Using a Formal High-level Language to Instruct Manikins to Assemble Cables

Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 23 (2014) 29 – 34

www.elsevier.com/locate/procedia

Conference on Assembly Technologies and Systems

Using a formal high-level language to instruct manikins to assemble cables

Peter Mårdberg[a]*, Johan S. Carlson[a], Robert Bohlin[a], Niclas Delfs[a], Stefan Gustafsson[a], Lars Hanson[b,c]

[a]*Fraunhofer-Chalmers Centre, Chalmers Science Park, SE-412 88 Göteborg, Sweden*
[b]*Second Product and Production Development, Chalmers University of Technology, SE-412 96 and Göteborg, Sweden*
Industrial Development, Scania CV, SE-151 87 Södertälje, Sweden

* Peter Mårdberg. Tel.: +46-31-7724000; fax: +46-31-7724260. *E-mail address:* peter.mardberg@fcc.chalmers.se

Abstract

In this paper, a formal high-level language is used to generate simulations where a manikin assembles flexible cables. The language generates assembly instructions for the manikin, which automatically performs the corresponding assembly motion with as good ergonomic as possible. Due to weight, stiffness and narrow regions, it may be difficult to perform an assembly of the cable. Our approach allows us to verify that it may be performed in an ergonomically sound way. The generated instructions are formally verified to ensure that assembly order is held and to prevent erroneous assembly states. The simulations have been made on industrial test cases.

Selection and peer-review under responsibility of the International Scientific Committee of 5th CATS 2014 in the person of the Conference Chair Prof. Dr. Matthias Putz matthias.putz@iwu.fraunhofer.de

Keywords: Assembly; Digtal human modeling; Formal methods;

1. Introduction

Today, an electronic device is a commonly assembled part in the manufacturing industry. Usually, these devices needs to be further connected to other devices, to for instance get power supply and information from actuators. Thus, an assembled product may include several operations where cables need to be routed.

These cables have to be manually routed through the assembly operation. Hence, all the forces and torques that occurs due to weight, stiffness and shearing from the cable, also needs to be handled throughout the routing. Moreover, due to compact constructions, the routing may occur in narrow and inaccessible regions.

In order to reach a high quality production, it is important to ensure that it is both possible to correctly assemble the cables, and that it is possible to make in an ergonomic sound way.

A Digital Human Modeling software is an important tool in virtual manufacturing, which allows simulation of manual assembly work long before any physical product and work place has been built (Laring, 2004). Thus it is possible to solve design issues, troublesome assembly sequences and logistic bottlenecks early in the conceptual development. This reduces the cost of late design changes, increases the production quality and decreases the ramp-up time of a manufacturing process (Falck, et al., 2010).

A step towards integrating a manikin with flexible cables is presented by (Wegner, et al., 2013), and in the work of (Delfs, et al., 2013), it is shown how it is possible to integrate the cables in the kinematical skeleton of the manikin.

However, far from all assembly operations are simulated and evaluated even if all the necessary data is available. One reason for this is the time consuming and tedious work that is required to setup and to define all the realistic motions needed by a manikin to perform a simulation (Lämkull, 2009; Raschke, et al., 2005). In each assembly simulation, the user has to position the manikin at the workstation, adjust the manikin into the desired posture and select the correct grip. Moreover, to make the simulation relevant, the user must

Selection and peer-review under responsibility of the International Scientific Committee of 5th CATS 2014 in the person of the Conference Chair Prof. Dr. Matthias Putz matthias.putz@iwu.fraunhofer.de
doi:10.1016/j.procir.2014.10.074

ensure that the manikin maintains balance during the simulation and that it avoids collision with objects in the environment.

In order to create a realistic simulation an assembly motion of a cable, the torques and forces from the cable needs to be included in the simulation.

Hence, even a small assembly case may be time consuming to simulate, and it is well motivated to find an easier way to construct assembly simulations.

A novel approach that uses a formal high-level language to instruct automated manikins to reduce the time needed to construct assembly simulations has been introduced in (Mårdberg, et al., 2013). This paper shows how this language also may be used to easily construct assembly simulations with flexible cables. The user define were the cable should be connected and then instruct the manikin with high-level instructions to perform the assembly.

This is possible by including the cables into the assembly model, as the instruction language, the manikin and all the objects used in the assembly are composed into the same discrete model (Mårdberg, et al., 2013). It is also formally verified, to ensure that it is valid.

The grammatical structure of the language divides the instructions into a hierarchical tree, where the lowest levels in the tree contains the basic instructions for maneuvering the manikin, such as Move, Position and Grasp, and the higher levels contain more abstract instructions such as Get and Assemble. Thus, a high-level instruction such as Assemble defines sequences of other instructions, whereas a low-level instruction such as Grasp corresponds to a direct instruction to the manikin.

This approach has been implemented in the manikin simulation software Intelligently Moving Manikins (IMMA) (Hanson, et al., 2011), and it has been tested on relevant assembly cases from the automotive industry. It is shown that it is possible to efficiently construct simulations where flexible cables are assembled.

The main contributions of this paper are that we based on the work proposed in (Delfs, et al., 2013) expand the assembly model and instruction language proposed in (Mårdberg, et al., 2013) to reduce the time needed to perform an assembly simulation with a flexible cable.

This paper is organized as follows. Section 2 covers the requirements of the presented approach, whereas Section 3 covers our modeling approach. Section 4 shows some case studies followed by discussion and future work in Section 5. Concluding remarks are found in Section 6.

2. Assembly model requirements

In this work it is shown that it is possible to automate the process of generating assembly simulations with flexible cables. However, this approach requires a model that contains an automated manikin, a formal instruction language and all the cables and geometries used in the assembly.

2.1. Automated manikin

A manikin can be said to be automated if it is able to automatically perform an assembly operation. Thus, if the manikin is instructed to grasp an object, then it should be able to automatically reposition itself and grasp the object without any further help from the user. Moreover, it is not sufficient for the manikin to just automatically perform the assembly operation; it also needs to maintain balance during the operation. The balance has to consider the body parts and the objects being carried as well as exterior forces and torques from the environment. Furthermore, it also needs to automatically avoid collision with the objects in the assembly station (Bohlin, et al., 2012; Delfs, et al., 2013).

2.2. Formal Language Definition

The set of available instructions that the manikin may perform during a simulation depends on the current state of the manikin and on the state of the objects in the assembly station. For instance, if the manikin grasps an object with both hands, it is then seen as impossible for the manikin to grasp another object.

The properties of objects, such as grasping points and mating points also help to define the set of available instructions for the manikin. Each low-level keyword must have a corresponding action in the simulation. A *Grasp* instruction may only be used if there is an object that is available for the manikin to grasp.

It is also possible to consider the order in which the different parts should be assembled when constructing instruction sequences for the manikin. Thus, it is possible to prevent a manikin from performing an assembly instruction unless all the preconditions to that instruction are fulfilled.

2.3. Flexible Cable

The flexible cables used in the assembly are modeled as cosserat rods (Hermansson, et al., 2013). Manipulation handles, which may for instance be defined as clips clamped onto the cable, work as constrains for the cables, see Figure 1. Thus, properties of the material in the cable are represented by introducing reaction torques and forces into the handles. A cable is said to be placed in desired configuration when all constrains consisting of positions and orientation in all handles are in mechanical equilibrium (Hermansson, et al., 2013).

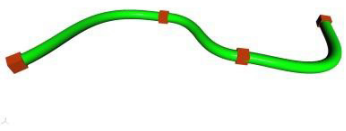

Figure 1: A cable with four manipulation handles.

2.4. Assembly model

All objects in the assembly station and the manikin are composed into the same discrete model. Moreover, the transitions in the model are defined by the instruction language. Thus, since they utilize the same discrete computational model, it is possible to apply formal methods to verify that the assembly operations are correctly performed, that the model specification is not violated and that no deadlocks occur.

3. Modeling a Cable Assembly

This section is divided into four subsections. The first shows how we define the grammar for the high and low-level language instructions. This is followed by two sections in which we show how to construct a model of the assembly operation and how the mounting instruction of a cable is defined. Then a brief section on how we use formal methods to validate our model.

3.1. Language instruction grammar

The instruction language is a context free language where the keywords are divided into high and low-level instructions. The high-level instructions define sequences of other instructions, whereas low-level instructions correspond to direct instructions to the manikin. Since the language is context free, there exists a non-deterministic pushdown automaton (Hopcroft, et al., 2007) that defines a state machine controlling the transitions in the underlying Discrete Event System (DES) (see Section 3.2). Each instruction corresponds to a non-empty set of transitions in the composed model. In this way it is possible to integrate the language into the DES model.

A grammar is defined to formally structure the instruction levels into a hierarchical tree (Hopcroft, et al., 2007; Aho, et al., 2007). The grammar furthermore defines how the instruction sentences are generated, and hence ensures that all the corresponding arguments are set (Aho, et al., 2007). For instance, consider the instruction $Grip \rightarrow RightHand \rightarrow Object_i \rightarrow TargetPoint_j$. Here $Grip$ is the keyword and $RightHand$, $Object_i$ and $TargetPoint_j$ are the corresponding arguments.

3.2. Modeling the assembly operation

A sequence of assembly instructions is said to be valid if it does not contain any contradictory instructions that violate the model specification. For instance, to grasp an object that currently may not be grasped or to give identical instructions in consecutive order, such as $Release \rightarrow RightHand$ directly followed by $Release \rightarrow RightHand$, are contradictory instructions.

To be able to prove that the generated assembly instructions are formally correct, the manikin has to be modeled into the same discrete event system as all the objects in the assembly operation. The model must also include all the properties of the objects that may directly or indirectly be used in the assembly.

All objects are modeled by an Extended Finite Automaton (EFA). Using the notation in (Bengtsson, et al., 2012), an EFA E might be defined as $E = \langle Q \times V, \Sigma, G, A, \rightarrow, (q_0, v_0) \rangle$, where Q denotes the set of state locations, V is finite set of variables and Σ the non-empty finite set of events. The guard predicates are denoted by G whereas A denotes the action functions that update $v \in V$. The transition relation and the starting state of the automation are denoted $\rightarrow \subseteq Q \times \Sigma \times G \times A \times Q$ and (q_0, v_0), respectively. A transition in E may only occur if the corresponding guard predicate is fulfilled. If the transition occurs, then the corresponding action updates the variable set V (Bengtsson, et al., 2012; Miremadi, et al., 2011).

All objects are modeled separately as EFAs and are composed into the same model using another EFA, denoted E_{All}, defined as the parallel synchronization of all EFAs in the scene: $E_{All} = E_0 \parallel E_1 \parallel \cdots$ (Sköldstam & Åkesson, 2007; Miremadi, et al., 2008). The composed automaton is automatically constructed from the objects in the scene. Thus, when a manikin is included in the scene, the corresponding EFAs are added to E_{All}. Moreover, when a user defines a grip point on an object, an EFA for that grip point is also added to E_{All}.

Each EFA may contain actions $a \in A$ and predicates that define illegal states, such as a grip point being used by both hands at the same time. These predicates are used to create guards that prevent these states from occurring. The guards in the EFAs form the model specification for the composed automata E_{All}, and are used to automatically define a set of guard predicates for E_{All} to prevent transitions which violate the model specification. Once all the guards have been added to E_{All}, the set of allowed transitions corresponds to the set of instructions that the user may perform when constructing assembly sequences. Furthermore, the user may add constraints, such as assembly order, into E_{All}, which then also are included in the set of allowed transitions.

The following example shows how a grip instruction may be modeled by a pair of two-state automata, see Figure 2. Let $Q_{rh} = \{rh_0 \stackrel{\text{def}}{=} Right\ Hand\ Free, rh_1 \stackrel{\text{def}}{=} Right\ Hand\ Used\}$ and $Q_{gp} = \{gp_0 \stackrel{\text{def}}{=} Grip\ Point\ Free,\ gp_1 \stackrel{\text{def}}{=} Grip\ Point\ Used\}$ define the states in each automation, and let $\Sigma_{rh} = \{\sigma_2 \stackrel{\text{def}}{=} Use\ Right\ Hand, \sigma_3 \stackrel{\text{def}}{=} Free\ Right\ Hand\}$ define the events. Moreover, let v be defined as a Boolean string of length 2 as $v \in \mathbb{B}^2$, and let a guard be defined as Boolean function as $G(v) \rightarrow \mathbb{B}$. The Boolean string v and the guard $G(v)$ are shared by both automata. The transitions of these automata define the set of instructions that are available for the user. In this example, it is possible for the user to form a high-level instruction that allows the user to use the grip point as support point for the right hand, expressed in pseudo code as $Get \rightarrow Object_i \rightarrow GripPoint_j \rightarrow$ Support. Based on the grammatical structure, all the low-level instructions needed for the manikin to perform the operation are automatically generated. For instance, the manikin needs to grasp the object in order to take support at object, and a low level-instruction is created as, $Grip \rightarrow RightHand \rightarrow Object_i \rightarrow$

$GripPoint_j$. When a low-level instructions is created its corresponding events are called in the underlying automata. Thus, in this case, $Grip \rightarrow RightHand \rightarrow Object_i \rightarrow GripPoint_j$. will execute the transitions $\{\sigma_0, \sigma_2\}$. If the corresponding guard, $G(v) = \{\bar{v}_0 \wedge \bar{v}_1\}$, evaluates to true then v_0 and $v_1$are updated to true, and state transitions are made, and the low-level instruction is generated. Thus, the shared guards G and variables V may be used to control the transitions in the composed automaton. If all low-level instructions are generated, then they high-level instruction, in this case $Get \rightarrow Object_i \rightarrow GripPoint_j \rightarrow$ Support, may be executed.

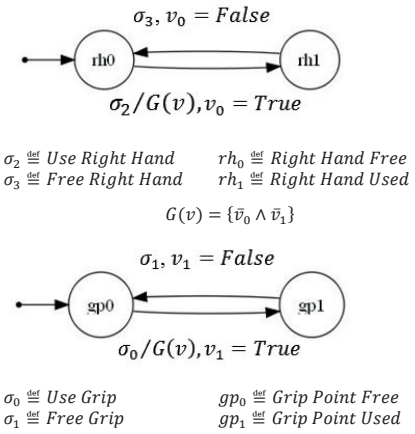

Figure 2: A pair of two-state automata. The upper models the grip of the right hand whereas the lower models the state of the grip.

3.3. Grasping and releasing a cable

Manipulation handles are mounted at fixed points along the cable. Two handles are needed in the start and end of a cable. Movements of handles are represented by transformations in $R^3 \times SE(3)$ (Hermansson, et al., 2013). Thus, when moved, torques and forces are created due to reaction from strain and shear in the cable and due to its own weight. These repelling torques and forces are contained in each handle and are added to the kinematical model of the manikin when they are grasped and removed when released.

The assembly of a clip is represented by following the clip along a short path in the clip direction, see Figure 3. Moreover, each assemble clip also contains a force that represent the force needed by the manikin in order to push the clip in place. It is automatically added to the kinematical model when a push command is executed.

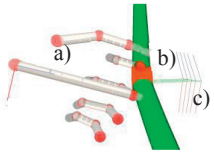

Figure 3: a) The grasp used to move and mount the clip. b) The path where the cable is moved into mounting position. c) The path to where the clip is mounted.

3.4. Formal Methods

Since the assembly information, the automated manikin, the instruction language and all the objects used in the assembly are contained in the model. It is important to verify that it is correct.

Formal verification is a methodology to prove if a set of properties hold for a model (Voronov & Åkesson, 2009), and it is used to verify that no language instruction violates the model E_{All} and to ensure that there are no contradictions in the model specification. By modelling properties of the manikin and the objects into E_{All}, it is possible to also include them in the verification synthesis. For instance, it is possible to verify if a set of states are reachable from the initial state, or that there exists a way to return to the initial state. In this way, the manikin may be prevented from reaching a deadlock state or from violate any properties in the model.

4. Test Case

The presented test cases are based cable assemblies in a truck cabin, and we show two assembly methods with different automation levels.

The test cases are based cable assemblies in a truck cabin. The first test case shows how it is possible to construct a simulation by individually defining the instructions needed to connect each clip of the cable. In the second test case, it is shown how a sequence of push instructions may automatically be generated. Based on the grip points, the instructions needed for mounting the clips are automatically generated.

The flexible cables are constructed in IPS (Industrial Path Solutions, 2012). However, the collision model is not used, and the cables are allowed to intersect with other geometric objects. Also, the generated paths used to move the clips and to define the path to which the clips are pushed are not guaranteed to be collision free.

For each clip to be mounted a grip point has been defined. The force required to mount a clip is set to 35 [N] and is based on an industrial case (unpublished data). Each clip is set to be is pushed 20[mm]. Thus, when a push instruction is used on a grip point, the clip will be moved 20[mm] along the local z-axis of the grip point.

The manikins are instructed using a graphical language, where the user gradually builds up an instruction by selecting different options, see Figure 4. The set of available options depend on the current instruction and on the objects in the scene. The user creates operations that may consist of one or several instructions. An operation must be verified in order to be executed. In the execution step, a list with all the necessary low-level instructions needed for the manikin to perform the operation is automatically generated. The low-level instructions are then sequentially executed.

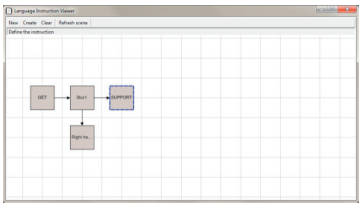

Figure 4: The graphical interface used to instruct the manikins in IMMA.

4.1. Test case A

In this case the cable is assembled by mounting two clips in the chassis, see Figure 5. The grip points have been defined for mounting the clips and to allow support for the manikin. Three instructions are defined and composed into an operation. The clips shown in Figure 5 a) and b) are mounted using two push instructions. One instruction is defined to allow the manikin to use a grip point, see Figure 5 c), for support when performing the assembly.

Figure 6, shows two frames of the assembly where the manikin leans into the chassis, takes support, and mounts the clips. Table 1 shows pseudo code of the low-level instructions that automatically have been generated too perform the assembly operation.

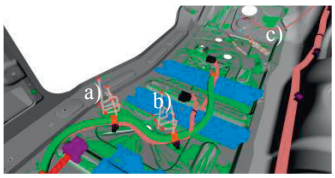

Figure 5: a) and b) shows the clips to be mounted, and c) the support point. CAD models courtesy of Scania.

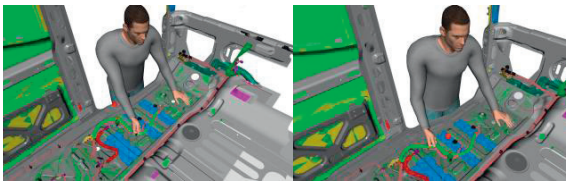

Figure 6: Two frames showing the assembly of the two cable clips. CAD models courtesy of Scania.

Table 1: Pseudo code of the low-level instructions used to perform the assembly of two cable clips in Figure 6.

Get Chassis left hand grip (Figure 4c)	**Get Clip2 right hand grip (Figure 4a)**
Set left hand as support	**Create the path needed to push the clip**
Get Clip1 right hand grip (Figure 4b)	**Push the clip along the generated path**
Create the path needed to push the clip	**Release Clip2**
Push the clip along the generated path	**Release Chassis**
Release Clip1	

4.2. Test case B

A cable consisting of four clips is to be mounted in the chassis. On each clip to be mounted, a grip point is defined and positioned to in the direction of the intended push operation, see Figure 7. One push instruction is defined, and the cable is selected as to object to be pushed. When a cable is selected as push object a push instruction is created for each clip on the cable that has a grip point. Thus, in this case, four push instructions are generated. The order of the push instructions is defined of the order in which they are placed in the operation. The pseudo code of the low-level instructions that are automatically generated out of the four push instructions are shown in Table 2. Figure 8, shows four frames of the assembly.

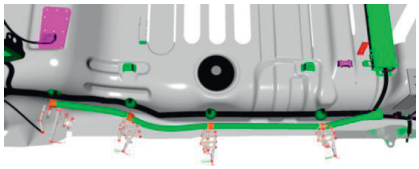

Figure 7: The flexible cable with the corresponding grip points. CAD models courtesy of Scania.

Figure 8: Four frames showing of the assembly of the cable in Figure 7. CAD models courtesy of Scania.

Table 2: Pseudo code of the low-level instructions used to perform the assembly of the four clips of the cable in Figure 7.

Get Clip1 right hand grip	**Get Clip3 right hand grip**
Create the path needed to push the clip	**Create the path needed to push the clip**
Push the clip along the generated path	**Push the clip along the generated path**
Release Clip2	**Release Clip3**
Get Clip2 right hand grip	**Get Clip4 right hand grip**
Create the path needed to push the clip	**Create the path needed to push the clip**
Push the clip along the generated path	**Push the clip along the generated path**
Release Clip2	**Release Clip4**

5. Discussion and Future Work

It is shown that with our approach it is possible to easily construct simulations with flexible cables. The forces and torques from the cable are included in the kinematical model of the manikin and are taken into consideration when the manikin tries to avoid collision and maintain the balance in an ergonomic sound way. Since the assembly operation and the manikin are merged into the same model, it is also possible to formally prove that the manikin correctly performs the assembly operation.

Two methods for constructing cable simulations are presented. In the second method (test case B), it is shown that it is possible further reduce the time needed to construct simulations. However, the current implementation only allows the same hand to be used in one mounting sequence. To allow the manikin to automatically use different hands to both the support and mounting clips will be covered in the future work.

There are no collision between the cables and objects in the environment. However, this is necessary in order to achieve a realistic simulation and will be considered in the future work.

Even if it is an important factor, the assembly time has not been considered in the simulations. However, the assembly time may be computed from a Predetermined Motion Time system in a post computational step (Mårdberg, et al., 2013).

In the test cases, the cable has been placed close to its final assembly position. The part where the cable has been brought from the material shelf and been straightened out into the chassis are omitted from the simulations. In order to simulate this, there is a need for a possibility to set a handle to be active or not. A handle is active when it is used by the manikin or has been mounted and inactive otherwise. An inactive handle follows the activated handles on the cable.

6. Conclusion

In this work we show how a high-level language is used to instruct an automated manikin to automatically perform assembly of a flexible cable. The assembly is made on test cases from the automotive industry, and it is shown it is possible to efficiently construct simulations with flexible cables. The instruction language is designed to use the automated functions of the IMMA manikin. This reduces the number of instructions needed to generate an assembly simulation with flexible cables. For instance, the torques and forces from the stiffness, shearing and weight of the cables will automatically be included into the kinematic model of the manikin when it assembles a cable.

The modeling approach is general and may be expanded to allow more properties in the objects, more complex assembly instructions and more manikins to be included into the model.

Moreover, with our modeling approach, the whole assembly operation may be included into the same model as the manikin. This way, it is possible to formally prove that the manikin correctly performs the assembly operation.

Acknowledgements

This work was carried out within The Swedish Foundation for strategic Research (SSF) ProViking II program and the Wingquist Laboratory VINN Excellence Centre, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA).

This work is a part of the Sustainable Production Initiative and the Production Area of Advance at Chalmers University of Technology.

References

Aho, A. V., Lam, M. S., Sethi, R. & Ullman, J. D., 2007. *Compilers : principles, techniques, & tools.* Boston: Pearson Addison-Wesley.

Bengtsson, K. o.a., 2012. Sequence Planning Using Multiple and Coordinated Sequences of Operations. *IEEE Transactions on Automation Science and Engineering,* 9(2), pp. 308 - 319.

Bohlin, R. o.a., 2012. *Automatic Creation of Virtual Manikin Motions Maximizing Comfort in Manual Assembly Processes.* Ann Arbor, Michigan, USA, u.n.

Delfs, N. o.a., 2013. Automatic Creation of Manikin Motions Affected by Cable Forces. *Submitted to International Journal of Human Factors Modelling and Simulation.*

Delfs, N. o.a., 2013. *Introducing Stability of Forces to the Automatic Creation of Digital Human Postures.* Ann Arbor, Michigan, USA, u.n.

Falck, A.-C., Örtengren, R. & Högberg, D., 2010. The Impact of Poor Assembly Ergonomics on Product Quality: A Cost–Benefit Analysis in Car Manufacturing. 20(1), p. 24–41.

Hanson, L., Högberg, D., Bohlin, R. & Carlson, J., 2011. *IMMA – Intelligently Moving Manikins – Project Status 2011.* Lyon, First International Symposium on Digital Human Modeling.

Hermansson, T., Bohlin, R., Carlson, J. & Söderberg, R., 2013. Automatic assembly path planning for wiring harness installations. *Journal of Manufacturing Systems,* 32(3), pp. 417-422.

Hopcroft, J. E., Motwani, R. & Ullman, J. D., 2007. *Introduction to automata theory, languages and computation.* Boston : Pearson Addison-Wesley, cop.

Industrial Path Solutions, 2012. [Online] Available at: www.industrialpathsolutions.com

Laring, J., 2004. *Ergonomic Workplace Design Analysis: Development of a practitioner's tool for enhanced productivity.* Göteborg: Chalmers University of Technology.

Lämkull, D., 2009. *Computer Manikins in Evaluation of Manual Assembly Tasks.* Göteborg: Chalmers University of Technology.

Miremadi, S., Åkesson, K. & Lennartson, B., 2008. *Extraction and Representation of a Supervisor Using Guards in Extended Finite Automata.* Göteborg, u.n.

Miremadi, S., Åkesson, K. & Lennartson, B., 2011. Symbolic Computation of Reduced Guards in Supervisory Control. *IEEE Transactions on Automation Science and Engineering,* 8(4), pp. 754-765.

Mårdberg, P. o.a., 2013. *Introducing a Formal High-Level Language for Instructing Automated Manikins.* Ann Arbor, Michigan, USA, u.n.

Mårdberg, P. o.a., 2013. Using a Formal High-Level Language and Automated Manikin to Automatically Generate Assembly Instructions. *Submitted to International Journal of Human Factors Modelling and Simulation.*

Raschke, U., Kuhlmann, H. & Hollick, M., 2005. *On the Design of a Task Based Human Simulation System.* Iowa City, Iowa, USA, SAE International.

Sköldstam, M. & Åkesson, K., 2007. *Modeling of Discrete Event Systems using Finite Automata With Variables.* New Orleans. LA, USA, Proceedings of the 46th IEEE Conference on Decision and Control,.

Wegner, D. o.a., 2013. *Simulation of Flexible Part Installation.* Ann Arbor, Michigan, USA, u.n.

Voronov, A. & Åkesson, K., 2009. *Verification of process operations using model checking.* Bangalore, India, August 22-2, 2009, u.n.