MDL, 2020, pp. 1–12

doi: 000000
MDL
Paper

PAPER

MDL+ A Manufacturing Description Language to Describe and Control Assembling Tasks in Industry 4.0

Mauricio-Andres Zamora-Hernandez,[1,*] Jose Andrez Chaves Ceciliano,[1] Alonso Villalobos Granados,[1] John Alejandro Castro Vargas,[2] Jose Garcia-Rodriguez[2] and Jorge Azorin-Lopez[2]

[1]Industrial engineering, University of Costa Rica, Ciudad Universitaria Rodrigo Facio, 2060, San José, Costa Rica and [2]Department of Computer Technology, University of Alicante, San Vicente del Raspeig, 03690, Alicante, Spain
[*]Corresponding author. mauricio.zamorahernandez@ucr.ac.cr
FOR PUBLISHER ONLY Received on Date Month Year; revised on Date Month Year; accepted on Date Month Year

Abstract

The assembly of products or components by operators in industries is a complex task with recurring problems. In these processes, operators often make errors that can lead to defective products. Therefore, they need to be inspected later to verify their correct assembly. The main problems are caused by several reasons including: high employee turnover, lack of experience in manufacturing specific products, or confusion in interpreting instructions to assemble similar components. In this paper, a novel structured language aimed to describe the required actions to manufacture a product in industrial assembly environments is presented. The main contribution is to provide a formal language to feed automatic systems that can verify through visual control whether the actions performed by the operator are carried out following the standard described by this language. In general, the use of this formal language will allow to minimize the negative impact of errors during assembly and to reduce waste in many forms in the industry.

Key words: Assembly representation, Automatic Inspection, Control language, Industry 4.0, Recommendation system

Introduction

This research work focuses on improving the quality and safety in the industries during the manual product assembly process by checking that the actions carried out by the operators are according to the defined quality standards. The aim is to achieve homogeneity in the assembly of final products, minimizing manufacturing problems, or waste of time and money due to reprocessing assemblies.

The control of manufacturing processes is a problem that has always been present in industry. Kanawaty [2008] defines that "the study or engineering of methods is the registry and systematic critical examination of the ways of carrying out activities, in order to make improvements". Studying the method allows us to analyze processes from their most basic elements, such as the sequence of necessary movements, to complete tasks. In this way, improvements can be made in production processes, determining changes in sequences or reducing unnecessary movements. Therefore, it is a crucial topic for research in Industrial Engineering, since it permits a correct production planning and provides an adequate analysis of operations. This becomes a way to establish more precise calculations in the production capacities for the industry. In addition, it promotes the improvement of the quality in the processes from a granular level, reducing wastes in the form of money, time, and materials.

Along with advances in technology, alternatives have been proposed to solve the problem of assembly control, for which the techniques of automatic inspection systems are commonly used. These approaches compare the assemblies against measurement standards. Due to the growing need for high-quality and personalized products, new requirements have been defined in quality control systems. Systems should learn to identify or process specific parts that are designed for particular solutions. These quality inspection processes can be implemented by making use of simple sensors such as those that measure weight, color or size [Fast-Berglund et al., 2013]. However, these inspections are limited to the capability of

the sensors available in the market, so they might not be as useful in many cases. Recently, Computer Vision (CV) is one of the most used technologies to validate the quality of the manufactured products by visual control mechanisms at workstations [Hedelind and Jackson, 2011].

Among the applications of computer vision in the manufacturing industry for quality control stand out: quality control (shapes, sizes, colors), collision detection [Wang et al., 2013], navigation [Hornung et al., 2010] and augmented reality [Makris et al., 2016]. However, this work proposes the use of CV beyond the measurement of characteristics by visually identify the actions executed by the operator and compare them with a specified standard. The standard is described using our proposed *Manufacturing Description Language*. This language defines the sequence of manual construction steps needed to succesfully build a product that meets the quality standard defined by the company, guiding the operator through the process.

There are already research proposals based on Artificial intelligence (AI). For example, Luo et al. [2019] used the image captioning technique for automatically generating an image description. In this case, it could be applied to describe what happens in assembly environments, considering verification of the necessary manufacturing steps. However, usually in manufacturing processes, more than a description with a simple label is required, a complete semantic description of the actions performed is demanded by the quality control department. For this reason, researchers such as Wang et al. [2018] have focused on captioning for video using Hierarchical Reinforcement Learning techniques to generate a more detailed description. Another important work has been developed by Krishna et al. [2017] using long short-term memory (LSTM) deep networks in a dense-captioning model for event detection. In both cases, descriptive narratives in natural language are used. In order to advance in this area, Yao et al. [2017], proposed to create narratives in different ways: template-based methods where structures are created, or search for generate narratives in fixed forms. In general, visual element-based approaches are being used to generate texts and language-based models using k-nearest neighbour retrieval models. However, these tehcniques have not a formal specification of the image descriptions. It is convenient to use grammatical systems to structure the instructions in such a way that they are able to express rules applicable to the industry. On the other hand, authors as Nguyen et al. [2019] proposed techniques to understand and imitate human actions without defining objectives or validations for the actions. Therefore, a grammar that allows the quality control systems to structure manufactuing instructions and enables adequate communication of the actions to be carried out without creating ambiguities between the parties involved is required.

To mitigate these deficiencies, Yang et al. [2015] proposed a system of convolutional neural networks which, through video analysis, constructs grammar trees of the observed actions. These are unrestricted general use grammars for a particular use, so it could generate unstructured actions for the strict verification of what is captured. Mancini et al. [2018] are working on object detection in specific domains of industry, but, without the definition of a grammar to describe the sequence of actions. Therefore, our proposal of designing an assembly specific-domain grammar in Industry 4.0 is a novel researh area.

The goal is to create a simple grammar that describes the daily activities in the manual assembly of a product. In order to increase its usability, the principles of the *Therbligs* theory have been applied Groover [2007], Universidad Politécnica deValencia [2018], Ferguson [2000] to allows to represent all the assembly sequences using micro-movement primitives. The proposed language is based on the analysis of the most basic movements and actions of the human hands while a product is being manufactured.

The language allows describing the actions of the operators in a manufacturing cell, supported by the grammar that will be detailed in the following sections. A new method of representing assembly instructions is proposed, which will be the base of a visual control system implemented for the quality control of the manual assembly process. Moreover, this paper extends the features of our previous work [Zamora-Hernández et al., 2021], including an extension to represent parallel tasks developed in a coordinate way by different operators or whith a single operator but using both hands. On the other hand, it introduces new quantative experimentation with different actions and evaluating the advantages of using the language with operators manufacturing evaluation tests.

The remaining of the paper is structured as follows: Section 2 presents the different forms of description of actions in the industry, then the proposed language and the general structure of its main elements, as well as particular syntax elements is presented in Section 3. In section 4, the general validation of the proposal is carried out using some examples that shows how different representative assembly tasks are developed. Finally, the conclusions and future works are presented.

Description of Actions in Industry

The use of languages is required to express, in a formal and standarized manner, the necessary steps to manufacture a product. For this purpose, it is needed to define particular semantics for the language. A literature review was carried out in order to find existing languages that could represent the necessary instructions for the manual assembly of products by the operators. However, the specific work is scarce. Among the most significant contributions, IBM proposed in the mid-80s a language for the product assembly process, but restricted to the use of robots without any interaction with the operators. Moreover, the robots were only CNC-type machines for repetitive work on fixed positions, without any kind of cognitive analysis [Nackman et al., 1986].

Some works used CV to identify actions that can be described by text, in some cases using Natural Language Processing. However, as semantics are not available, a standardized format is not generated, and that would complicate the process of analyzing actions oriented towards manufacturing.

One of the most advanced challenges in this area is to provide to a machine with the capabilities to understand the actions performed by a human (in this case, an operator) through CV, so that a machine performs exacly the same actions through an observation and learning process. One of the possibilites to provide it is by recognizing actions through task detection, which can output *captioning* in sentences using natural language, but without grammar knowledge because this form of language lacks of the semantics that allow to express ideas correctly [Nguyen et al., 2019].

Human-robot collaboration applications should be natural for people, so there can be a better understanding of the steps needed to be followed along a process. In this line of research, Starke et al. [2019] studied human factors generating

data from the position of the hands during the execution of actions on assemblies and tools at the level of grip pressure and movement of the skeletons. This could improve the definition of manufacturing actions and their optimization in the assembly processes.

Manufacturing Description Language

In this section we detail our proposal for the *Manufacturing Description Language* (MDL). This language allows factory managers to document the sequences of activities neceded for the development of a product and to ensure that the operator does it properly. This language was designed, not only as a complement to automated visual inspection research, but for general use in the description of manufacturing processes in a standarized way.

While designing the language, it was established that it would form a basic part of a global solution, which would use CV systems, to indicate the general operating settings automatically. For example, working area characteristics, tools to be used, parts and components needed for assembly, and their locations. However, for environments where these capabilities are not available, the language provides a *setup* section to set the parameters manually. In the latter, it is required that the location for each element on the workplace is standarized in order to eliminate the need to set the parameters manually each time that the process is going to be performed.

Another language requirement during the design stage was a simple and concise syntactic structure for the instructions. Thus, its use should only requires brief and simple training, since it can be used by people of different specialities and skills: quality and production managers, and the operators of the manufacturing cells themselves. Another relevant feature to highlight is that the system allows defining an assembly by its components, this generates an advantage since milestones can be established during construction for quality evaluation along the process. It also means flexibility, allowing to establish alternate assembly routes for specific products.

A basic set of tools was defined, but the system gives the possibility of extending the language with new tools and configurations. Among those available within the basic version of the language, it can be found:

- Clamp
- Gun Drill
- Screw Drill
- Ball Pein Hammer
- Claw Hammer
- Nut Driver
- Diagonal Pliers
- Lineman Pliers
- Locking Pliers
- Long Nose Pliers
- Ratchet
- Electric Screwdriver
- Phillips Screwdriver
- Slotted Screwdriver
- Socket
- Adjustable Wrench
- Allen Wrench
- Combination Wrench

Also, as a complement to the tools, some of them have accessories defined in the language. As well as parts and components of general use, among these are:

- Bolt
- Drill bit
- Gears
- Nut
- Screw
- Washer

A simple grammar is proposed using the micro-movements theory of *Therbligs* and the study of how companies document their processes. This grammar allows describing the daily activities in the manual assembly of products, taking into account the needs of people who work in this area and those who evaluate the quality of the product.

Instructions developed with the micro-movement primitives are used to represent entire assembly sequences. This also encourages its use by Industrial engineering professionals due to the advantages of using proven techniques. Also, since this language was proposed as the basis of a visual control system, some of its characteristics and structure were designed with this purpose. In consequence, the instructions generated using a formal grammar could be analyzed, interpreted and finally compared with the visual inputs of the control system.

Therefore, the main objective of the language is to evaluate whether an operator is assembling a component or product according to the specifications given by the instructions established. This evaluaton is carried out in two stages:

1. Expert stage: a process or quality engineer uses the language to describe the actions necessary for the assembly of a product.
2. Operator stage: the operator carries out the assembly tasks, while an artificial vision system records as input what is happening in the working area, processes it, and converts it into a textual description using video *captioning* techniques.

Subsequently, a comparison is made with the language description to determine if the operator followed the instructions correctly.

Besides, the system would be capable of determining the current step or action, to indicate to the operator the next steps. It reduces the possibility of errors when making the assembly and the waste of time generated by the operator trying to remember the next step (this happens to be a recurring issue when an operator is trained to assemble many different products). The full language proposal can be found in our public repository [1]

In the next subsection a general description of the sections that make up the language are decribed. First, the instructions that allow describing elements of the work environment are explained. After that, the instructions that allow describing the activities of the manual assembly process are detailed. Finally, the whole grammar of the language is listed.

Operation parameters

This section details how the initial execution settings are described within the language distinguising two different modes of operation: automatic and manual.

In the automatic mode, the system will be self-configured using visual information of the cell obtained through computer vision techniques that are able to describe the work environment. It is carried out by identifying the elements present on the working table and their locations.

In the second mode, an operator or process engineer is required to describe the working area using the elements that the language provides, which are detailed below. As mentioned before, this alternative requires the position of the elements on the workplace to be standarized.

To carry out the manual configuration, the following instructions are necesary (can be seen in detail in listing 8):

[1] `https://github.com/mazamorahdez/manufacturing_language`

1. *product*: establishes the ID code or name for the product to be registered in the assembly instructions.
2. *setup*: the initial positions where parts, components, and tools are placed within the working area. In addition, the location of the existing components (assemblies and sub-assemblies) is indicated; as well as the amount of components and sub-assemblies that will be generated during the manufacturing process. In this same section of the code, the dominant hand for the operator is defined, so that the system will configure the instructions according to the individual characteristics of each user.

As it can be seen, the quantities of each complement as bolts, screws and such items are defined in the setup. This allows the automatic inspection system to let the operator know whenever he is consuming the totality of the available elements.

Listing 1. Operation parameters

```
<set> := assembly <identifier> [to-create] [in <
    coordinate>:<offset>:<unit>];
<set> := hand <x-position>;
<set> := bin <part> [in <coordinate>:<offset>:<unit>];
<set> := tool <tool> [in <coordinate>:<offset>:<unit>];
<set> := accessory <accessory> [in <coordinate>:<offset
    >:<unit>];
```

In the language grammar the section “Setup” can be found. This section starts with the command *setup-begin* and proceeds with <sets >setup-end. Where the *sets* can take any of the following options:

- *assembly*: establishes the location of an assembly or sub-assembly that will be used during the manufacturing process. In case *to _create* is indicated, it means that the assembly will be created from the union of two or more assemblies during the execution. Hence, the corresponding blocks for these assemblies are defined in the system.
- *hand*: indicates to the system the operator’s dominant hand, thus adjusting the dominant and non-dominant hand instructions according to each individual.
- *bin*: defines the location of the container and its content, so that when the instructions are established the system knows where it should take from or where to place the assembly supplies such as components.
- *accessory*: some tools use accessories as drill bits for example. This indicates the location of these elements to the system, so that when shown in the instructions it can be verified if the correct accessories were used, by checking the location where the operator took the required element.

Assembly actions

In this subsection, the instructions that allow the description of the assembly sequence are introduce. Each instruction within this segment of code is called a *step*.

Each *step* instruction is classified into two possible types: those that define a single operation (which we will call *Individual steps*) and the instructions that define sets of steps that we call *Step blocks*. These are detailed in the following points.

Individual steps

At this point, the instructions that allow the definition of individual actions are detailed. Within this type of actions, there are three categories, which are defined below:

- *hand*: these operations are related to movements or actions performed with the hands, and are the most basic actions that can be performed by the operator along the process. See an example in listing 2.

Listing 2. Hand operations.

```
<step> := <hand-action>:(<part>|<tool>) with <handused
    >
[in assembly #<identifier>];
<step> := <hand-action>:<handused> [in <coordinate>:<
    offset>:<unit>];
<step> := <hand-action>:assembly #<identifier> with
  (assembly #<identifier> | <handused>);
<step> := move:assembly #<identifier>
with <handused> from <coordinate>:<offset>
to <coordinate>:<offset>:<unit>;
<handused> := hand | hand-nondominant | hand-any |
    hand-both
<hand-action> := put | hold | take | grip | release
| push | spin | turn | join | move
```

- *tool*: these instructions are related to the use of tools only or in conjunction with their respective accessories, as can be seen in the listing 3: there is an example of the actions defined for each one of the tools listed in listing 6.

Listing 3. Tool operations.

```
<step> := <substep> [in <coordinate>:<offset>:<unit>];
<substep> := <hammer-action>:<hammer> | <wrench-action
    >:<wrench>
| <screwdriver-action>:<screwdriver> | <pliers-action
    >:<pliers>
| <driller-action>:<driller> with <accessory> | <clamp
    -action>:clamp
| <ratchet-action>:ratchet with (socket|none)
| <screwdriver-action>:nut_driver
```

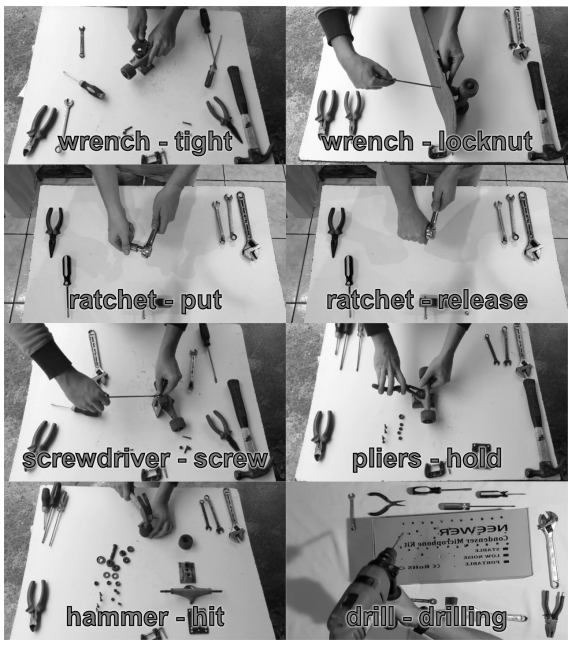


Fig. 1. Real images for training tools and actions

- *move*: these operations are intended to define offsets of the assemblies in the working area from a source location to a destination. This has been inroduced to simplify the combination of the take, move and put hand-actions in single instructions. See examples in listing 4.

Listing 4. Movements operations.

```
<step> := move:assembly #<identifier> with <handused>
from <coordinate>:<offset>
to <coordinate>:<offset>:<unit>;
```

Steps blocks

These blocks of steps are sequences made up with instructions of the *assembly-action* type required to complete a specific part of the global assembly. See examples in listing 5.

These are the available steps blocks types:

- *make-assembly*: this is one of the most relevant blocks of the language, since it allows to describe the concept of assembly (or sub-assembly), which defines milestones during the process. This permits the construction of complex elements from the union of other more basic ones. A relevant property of this block is that it defines named units, which can be invoked to be part of another *assembly*.
- *repetition*: there are sets of steps (including blocks), which are required to be executed several times. In these cases, the set of steps to be repeated and the number of times that they must be executed are defined.
- *any-order*: a basic premise of the language is that statements are executed in the order established unless it is stated otherwise. To do this, the *any-order* block is used, which tells the system that the instructions within this block can be executed in any order.
- *parallel*: it is used when it is required to execute instructions simultaneosly, either because there are more than one operator interacting in the working area, or to describe a coordinate action that involve the use of both hands.

The *parallel* block is useful when the operator performs more than one action at the same time. For example, when the operator holds a nut with a wrench and install a screw with the screwdriver by simultaneously turning both hands in different directions. This block of parallel actions was designed to consider when more than one operator are contributing to an assembly. It allows to specify the actions for the operators involved (although this case is less common than the previously detailed use).

Listing 5. Steps blocks

```
<make-assembly> := assembly-start #<identifier>:<steps>
      assembly-end;
<repetition> := repeat:<steps> until <digits> times;
<in-any-order> := any-order-begin <steps> any-order-end;
<parallel> := parallel-begin: <steps> parallel-end;
```

Parameterization

The key elements in the definition of the language with general purpose characteristics are those described below:

- *hand-action*: it defines all possible actions that can be performed with the hands, and represent the base of the steps required to build the assembly. Some of these were taken from the *Therbligs*. Others were incorporated according to real assemblies needs.
- *tool*: it defines the list of available tools to perform transformations or work on the elements in the assembly process. Basic and common families are created by studying manual assembly operators in several industries. However, the flexibility principle of the language guarantees the possibility to incorporate more tools, mainly due to the wide range of products and industries, and the constant development of new equipment. You can see the basic list of tools in listing 6.
- *tool-action*: these are the actions that can be performed on, or through, the tools. Since not all tools share the same range of actions, this element allows the association to be made. Like tools, this section can be updated to represent the actions of new tools coming to market. The system has the capability to extend the actions, but a basic set is defined that can be seen in listing 6.
- *substep*: the actions of a tool are matched with the respective family of tools.
- *complement*: the parts are the simplest and most common elements used in assemblies. These can be extended and modified like the rest of the elements. The system already incorporates a basic list that contains: screws, nuts, washers, among others.
- *part*: pieces that are used to build the product through actions that usually involve tools and complements in order to join them. These parts can be seen as the basic components of sub-assemblies that make up a more complex part when put together.

Listing 6. Tools and their actions

```
<tool> := <hammer> | <wrench> | <screwdriver>
| <pliers> | <driller> | ratchet| clamp
| nut_driver
<hammer> := hammer_ball_pein | hammer_claw
<wrench> := wrench_adjustable | wrench_allen |
      wrench_combination
<screwdriver> := screwdriver_electric |
screwdriver_phillips | screwdriver_slotted
<pliers> := pliers_diagonal | pliers_lineman |
      pliers_locking | pliers_long_nose
<driller> := drill_gun | drill_screw
<hammer-action> := nail | hammer_out | hit
<wrench-action> := pull | tight | locknut
<screwdriver-action> := screw
<pliers-action> := loosen | cut | hold | tighten
<driller-action> := drilling
<clamp-action> := loosen | tighten
<ratchet-action> := turn
```

- *accessory*: it is defined as a complement for a particular tool. Similar to *tool-actions*, they are specific to each tool, so their relationship must also be established. In this case, it is done with a *substep*. Some examples of accessories are: drill bits and nut hubs, among others.
- *coordinate*: it allows placing an ordered pair to locate the elements on the table, where the centroid of the work table is assumed as the point (0,0). It is based on the assumption that the visual control camera is located over the working bench, which makes it possible to interpret the workspace as a plane.
- *offset*: set the coordinate origin, define the offset, which is set as the length of each of the sides of a square, where the centroid of the square is the coordinate indicated.
- *unit*: the measurement units with which the offset and coordinates work.

The symbols and basic elements that are part of the parametrization of the language are shown in listing 7.

Listing 7. Language symbols

```
<offset> := <digits>
<unit> := mm | cm | mm
<coordinate> := <sign><digits>,<sign><digits>
<sign> := <void> | <positive> | <negative>
<void> := ''
<positive> := +
<negative> := -
<position> := <x-position> | <y-position> | <y-position
      ><-><x-position>
<x-position> := rigth | left
<y-position> := upper | lower
<identifier> := <char> | <char><word> | #bytes#
<word> := <alpha><word>
<alpha> := <char> | <digit> | <void>
<char> := a | b | ... | z | A | B |...| Z | _ | - | & | '
      ' | . | , | @
<digits> := <digit> | <digit><digits>
<digit> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Finally, the complete MDL+grammar is summarized in listing 8.

Listing 8. Language grammar

```
<S> := <assemble>
<assemble> := assemble-begin <product> <setup> start <
      steps> end assemble-end
<product> := <word>;
<setup> := setup-begin <sets> setup-end
<sets> := <set> | <set><sets>
<set> := assembly #<identifier> [to-create] [in <
      coordinate>:<offset>:<unit>];
<set> := hand <x-position>;
<set> := bin <part> [in <coordinate>:<offset>:<unit>];
<set> := tool <tool> [in <coordinate>:<offset>:<unit>];
<set> := accessory <accessory> [in <coordinate>:<offset
      >:<unit>];
<offset> := <digits>
<unit> := mm | cm | mm
<steps> := <step> | <step><steps>
<step> := <make-assembly>
<step> := <hand-action>:(<part>|<tool>) with <handused>
[in assembly #<identifier>];
<step> := <hand-action>:<handused> [in <coordinate>:<
      offset>:<unit>]
[in assembly #<identifier>];
<step> := <hand-action>:assembly #<identifier> with
 (assembly #<identifier> | <handused>);
<step> := move:assembly #<identifier> with <handused>
      from <coordinate>:<offset>
to <coordinate>:<offset>:<unit>;
<step> := <substep> [in <coordinate>:<offset>:<unit>]
[in assembly #<identifier>];
<step> := <repetition>
<step> := <any-order>
<step> := <parallel>
<substep> := <hammer-action>:<hammer> | <wrench-action>:<
      wrench>
| <screwdriver-action>:<screwdriver> | <pliers-action>:<
      pliers>
| <driller-action>:<driller> with <accessory> | <clamp-
      action>:clamp
| <ratchet-action>:ratchet with (socket|none) | <
      screwdriver-action>:nut_driver
<handused> := hand | hand-nondominant | hand-any | hand-
      both
<hand-action> := put | hold | take | grip | release
| push | spin | turn | join |move
<part> := bolt | gears | nut | screw | washer
<accessory> := drill_bit | socket | none
<make-assembly> := assembly-start #<identifier>:<steps>
      assembly-end;
<repetition> := repeat:<steps> until <digits> times;
<parallel> := parallel-begin: <steps> parallel-end;
<in-any-order> := any-order-begin <steps> any-order-end;
<tool> := <hammer> | <wrench> | <screwdriver>
| <pliers> | <driller> | ratchet| clamp
| nut_driver
<hammer> := hammer_ball_pein | hammer_claw
<wrench> := wrench_adjustable | wrench_allen |
      wrench_combination
<screwdriver> := screwdriver_electric
| screwdriver_phillips | screwdriver_slotted
<pliers> := pliers_diagonal | pliers_lineman |
      pliers_locking | pliers_long_nose
<driller> := drill_gun | drill_screw
<hammer-action> := nail | hammer_out | hit
<wrench-action> := pull | tight | locknut
<screwdriver-action> := screw
<pliers-action> := loosen | cut | hold | tighten
<driller-action> := drilling
<clamp-action> := loosen | tighten
<ratchet-action> := turn
<coordinate> := <sign><digits>,<sign><digits>
<sign> := <void> | <positive> | <negative>
<void> := ''
<positive> := +
<negative> := -
<position> := <x-position> | <y-position> | <y-position
      >-<x-position>
<x-position> := rigth | left
<y-position> := upper | lower
<identifier> := <char> | <char><word> | #bytes#
<word> := <alpha><word>
<alpha> := <char> | <digit> | <void>
<char> := a | b | ... | z | A | B |...| Z | _ | - | & | '
      ' | . | , | @
<digits> := <digit> | <digit><digits>
<digit> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Experimentation

The validation of the MDL+ is twofold: firstly, validating the efficiency and applicability of the language in the manufacturing cells. Second, validating the capacity of the MDL+ to describe actions.

Language validation

An experiment was designed oo carry out the language validation with a group of companies, whose production is developed in manufacturing cells. Each company provided at least one representative. This experimented worker participated in an interview that aimed to identify how to represent the instructions during the assembly process, and the tools that they normally use to perform these tasks.

For the design of the experiment, several phases were established aimed to specify a language that was conceptually equivalent to the regular instructions used by each one of the factories. The first phase of the experiment is detailed below:

1. Identify examples of production sequences used by the company to generate the step list required for each of the assemblies. During the research, the different ways of documenting processes and product assemblies of each company were collected. After that, an equivalent codification of the instructions that describe the process was generated using the *Manufacturing Description Language*.
2. In order to evaluate the equivalence between the original format used by the involved factories and the description of the process made using the *Manufacturing Description Language*, a comparison was developed and graded by each of the interviewed workers. Together with the collaborators selected, the new version of the instructions of their respective process was reviewed to determine its level of accuracy. It was possible to determine that aproximately 97 % of the instructions were succesfully converted, meanwhile, the remaining 3 % corresponded to very particular elements of each industry or process. To solve this point, a particular extension of the language was generated in order to complete the sequence. It is important to indicate that this was a consideration that was taken as a requirement when designing the language: its flexibility, offering an oportunity for the user to incorporate new tools and actions, meeting better the needs of each particular process.
3. To conclude this phase, a test was made by taking a group of operators and giving them a brief introduction on the use of language. Then, they were given a set of instructions to be executed on the workstations according to the new language.

The last stage in the validation was designed to determine if factors such as the learning curve in the assembly of products and previous knowledge of process documentation standards affect the results. In order to do so, a group of Industrial Engineering students volunteers from the University of Costa Rica carry out the validation. Since the Industrial Engineering degree is offered in three locations. The experiment was replicated under the same conditions for each of them. As requirements for this experiment, they had to be students of the first three semesters without experience in manufacturing companies. The number of men and women was balanced. Specfically, there was a participation of 24 to 28 students per

location, 12 to 13 for each gender, having a total of 8 to 9 students per semester (see Table 1).

Table 1. Distribution of experiment participants

Location	*Semester*	*Women*	*Men*
1	1	4	5
	2	4	4
	3	5	4
	Total	**13**	**13**
2	1	5	4
	2	5	5
	3	5	4
	Total	**15**	**13**
3	1	4	4
	2	4	4
	3	4	4
	Total	**12**	**12**

The assembly process of a skateboard was selected because its simplicity but it requires handling different tools and complements. The whole process was described in two different manners: first, by using traditional product description techniques; and second, by using the language proposed in this research.

The participants were randomly divided into two groups to alternate the execution of the sequences. Team 1 first assembled the skateboard using traditional techniques. They were derived from those acquired in the previous validation stages (involving representatives from different companies), and later on, assembled the product using the proposed language. Team 2 executed both alternatives in the opposite order, to mitigate the order effect.

Table 2. Run time of experiments

Experiment	**Loc.**	**Team 1**		**Team 2**	
		Convent.	*Proposal*	*Convent.*	*Proposal*
Experiment 1	1	00:04:59	00:03:56	00:05:11	00:04:30
Experiment 2		00:10:35	00:09:45	00:08:27	00:06:59
Experiment 3		00:07:06	00:05:53	00:09:14	00:07:55
Total		**00:22:40**	**00:19:34**	**00:22:52**	**00:19:24**
Experiment 1	2	00:04:34	00:04:01	00:04:36	00:03:59
Experiment 2		00:08:32	00:07:35	00:09:06	00:07:44
Experiment 3		00:08:21	00:07:35	00:08:46	00:08:02
Total		**00:21:27**	**00:19:11**	**00:22:28**	**00:19:45**
Experiment 1	3	00:04:44	00:03:41	00:04:19	00:03:18
Experiment 2		00:08:35	00:07:32	00:09:30	00:08:42
Experiment 3		00:09:36	00:08:46	00:08:26	00:07:33
Total		**00:22:55**	**00:19:59**	**00:22:15**	**00:19:33**

For the execution of the experiment, the following steps were followed:

1. All participants were trained to ensure that there were no variations in the understanding of the language, minimizing the variation caused by this factor.
2. The experiment was carried out in two consecutive classrooms with the same levels of environmental conditions (temperature, humidity, noise, lighting, etc.).
3. The individuals participating were separated so that they could not get more information of the assembly before it was their turn to proceed.

The experiments produced the results that can be observed in Table 2. In each experiment, on both teams, there was a reduction in execution time when using MDL + with respect to the traditional description of actions. Doing a comparative analysis of the results in percentage terms there is an average reduction of 12,74 % showing that MDL+ was easier to understand and apply.

Table 3. Percentage reduction in execution time

Experiment	**Loc.**	**Team 1**		**Team 2**	
		Time red.	*% red.*	*Time red.*	*% red.*
Experiment 1	1	00:01:03	21,07%	00:00:41	13,18%
Experiment 2		00:00:50	7,87%	00:01:28	17,36%
Experiment 3		00:01:13	17,14%	00:01:19	14,26%
Total		**00:03:06**	**13,68%**	**00:03:28**	**15,16%**
Experiment 1	2	00:00:33	12,04%	00:00:37	13,41%
Experiment 2		00:00:57	11,13%	00:01:22	15,02%
Experiment 3		00:00:46	9,18%	00:00:44	8,37%
Total		**00:02:16**	**10,57%**	**00:02:43**	**12,09%**
Experiment 1	3	00:01:03	22,18%	00:01:01	23,55%
Experiment 2		00:01:03	12,23%	00:00:48	8,42%
Experiment 3		00:00:50	8,68%	00:00:53	10,47%
Total		**00:02:56**	**12,80%**	**00:02:42**	**12,13%**

These results has been considered promsing in this phase of the experimentation and it is able to validate the improvements with respect to traditional techniques. The participants also offered feedback that will be used for improvements in future versions.

Description validation

In the second, and last phase of the validation experiment, the language was used to describe an entire assembly process. The research team recorded several videos and generated the complete code of the instructions to reproduce the assembly process .

The team carried out the following steps in order to generate these datasets:

1. The time duration should not to exceed five minutes, including at least three different actions and two different tools.
2. The playback speed of the video was reduced in a 50 %, in order to guarantee a better understanding of each action and movement.
3. For each action, the playback was paused to write the action displayed in the code. If required, object´s coordinates were written down. Distances were measured converting the distance from pixels to centimeters.
4. To reduce the size of the generated example, repetitive sequences were simplified by reducing the sequences that appeared several times, although in the real practice the step-block *repetition* can be used for such cases.

Basic rules for specifying assemblies

In order to make it clearer using a human-like language, we recommend employ a kind of pseudocode (similar instructions

than the ones used in MDL+ language but in natural language format) to be able to describe the general actions and then proceed with the language translation.

To explain how to express the assembly of a product in this pseudocode and their proper conversion in MDL+, sample code segments taken from the examples are explained in the Listings 9, 10 and 11. Additionally, the complementary images that illustrate each Listing are presented. They can be seen in Figures 2, 3 and 4.

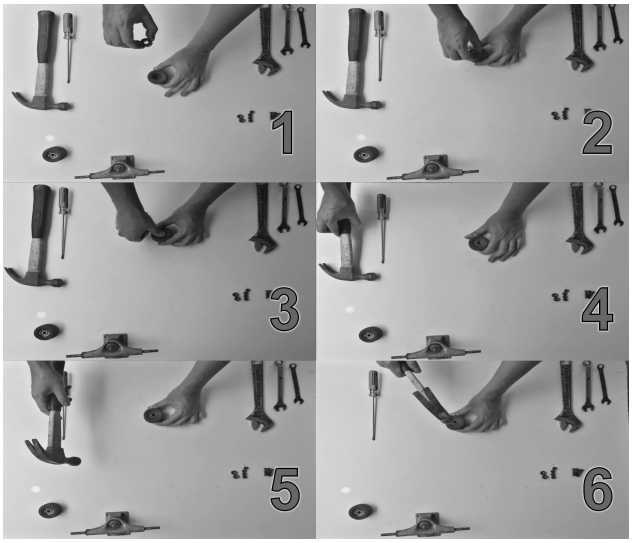


Fig. 2. Images for assembly of a skateboard truck

Assembly of a skateboard truck

The assembly process of skateboard wheels (truck) is described in Listing 9 and Figure 2. First, using a simplify version with pseudocode and MDL+ involved instructions, and next by the MDL+ language with the full listing. As it can be seen, this process involves wrenches and screwdriver-type tools, and also alternates the use of both hands.

1. The washer is taken to place it on the wheels.
```
// Frame #1
take : washer with hand;
```
2. The washer is pressed into the wheel.
```
// Frame #2
put : washer with hand in 0,0 : 1 : cm;
push : washer with hand in assembly #1;
```
3. Assemblies are joined together.
```
// Frame #3
take : assembly #1 with hand-nondominant;
hold : assembly #1 with hand-nondominant;
```
4. Take the hammer and hit the subassembly.
```
// Frame #4
take : hammer_claw with hand;
hit : hammer_claw in 0,0 : 4 : cm;
```
5. With the hammer, hit the assembly and turn it later.
```
// Frame #5
hit : hammer_claw in 0,0 : 4 : cm;
turn : assembly #1 with hand-nondominant;
```
6. The hammer is released and assembly is complete.
```
// Frame #6
release : hammer_claw with hand in 0,0 : 4 : cm;
end
```

Listing 9. Assembly of a trucker
```
assemble-begin
skateboard;
setup-begin
hand right;
bin washer in -5,20 : 3 : cm;
bin screws in -5,25 : 3 : cm;
bin nut;
tool screwdriver_phillips in -30,35 : 3 : cm;
tool wrench_adjustable in 30,35 : 3 : cm;
tool wrench_combination in 33,35 : 3 : cm;
tool wrench_combination in 36,35 : 3 : cm;
tool hammer_claw in -33,35 : 3 : cm;
assembly #1 in -10,-35 : 7 cm;
assembly #2 in -30,-32 : 2 : cm;
assembly #3 in -10,-10 : 2 : cm;
setup-end
start
// Frame #1
take : washer with hand;
move : assembly #1 with hand-nondominant
from -10,-10 : 3 to 0,0 : 4 : cm;
hold : assembly #1 with hand-nondominant;
// Frame #2
put : washer with hand in 0,0 : 1 : cm;
push : washer with hand in assembly #1;
release : assembly #1 with hand-nondominant;
// Frame #3
take : assembly #1 with hand-nondominant;
hold : assembly #1 with hand-nondominant;
// Frame #4
take : hammer_claw with hand;
hit : hammer_claw in 0,0 : 4 : cm;
spin : assembly #1 with hand-nondominant;
// Frame #5
hit : hammer_claw in 0,0 : 4 : cm;
turn : assembly #1 with hand-nondominant;
// Frame #6
release : hammer_claw with hand in 0,0 : 4 : cm;
end
assemble-end
```

Assembly for skateboard

In this example, the description of a process that involves sub-assemblies, a screwdriver, pliers-type tools and the simultaneous use of both hands is described through the language (see Listing 10 and Figure 3).

1. The workplace is configured to start
```
// See Frame 01
hand right;
assembly #truck;
assembly #deck;
tool hammer_claw;
tool clamp;
tool pliers_diagonal;
tool pliers_long_nose;
tool screwdriver_slotted;
tool screwdriver_slotted;
tool screwdriver_phillips;
tool wrench_combination;
tool wrench_combination;
tool wrench_adjustable;
bin bolt;
bin nut;
setup-end
```
2. The truck and the deck are taken to place it on the work table to start the assembly.
```
// See Frame 02
parallel-begin:
take:assembly #truck with hand-nondominant;
take:assembly #deck with hand;
parallel-end;
move:assembly #deck with hand to 10:0:cm;
```

3. The bolts and nuts of the trucker are placed on the deck with the hands.
```
// See Frame 03
take:nut with hand-nondominant;
take:screwdriver_slotted with hand;
parallel-begin:
locknut:wrench_combination with hand-nondominant in assembly #truck;
screw:screwdriver_slotted with hand in assembly #deck;
parallel-end;
```
4. The deck is turned over to continue the assembly.
```
// See Frame 04
spin:assembly #deck with any-hand;
```
5. The bolts and nuts of the trucks are placed on the deck with the hands.
```
// See Frame 05
put:nut with any-hand in assembly #deck;
take:bolt with hand-nondominant;
put:bolt with hand-nondominant in assembly #deck;
```
6. Assembly is completed.
```
end
```

Listing 10. Assembly for skateboard
```
assemble-begin
skateboard_assembly;
setup-begin
// See Frame 01
hand right;
assembly #truck;
assembly #deck;
tool hammer_claw;
tool clamp;
tool pliers_diagonal;
tool pliers_long_nose;
tool screwdriver_slotted;
tool screwdriver_slotted;
tool screwdriver_phillips;
tool wrench_combination;
tool wrench_combination;
tool wrench_adjustable;
bin bolt;
bin nut;
setup-end
start
// See Frame 02
parallel-begin:
take:assembly #truck with hand-nondominant;
take:assembly #deck with hand;
parallel-end;
move:assembly #deck with hand to 10:0:cm;
take:nut with hand;
join:assembly #truck with assembly #deck;
parallel-begin:
take:nut with hand-nondominant;
take:bolt with hand;
put:bolt with hand in assembly #deck;
put:nut with hand-nondominant in assembly #deck;
take:screwdriver_slotted with hand;
take:wrench_combination with hand-nondominant;
locknut:wrench_combination with hand-nondominant in
    assembly #truck;
screw:screwdriver_slotted with hand in assembly #deck;
parallel-end;
// See Frame 03
take:nut with hand-nondominant;
take:screwdriver_slotted with hand;
parallel-begin:
locknut:wrench_combination with hand-nondominant in
    assembly #truck;
screw:screwdriver_slotted with hand in assembly #deck;
parallel-end;
// See Frame 04
spin:assembly #deck with any-hand;
take:nut with hand-nondominant;
put:nut with hand-nondominant in assembly #deck;
take:bolt with hand-nondominant;
put:nut with hand-nondominant in assembly #deck;
take:screwdriver_slotted with hand-nondominant;
take:nut with hand-nondominant;
// See Frame 05
put:nut with any-hand in assembly #deck;
take:bolt with hand-nondominant;
put:bolt with hand-nondominant in assembly #deck;
take:screwdriver_slotted with hand;
// See Frame 06
end
assemble-end
```

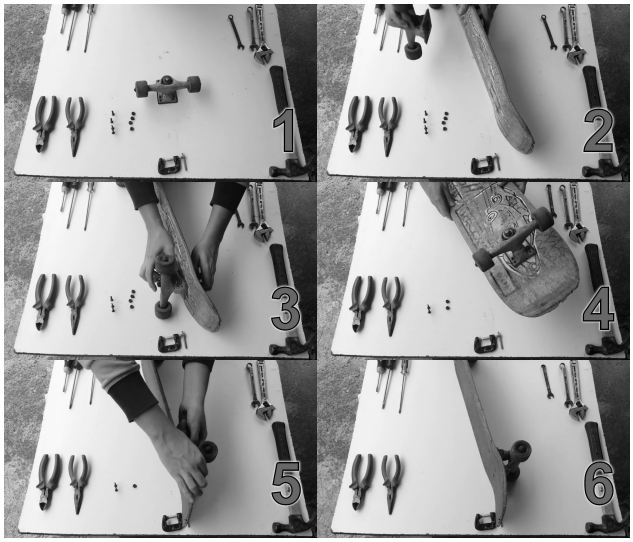


Fig. 3. Images from video of Assembly for skateboard

Assembly with parallel execution

This example was selected because it includes some sections blocks of instructions that can be executed in parallel. It has different sub-assemblies and parts to be assembled, the operator tends to perform actions in parallel, so it is necessary to reflect these capabilities in the language. See Listing 11 and Figure 4.

1. Take the wheels with both hands.
```
// See Frame 01
parallel-begin:
take:assembly #wheel1 with hand-nondominant;
take:assembly #wheel2 with hand;
parallel-end
```
2. Assembling the truck with both hands
```
// See Frame 02
move:assembly #truck with hand to 0:5:cm;
join:assembly #truck with assembly #wheel1;
hold:assembly #truck with hand-nondominant;
```
3. Put the bolt and nut on the truck
```
// See Frame 03
hold:wrench_combination with hand;
take:washer with hand; put:washer with hand in assembly #truck; tight:wrench_com
in assembly #truck; take:nut with hand; put:nut with hand in assembly
#truck;
```
4. Screw the base plate with the screw and nut on the truck.
```
// See Frame 04
tight:wrench_combination in assembly #truck;
move:assembly #truck with hand to 0:5:cm;
take:bolt with hand;
take:assembly #base_plate with hand-nondominant;
```
5. Assemble the screw, washer, nut and base plate on the trucker using the pliers.
```
// See Frame 05
join:bolt with assembly #base_plate;
take:washer with hand;
put:washer with hand in assembly #base_plate;
```

```
hold:pliers_long_nose with hand;
hold:pliers_long_nose in assembly #base_plate;
take:assembly #base_plate with hand-both;
join:assembly #truck with assembly #base_plate;
```

6. Place the nut to tighten it with the pliers on the truck.

```
//See Frame 06
put:nut with hand in assembly #truck;
hold:pliers_long_nose with hand;
hold:pliers_long_nose in assembly #base_plate;
move:assembly #base_plate with hand to 0:0:cm;
```

Listing 11. Assembly with parallel execution

```
assemble-begin
example03;
setup-begin
hand right;
assembly #truck in 55:-28:cm;
assembly #wheel1 in 14:60:cm;
assembly #wheel2 in 14:45:cm;
assembly #base_plate in 0:5:cm;
tool hammer_claw;
tool clamp;
tool pliers_diagonal;
tool pliers_long_nose;
tool screwdriver_slotted;
tool screwdriver_slotted;
tool screwdriver_phillips;
tool wrench_combination;
tool wrench_combination;
tool wrench_adjustable;
bin bolt in -50:-30:cm;
bin washer in -32:-6:cm
bin bolt in -30:30:cm;
setup-end
start
// See Frame 01
parallel-begin:
take:assembly #wheel1 with hand-nondominant;
take:assembly #wheel2 with hand;
parallel-end
// See Frame 02
move:assembly #truck with hand to 0:5:cm;
join:assembly #truck with assembly #wheel1;
hold:assembly #truck with hand-nondominant;
// See Frame 03
hold:wrench_combination with hand;
take:washer with hand;
put:washer with hand in assembly #truck;
tight:wrench_combination in assembly #truck;
take:nut with hand;
put:nut with hand in assembly #truck;
// See Frame 04
tight:wrench_combination in assembly #truck;
move:assembly #truck with hand to 0:5:cm;
take:bolt with hand;
take:assembly #base_plate with hand-nondominant;
// See Frame 05
join:bolt with assembly #base_plate;
take:washer with hand;
put:washer with hand in assembly #base_plate;
hold:pliers_long_nose with hand;
hold:pliers_long_nose in assembly #base_plate;
take:assembly #base_plate with hand-both;
join:assembly #truck with assembly #base_plate;
// See Frame 06
put:nut with hand in assembly #truck;
hold:pliers_long_nose with hand;
hold:pliers_long_nose in assembly #base_plate;
move:assembly #base_plate with hand to 0:0:cm;
end assemble-end
```

Conclusions

This paper proposes the extension MDL+ of our language for the description of the activities of manufacturing operations MDL [Zamora-Hernández et al., 2021]. To the best of our knowledge, it is the first structured language since IBM's 1986 proposal that can be used to describe manufacturing processes and guide product quality control through manufacturing supervision. Specifically, this language is part of a computer vision control system that allows determining the quality level of a product, and defines the steps of how it was built according

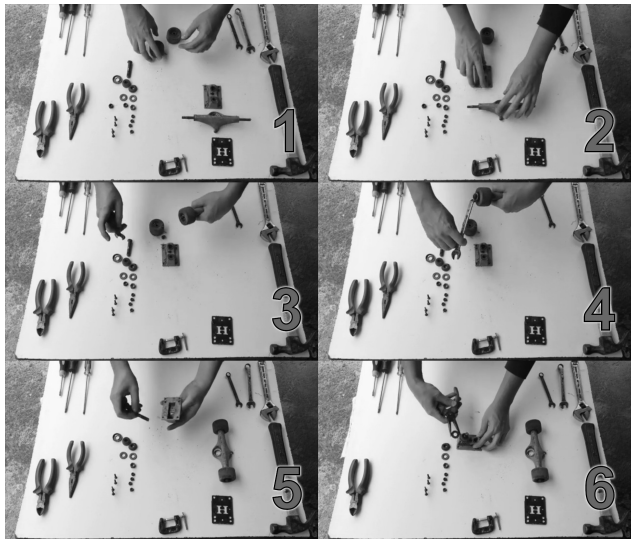


Fig. 4. Images for assembly with parallel execution

to the specifications of production in each organization. The language also allows transforming different process description systems in industries and works as a suggestion system for operators to minimize errors, creating a "poka yoke" system for assemblies. Moreover, we extended the features of MDL including extension to represent parallel task developed in a coordinate way by different operators or with a single operator but using both hands. On the other hand, it introduces new quantative experimentation with different actions and evaluating the advantages of using the language with operators manufacturing evaluation tests.

As future lines of research, it is proposed to use this language in conjunction with video analysis systems to formalize the instructions carried out and verify if the instructions described in the proposed language are satisfied. In addition, it is very useful in the area of job design for job measurement and standard time calculation, along with Operations Engineering to calculate production capacity. The system can promote occupational safety by signaling to the operator that there are items that should not be present.

Competing interests

There are NO Competing Interest.

Author contributions statement

All the authors contributed in the design of the language equaly. M-A.Z conceived and conducted the experiments with operators and analyzed the results. J-A.C.C; A.V.G & J-A.C.V recorded the assembly actions experiments and codify them with the language. M-A.Z, J.G & J.A analysed the experiments results, wrote and reviewed the manuscript.

Acknowledgments

This work has been funded by the Spanish Government PID2019-104818RB-I00 grant for the MoDeaAS project and TIN2017-89069-R grant, supported with Feder funds. This work

has also been supported by University of Alicante grant for PhD studies UAFPU2019-13.

References

Å. Fast-Berglund, T. Fässberg, F. Hellman, A. Davidsson, and J. Stahre. Relations between complexity, quality and cognitive automation in mixed-model assembly. *Journal of Manufacturing Systems*, 32(3):449–455, 2013. ISSN 02786125.

D. Ferguson. Therbligs: The Keys to Simplifying Work, 2000. URL `http://web.mit.edu/allanmc/www/Therblgs.pdf`.

M. P. Groover. *Work Systems and the Methods, Measurement, and Management of Work*. Pearson Education, Inc., 2007. ISBN 0-13-140650-7.

M. Hedelind and M. Jackson. How to improve the use of industrial robots in lean manufacturing systems. *Journal of Manufacturing Technology Management*, 22 (7):891–905, sep 2011. ISSN 1741-038X. doi: 10.1108/17410381111160951. URL `http://www.emeraldinsight.com/doi/abs/10.1108/17410381111160951`.

A. Hornung, M. Bennewitz, and H. Strasdat. Efficient vision-based navigation. *Autonomous Robots*, 29(2):137–149, 2010. ISSN 09295593. doi: 10.1007/s10514-010-9190-3.

G. Kanawaty. *Introducción al estudio del Trabajo*. EDITORIAL LIMUSA DE C.V., 11 edition, 2008. ISBN 978-968-18-5628-1.

R. Krishna, K. Hata, F. Ren, L. Fei-Fei, and J. C. Niebles. Dense-Captioning Events in Videos. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:706–715, 2017. ISSN 15505499. doi: 10.1109/ICCV.2017.83.

R. C. Luo, Y. T. Hsu, Y. C. Wen, and H. J. Ye. Visual image caption generation for service robotics and industrial applications. *Proceedings - 2019 IEEE International Conference on Industrial Cyber Physical Systems, ICPS 2019*, pages 827–832, 2019. doi: 10.1109/ICPHYS.2019.8780171.

S. Makris, P. Karagiannis, S. Koukas, and A. S. Matthaiakis. Augmented reality system for operator support in human???robot collaborative assembly. *CIRP Annals - Manufacturing Technology*, 65(1):61–64, 2016. ISSN 17260604. doi: 10.1016/j.cirp.2016.04.038. URL `http://dx.doi.org/10.1016/j.cirp.2016.04.038`.

M. Mancini, H. Karaoguz, E. Ricci, P. Jensfelt, and B. Caputo. Kitting in the Wild through Online Domain Adaptation. *IEEE International Conference on Intelligent Robots and Systems*, pages 1103–1109, 2018. ISSN 21530866. doi: 10.1109/IROS.2018.8593862.

L. R. Nackman, M. A. Lavin, R. Highsmith Taylor, W. C. Dietrich, and D. D. Grossman. AML/X: a programming language for design and manufacturing. *ACM '86: Proceedings of 1986 ACM Fall joint computer conference*, 1986. doi: 10.5555/324493.324560. URL `https://dl.acm.org/doi/10.5555/324493.324560`.

A. Nguyen, T.-T. Do, I. Reid, D. G. Caldwell, and N. G. Tsagarakis. V2cnet: A deep learning framework to translate videos to commands for robotic manipulation, 2019.

J. Starke, K. Chatzilygeroudis, A. Billard, and T. Asfour. On Force Synergies in Human Grasping Behavior. *IEEE-RAS International Conference on Humanoid Robots*, 2019-October:72–78, 2019. ISSN 21640580. doi: 10.1109/Humanoids43949.2019.9035047.

Universidad Politécnica deValencia. Therbligs, 2018. URL `http://evaluador.doe.upv.es/wiki/index.php/Therbligs`.

L. Wang, B. Schmidt, and A. Y. C. Nee. Vision-guided active collision avoidance for human-robot collaborations. *Manufacturing Letters*, 1(1):5–8, 2013. ISSN 22138463.

X. Wang, W. Chen, J. Wu, Y. F. Wang, and W. Y. Wang. Video Captioning via Hierarchical Reinforcement Learning. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4213–4222, 2018. ISSN 10636919. doi: 10.1109/CVPR.2018.00443.

Y. Yang, Y. Li, C. Fermüller, and Y. Aloimonos. Robot learning manipulation action plans by "watching" unconstrained videos from the World Wide Web. *Proceedings of the National Conference on Artificial Intelligence*, 5:3686–3692, 2015.

T. Yao, Y. Pan, Y. Li, Z. Qiu, and T. Mei. Boosting Image Captioning with Attributes. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob: 4904–4912, 2017. ISSN 15505499. doi: 10.1109/ICCV.2017.524.

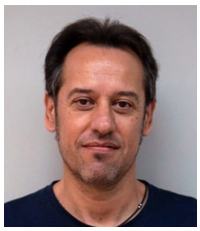
M.-A. Zamora-Hernández, J. A. C. Ceciliano, A. V. Granados, J. A. C. Vargas, J. Garcia-Rodriguez, and J. Azorín-López. Manufacturing description language for process control in industry 4.0. In Á. Herrero, C. Cambra, D. Urda, J. Sedano, H. Quintián, and E. Corchado, editors, *15th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2020)*, pages 790–799, Cham, 2021. Springer International Publishing. ISBN 978-3-030-57802-2.

Mauricio Andrés Zamora Hernández. received his Ph.D. degree, with specialization in Automatics and robotics, from the University of Alicante (Spain). He is currently Professor at School of Industrial Engineering of the University of Costa Rica. His research areas of interest include: computer vision, machine learning, pattern recognition, robotics, human - computer interaction, manufacturing, software arquitecture.

Jose Andrez Chaves Ceciliano. is a Bachelor of Industrial Engineering student at the University of Costa Rica (UCR). He received his title of medium technician in Electrotechnical at Heredia Professional Technical College in 2014. His main research interests include deep learning in robotic, virtual reality, value chain logistics, quality engineering and operations engineering.

Alonso Villalobos Granados. is a Bachelor of Industrial Engineering student at the University of Costa Rica (UCR), that also has been working as a researcher for the Industrial Engineering School (EII) since 2019, who also happens to be an avid outdoorsman. His main research interests go along the line of deep learning, computer vision, human behavior recognition, industrial process automation and supply chain management.

John Alejandro Castro Vargas. is a PhD student at the Department of Computer Technology (DTIC), University of Alicante. He received his MSc (Automation and Robotics) and BSc (Computer Science) from the same institution in 2017 and 2016 respectively. His main research interests include human behavior recognition with deep learning, virtual reality and parallel computing on GPUs.



Jose Garcia-Rodriguez received his Ph.D. degree, with specialization in Computer Vision and Neural Networks, from the University of Alicante (Spain). He is currently Full Professor at the Department of Computer Technology of the University of Alicante. His research areas of interest include: computer vision, machine learning, pattern recognition, robotics, man-machine interfaces, ambient intelligence, and parallel and multicore architectures.

Jorge Azorin-Lopez received his Ph.D. degree in Computer Science at the University of Alicante (Spain) in 2007. Since 2001, he has been a faculty member of the Department of Computer Technology at the same university, where he is currently an Associate Professor. His current research interests include 3D computer vision, computational intelligence, human activity analysis, and visual inspection.