfahrenstechnischer Anlagen: Praxishandbuch mit Checklisten und Beispielen. VDI-Buch. Berlin: Springer Vieweg. 2015.
[32] Westkämper, E.; Baudisch, T.; Schlögl, W.; Frank, G.: Automatic Model Generation for Virtual Commissioning of Specialized Production Machines. Softwaretechnik-Trends, 32(2), 2012, 82-83. https://doi.org/10.1007/BF03323491
[33] Weyrich, M.; Steden, F.: Automated Configuration of a Machine Simulation Based on a Modular Approach. In Proceedings of the 23rd CIRP Design Conference, 2013, 603–612.
[34] Whitney, D. E.: Mechanical assemblies: Their design, manufacture, and role in product development. Oxford series on advanced manufacturing. Oxford University Press. 2004.
[35] Xu, W.; Wang, J.; Yin, K.; Zhou, K.; Van de Panne, M.; Chen, F.; Guo, B.: Joint-aware manipulation of deformable models. ACM Transactions on Graphics, 28(3), 2009. https://doi.org/10.1145/1531326.1531341
[36] Zäh, M. F.; Wünsch, G.; Hensel, T.; Lindworsky, A.: Nutzen der virtuellen Inbetriebnahme: Ein Experiment. ZWF Zeitschrift Für Wirtschaftlichen Fabrikbetrieb, 101(10), 2006, 595–599. https://doi.org/10.3139/104.101070