The International Journal of Advanced Manufacturing Technology (2021) 117:3873–3893
https://doi.org/10.1007/s00170-021-07824-7

APPLICATION

ViTroVo: in vitro assembly search for in vivo adaptive operator guidance

An artificial intelligence framework for highly customised manufacturing

Corrado Grappiolo[1] · **Raimon Pruim**[2] · **Matthias Faeth**[1] · **Paolo de Heer**[3]

Received: 26 February 2021 / Accepted: 27 July 2021 / Published online: 31 August 2021

Abstract
Product customisation is a topic of growing interest in Smart Manufacturing. Allowing customers to design intended products brings additional challenges to the manufacturing task, such as the increase in flexibility of the assembly theatre, the compilation of assembly instructions for possibly unique products, and stress-related risks for human operators. This work introduces ViTroVo, an artificial intelligence framework capable of (1) autonomously building a graph of assembly steps via trial-and-error (in vitro Assembly Search) and (2) presenting relevant instructions to a human operator and, by autonomously detecting her progress and affective state, adapting accordingly (in vivo Adaptive Operator Guidance). The power of ViTroVo resides in its versatile way to manipulate a given product's component Augmented Computer Aided Design (CAD+) models throughout the whole assembly task. We conducted an empirical evaluation involving participants instructed to assemble a previously unseen product. The encouraging results make us believe ViTroVo's architecture could become the foundations of highly customised flexible manufacturing.

Keywords Customised manufacturing · Assembly Sequence Planning · Adaptive Operator Guidance · Augmented CAD models · Virtual Environment · Artificial Intelligence

1 Introduction

The last years have been witnessing an enormous contribution of Information and Communication Technology (ICT) in many physical domains. Industrie or Industry 4.0, a term coined to clearly distinguish new, ICT-enhanced industrial processes from those proper of the second half of the twentieth century, such as mass-scale manufacturing [1, 27], is a clear example. A term extremely related to Industry 4.0, to the point that sometimes it is also used as synonym, is Smart Manufacturing [26, 30, 72]. Smart Manufacturing covers a huge spectrum of paradigms, such as human-robot cooperation [12], (big) data collection from physical sensors [60] and simulation/modelling [59].

A growing trend in Smart Manufacturing is product customisation [43, 71] which, in a nutshell, aims to give higher levels of control to customers. Companies have already started providing basic customisation services (see for instance [23, 53]) and the newly coined Industry 5.0 term — which aims to characterise human-centric industry by means of, e.g., mass-customisation — starts to emerge, both in research [9, 46] and political domains [6, 44]. Product customisation will increase in importance and versatility in the coming years, to the point that customers could, for instance, design their idealised products by selecting components from generic online repositories [5]

✉ Corrado Grappiolo
corrado.grappiolo@tno.nl

Raimon Pruim
raimon.pruim@tno.nl

Matthias Faeth
matthias.faeth@tno.nl

Paolo de Heer
paolo.deheer@tno.nl

1 TNO Data Science, Anna van Buerenplein 1, NL-2595 DA The Hague, The Netherlands

2 TNO Intelligent Imaging, Oude Waalsdorperweg 63, NL-2597 AK The Hague, The Netherlands

3 TNO Modelling, Simulation and Gaming, Oude Waalsdorperweg 63, NL-2597 AK The Hague, The Netherlands

and, subsequently, modify their morphologies or even add customer-made parts. We would, in other words, witness the transformation of traditional manufacturing into a service-on-demand process. We will refer to this not-so-futuristic scenario as Highly Customised Manufacturing (HCM).

Customised manufacturing requires such a high degree of flexibility in the assembly theatre/process that it can hardly exist without human operators [7, 10].

Operators have to be guided throughout the assembly task via instructions, commonly compiled either manually or algorithmically. The manual compilation is a rather expensive process, usually performed for large batch sizes, as the compilation costs can be amortised. Intuitively, such approach would become unfeasible for HCM, which instead pushes the manufacturing process towards the batch-of-one scenario. Although the algorithmic way is to some extent cost-invariant against the batch size, the most commonly implemented approaches follow a disassemble-assemble strategy [52]. Starting from the final product's full design — usually handcrafted by domain experts — the algorithms first disassemble the product until loose components are obtained, then revert the result to create a graph of assembly steps. As HCM would allow customers to modify products with a high level of agency, it is likely that full knowledge on the final product's design might cease to exist, with the consequence that a disassembly-assembly approach would no longer be feasible. To be effective, HCM needs algorithmic approaches which can still generate assembly instructions based on less-than-full knowledge on the final product, hence by skipping the preliminary disassemble step.

HCM should not only give a centric role to the customer, it should also enhance the operator experience and make sure her well-being is as high as possible [39]. This can be achieved by monitoring her physiological state throughout the assembly task and by intervening in case this worsens, for example, if the algorithmically generated instructions are misunderstood, hence misexecuted. In these circumstances the interventions could correct the operator by suggesting to reverse the erroneous assembly step or, in case the misexecution could still lead to the final product, seamlessly guide the operator by presenting alternative instructions.

All in all, HCM would require a large body of automated computing techniques which would not only make possible to engineer end-user-driven (nearly) unique products, it would achieve so by tailoring the assembly task to the human operator's affective state and experience level [38].

Based on these considerations, the work presented in this manuscript assumes a Highly Customised Manufacturing scenario in which only partial knowledge regarding the final product to be assembled by a human operator is in possess. It then centres its attention on the following three research questions:

RQ1 how can assembly steps from loose components to final product be procedurally generated without relying on disassemble-assemble algorithmic approaches?

RQ2 How can these be used to adaptively guide a human operator so that her misunderstandings and/or misexecutions can be promptly corrected?

RQ3 How can her affective state throughout the given task be taken into account?

We tackled the questions by leveraging Artificial Intelligence (AI) and virtualisation. The outcome is ViTroVo[1], a framework which seamlessly transitions from a virtual phase (in vitro), in which assembly steps are sought, to a physical phase (in vivo), in which the product is assembled. The in vivo phase realises two communication flows, Operator-to-ViTroVo and vice versa. The former monitors the assembly stage, gathers the operator's affective state and other performance-related information, and updates the data structure used by the second, ViTroVo-to-Operator flow, in charge of presenting the most appropriate instructions[2].

We conducted an early experimental investigation on a fictional product composed of five parts. ViTroVo managed to efficiently find the appropriate sequences of assembly steps and, based on them, to successfully guide the participants of our experiment throughout the given task. We believe that ViTroVo has the potential to become the reference architecture for Highly Customised Manufacturing, given its modular composition — which makes it easily extendable — and the fact that the whole manufacturing journey leverages the same virtual environment and component models — albeit with ad hoc configurations.

The remainder of this manuscript is organised as follows: Section 2 provides an overview of the most relevant work to our research. Section 3 introduces the architecture of ViTroVo. Section 4 presents the case study of our investigation. The framework's technical details begin with Section 5, in which the virtualisation aspects are introduced; Section 6 delves into the task of retrieving the assembly steps; Section 7 describes the processes needed to prepare the module in charge of monitoring the physical assembly stage. Section 8 goes into the intricacies of operator guidance and its two information flows. Section 9 delves into the initial evaluation of our framework. Section 10 dissects the strengths and weaknesses of our framework and outlines future work. Section 11 concludes the manuscript.

[1]"Vi trovo", in Italian, means "I am going to find you (plural)".

[2]A video overview of ViTroVo can be found at the following link: https://vimeo.com/567089060

2 Related work

Our research tackles three main topics: retrieving assembly steps, adaptively presenting instructions to a human operator, and training neural networks with synthetic data to *understand* real-world assembly stages. Intuitively, all of them are supported by a large amount of research. The current section presents the work most closely related to ours.

Assembly Sequence Planning [52] (ASP) is a research field much similar to our Assembly Search process. ASP refers to a task for which planners, on the basis of their particular heuristics in assembling all the components of a product, arrange a specific assembly sequence according to the product design description [64]. Generally speaking, given a fully assembled product specification (components and how they are connected), ASP aims to first disassemble the product [33] — so that all possible steps leading to loose components can be found — and then to reverse the result, in order to obtain an assembly graph, usually represented via AND/OR graphs [8, 37, 45]. The most important difference between ViTroVo's Assembly Search (AS) and ASP is in the lack of a fully assembled product description, hence the fact that AS does not perform any disassembly-based search. Nonetheless, it is likely that the soft-computing algorithms used by ASP — e.g. ant colony optimisation [66], particle swarm optimisation [67] or evolutionary computation [50] — could not only compete with our Monte Carlo Graph Search (MCGS) algorithm (see Section 6.1) in generating a graph more efficiently, but also work in symbiosis with each other, as outlined in Section 10.

AND/OR graphs provide a compact representation of assembly plans, are extremely powerful in enabling parallel assembly, and are equivalent to the directed graph of assembly states [52]. It is therefore clear that ViTroVo should attempt to achieve such translation, as it would facilitate its application to teamed assemblies [45] and, furthermore, assembly line balancing [17, 56], a fundamental research domain linked to ASP. For a non-exhaustive overview on ASP, its approaches and techniques we recommend the literature review work of Rashid et al. [50], Su et al. [57], Guo et al. [18], and Lambert [31]. A closely ASP-related line of research is motion planning. The idea is much similar to the one we use for the *join* assembly action (see Section 6.1.1): executing disassembly motions in simulated environments to retrieve the (dis)assembly order. With this respect, the work of Xiong et al. [70] or Morato et al. [42] on Rapid-growing Random Trees (RRT) should also be investigated for ViTroVo. Along these lines, we highlight the promising work of Kaipa et al. [25], which relied on RRT to procedurally generate animated instructions of a product composed of 71 parts.

Arguably, the research on operator guidance is even bigger than ASP's, as it can easily ramify in multiple fields, e.g. human-computer interaction, user experience and cognitive psychology research. By focusing on the mere technical aspects related to animated instruction generation, Hořejší et al. [20] developed a module for the Unity Software suite — the same used by ViTroVo (see Section 5) — to generate assembly instruction videos. More remarkably, they conducted a thorough investigation on what instruction modalities — paper, virtual and video — have a better impact on the assembly performance. Similar work, conducted, e.g., by Watson et al. [69], empirically motivate our aim to generate animated virtual instructions. Nonetheless, their studies do not take into account the affective state of the operator but only the time taken to assemble a given product. Other work on adaptive instruction formalisation and design are those of Funk et al. [16], in which they outline requirements for cognitive-driven adaptive assistance, and the work of Mattsson and Fast-Berglund [38, 40], in which they provide guidelines to generate instructions facilitating the operator's expertise.

Cognitive modelling of an operator for her guidance is tightly related to her affective modelling. ViTroVo has just scratched the surface of this extremely important topic, by facilitating the affective data collection via computer vision (see Section 8.1). Further research and enrichment of our framework should definitely take into account, among others, the work of bin Khairai et al. [28], who examined the correlation between operator stress and work performance, and Mattsson et al. [41], in which they relied on wearable devices — a perfect additional modality ViTroVo could leverage — to monitor physiological features such as heart rate and skin temperature.

An effective adaptive operator guidance needs automated ways to detect the (mis)execution of given instructions. Antifakos et al. [2] centre their attention in detecting the assembly actions performed by human operators and in mapping them to a pre-defined assembly graph. They identify three very relevant modalities to present instructions: full-walk-through, assistance-on-demand and rescue-from-trap. Our work, currently, has a single modality of instruction; it would hence be beneficial to find ways to integrate their approach with Operator Modelling (see Section 10). Their assembly stage perception is achieved via motion sensors embedded in each component, rather than our vision-based, and their localisation process relies on Markov chains.

Another remarkable work with a focus on video-based localisation is done by Wang et al. [65]. They rely on probabilistic inference, an approach which could certainly improve ViTroVo's greedy localisation (outlined in Algorithm 2). Moreover, Wang et al. assume some additional information to the component's virtual models.

Their so-called part-interaction rules are extremely similar to the Augmented Computer Aided Design (CAD+) connection points/groups that will be used in our work (see Section 5). Our models, however, expect additional information (reported in Table 1), as they are used in different ways by different processes.

In order to provide appropriate guidance to the human operator using instructions, the progress of the assembly needs to be monitored. Most practically, this is done using a video-based system. This requires functionalities to analyse the video-stream, captured by a camera above the assembly table, to determine the assembly stage. Essential for such a system is to recognise and localise the components in view. Deep learning models are superior in performing such visual tasks, albeit that training such models typically requires large sets of manually annotated ground truth data. Such an annotation effort is not only very time consuming, it is also not available for our current use case. A promising approach to circumvent manual annotation is through the use of automatically generated labelled data using simulation. In a virtual environment, images can be captured from simulated scenes which are representative for the real world. This provides an automated process for the fully controlled generation of large amounts of labelled data. However, such approach is not sufficient to let a neural network operate on real-world images. This is due to the "reality gap" between the simulated environment and the real world [62]. Various techniques are developed to bridge this gap, such as photorealistic simulations [24], domain randomisation [62, 63] and structured domain randomisation [49].

Photorealistic simulations aim to bring the simulated environment as close to the real world as possible. Domain randomisation focuses on generating simulated scenes using highly randomised parameters (e.g. positioning, orientation, lighting, colour, texture). By varying over these parameters and rendering non-realistic images, the neural network is trained to become invariant of these variations and only learn essential features which would transfer to the real world, hence enforcing domain invariance. Structured domain randomisation aims to leverage the benefits of both realistic simulations and randomisation, by only randomising parameters within realistic ranges which take into account the structure and context of a scene. This method outperforms domain randomisation and can achieve performances closer to those obtained after trained on real-world data, albeit at the cost of higher simulation complexity and efforts [49].

3 The ViTroVo framework

The overall architecture of ViTroVo is presented in Fig. 1. The framework bridges the customer's Product Ideation phase with her Final Product. It does that by transitioning from a purely virtual phase — hereafter called in vitro — during which all data and computational modules are generated, to a physical phase — hereafter called in vivo — in which a human operator is guided throughout the actual assembly task.

The two maximum common divisors for the ideation, in vitro and in vivo phases correspond to the customised product's component models and a Virtual Environment. The former, which we will hereafter refer to as Augmented Computer Aided Design (CAD+) models, represent components with different configurations, so that each process can utilise them at best. The latter is the tool processes leverage to ad hoc manipulate CAD+ models. Currently, our framework relies on the Unity Software suite [61].

The highly customised manufacturing journey begins with Product Customisation. In this process the customer designs the final product, for instance, by selecting components from a database, by modifying their morphology, by combining them with customer-made models, and by configuring global functionalities. The output of this process are the Product Specifications. Intuitively, the richer the specifications, the more powerful ViTroVo. As our research has not fully delved into Product Customisation yet, the design of our framework assumed two minimal specifications: the exhaustive list of CAD+ models composing the product and the final product's partial look. As soon as Product Specifications are

Table 1 ViTroVo's CAD+ Models and Virtual Environment key-properties

Process		CAD+ Models			
	Uses Virtual Environment	Collision Detection	Component Motion	Connection Type	Connection Medium
Product Customisation	Yes	Detailed	No	Yes	No
Assembly Search	Yes	Detailed	Detailed	Yes	No
Assembly Stage Detector Preparation	Yes	No	No	No	Yes
Adaptive Operator Guidance	Yes	No	Detailed	Yes	Yes
Component Manufacturing	No	No	No	No	Yes

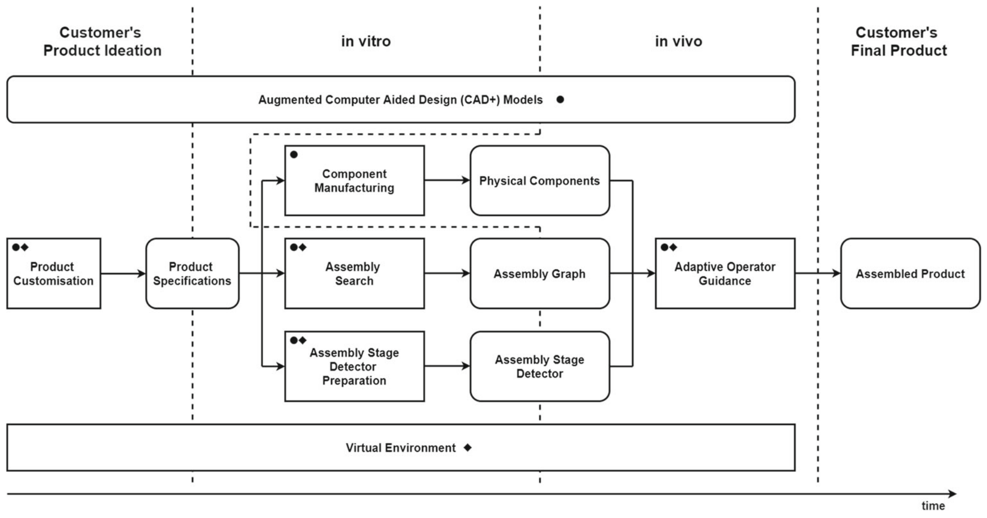


Fig. 1 The schematic representation of ViTroVo. The framework is presented with a left-to-right information/processing flow and an indicative timeline, so that concurrent processes and dependencies can easily be observed. Since the CAD+ models and the Virtual Environment are used throughout the whole manufacturing journey, albeit in different configurations, their link with other processes is represented by, respectively, the circle and rhombus symbols. Round-edge square items correspond to data and models; sharp-edge square items to processes. The two in vitro and in vivo phases are also outlined. The details of Assembly Search, Assembly Stage Detector Preparation and Adaptive Operator Guidance can be found, respectively, in Figs. 6, 11 and 14

defined, the in vivo Component Manufacturing process begins. The process, which is not covered in our work, can be executed, e.g., via three-dimensional (3D) printing of the CAD+ component models.

Product Specifications are also necessary to begin the in vitro phase. This phase is composed of two concurrent processes, Assembly Search and Assembly Stage Detector Preparation. Both of them rely on CAD+ models and the Virtual Environment. Assembly Search is the process in charge of exploring the combinatorial space of assembly steps, of evaluating their effectiveness with respect to the Product Specifications, and of ultimately storing them into an Assembly Graph. The graph has five distinctive characteristics: (1) its nodes represent assembly stages (e.g. how components are joined with each other), (2) its edges represent assembly actions (e.g. *joining* two components), (3) it contains one initial assembly stage (i.e. no incoming edges), corresponding to all loose components (*root* node), (4) it contains one unique node, representing the final assembly stage, corresponding to all components assembled into the final product (*goal* node), and finally (5) it contains more than one path from the root to the goal node. Essentially, Assembly Search simulates the physical assembly task by manipulating CAD+ models in the Virtual Environment.

The Assembly Graph contains the needed (in vitro) steps to guide the human operator throughout the assembly task via the in vivo Adaptive Operator Guidance process. The process leverages CAD+ models and the Virtual Environment to generate animated instructions. Adaptive Operator Guidance also needs to keep track of the physical assembly stage — which we assume it has a corresponding node in the Assembly Graph — so that the most appropriate instruction can be provided. One way is to let the operator query the framework for the next instruction. Although this can be sufficient in case of correct execution of the last given instruction, it would fail in case of misexecutions. For this reason, Adaptive Operator Guidance relies on a deep learning model (Assembly Stage Detector) to visually perceive the current physical stage and to identify the corresponding node into the graph (localisation). Adaptive Operator Guidance augments the assembly graph with operator's performance and affective data. This can additionally be used to generate more personalised paths and instructions, for instance, to minimise the operator's stress levels.

The last in vitro process, Assembly Stage Detector Preparation, is in charge of preparing the detector. This is done by exploiting deep learning models to detect the location and orientation of components, respectively using object detection and image retrieval methodology. Both of them are based on component images. However, as the real-world images do not exist yet, ViTroVo leverages the CAD+ models and the Virtual Environment to generate synthetic images.

4 A case study: product specifications

We imagine a customer designing a remotely connected luminaire. Five components make the final product: a Printed Circuit Board (PCB), a Light Emitting Diode (LED)

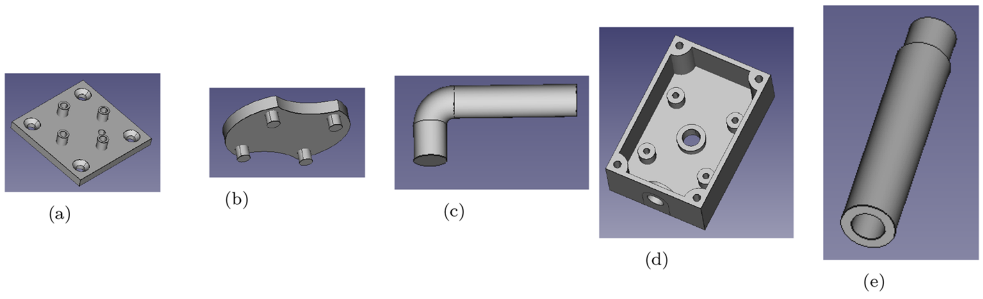


Fig. 2 The CAD models of our case study. (a) PCB. (b) LED board. (c) Antenna. (d) Encasing. (e) Stand

board, a radio-signal receiver (antenna), an encasing and a hollow stand connector. The related virtual models, which we assume were chosen by the customer, are designed by means of FreeCAD [51]. Figure 2 depicts the component CAD models designed in FreeCAD. Figure 3 depicts the derived CAD+ models used in our work (see Section 5). Figure 4 depicts their 3D-printed version, that is, the output of Product Manufacturing.

We assume that the only information regarding the final product specifications corresponds to its final assembled look. We do not have any information regarding (1) what component connects to which other, (2) in which order and (3) with which orientation. A snapshot of such assembly can be observed in Fig. 5.

5 CAD+ models and Virtual Environment

The vast majority of industrial manufacturing relies on Computer Aided Design (CAD) models of products and components. These models, often made by specialists, have the purpose to describe, in details, components for product manufacturing and/or final, assembled products. Essentially, their definition is narrowly scoped. CAD models for Highly Customised Manufacturing necessarily require additional (meta) information, as the models have a much broader scope — the very assumption made of a HCM's component database makes these compatible with a multitude of other components in yet-to-exist products — and are also algorithmically manipulated to determine the assembly steps. We will refer to them as Augmented CAD (CAD+) models.

In the research work we here present, the augmentation is obtained by storing the meta information in external comma-separated values tables and by manually shaping collider meshes within ViTroVo's Virtual Environment. Nonetheless, as we will state in Section 10, CAD+ models should become the standard representation for Highly Customised Manufacturing. This means that CAD software editors should equip themselves with functionalities aimed at creating CAD+ models seamlessly.

Aside from Component Manufacturing, the other four processes that use CAD+ models — i.e. Product Customisation, Assembly Search, Assembly Stage Detector Preparation and Adaptive Operator Guidance — also leverage the Virtual Environment. These processes actively manipulate the models, albeit with different purposes and level of details. We identified four key-properties. These are summarised in Table 1.

The first property is Collision Detection, that is, the ability to simulate the physical rigidity of the models and to determine whether two or more components intersect each other. Collision detection is fundamental for Product Customisation — as the customer should design realistic products — and Assembly Search — as this information is used to ultimately determine the order of the assembly steps. For these processes, collision detection has to be realised with the highest level of detail, that is, it must be able to take into account convexities and cavities in the models. On the other hand, neither Assembly Stage Detector Preparation nor Adaptive Operator Guidance need collision detection: the former treats collisions as partial occlusions Assembly Stage Detector must learn to cope with, the latter simply replays the (collision-free) steps in the Assembly Graph.

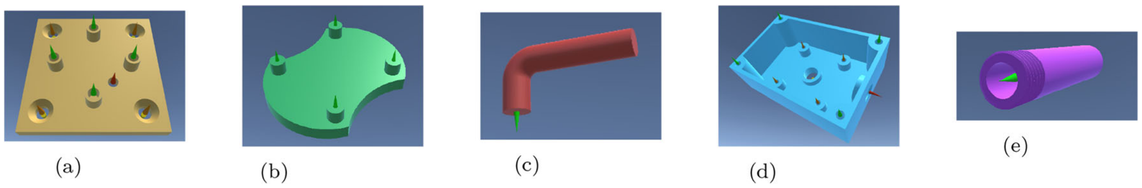


Fig. 3 The CAD+ component models of our case study. For each component, connection points with normal vectors (cones and their directions) are shown. Connection points with the same colour belong to the same group. In the encasing (d) and stand (e) it is possible to observe the threading detail. (a) PCB. (b) LED board. (c) Antenna. (d) Encasing. (e) Stand

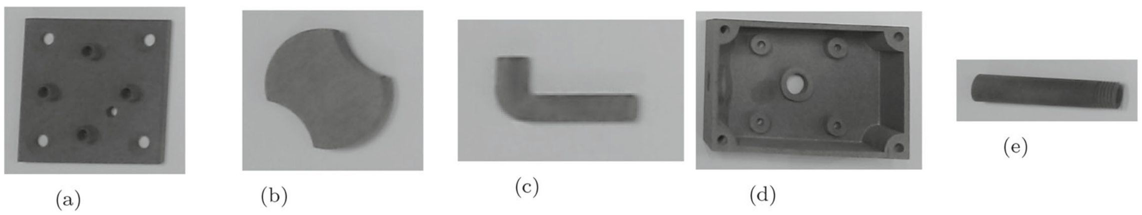


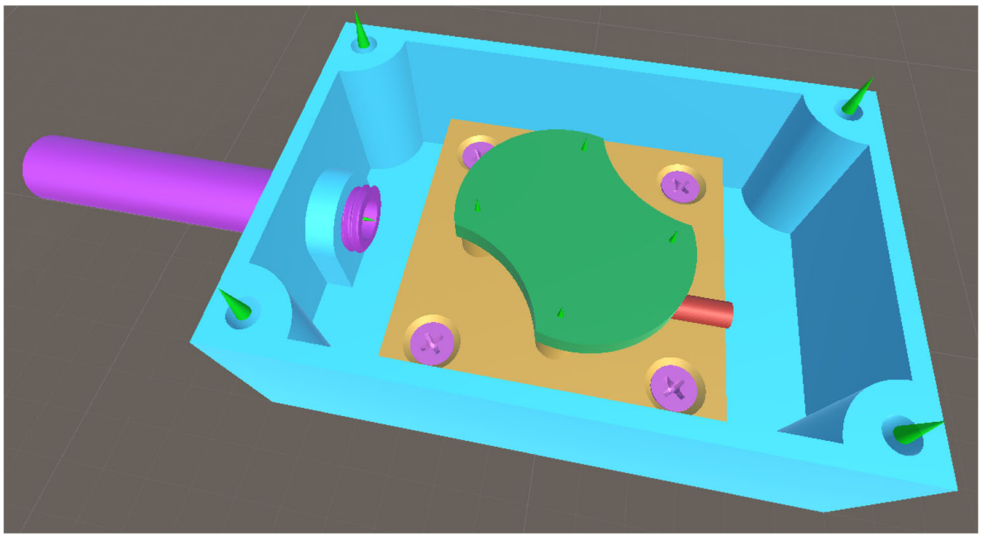
Fig. 4 The 3D-printed components as perceived by the camera used in our empirical investigation (see Section 9). (a) PCB. (b) LED board. (c) Antenna. (d) Encasing. (e) Stand

The second property is the dynamic motion of components in the Virtual Environment. We assume Product Customisation is realised via a software tool which allows the customer to manipulate objects within ViTroVo's Virtual Environment. The customer can drag and drop components, hence, no motion is required. Different case is for Assembly Search: the process sets the CAD+ models in motion to allow them to combine with others, to detect possible collisions, hence to identify promising assembly steps. Assembly Stage Detector Preparation does not need any form of motion, as its purpose is solely to generate images. Adaptive Operator Guidance, intuitively, does generate animated instructions with the same detail of Assembly Search.

The third and fourth properties specify how components can connect with each other by, respectively, the type and medium. Connection types, e.g. screw hole or pin, help in constraining which components can connect with each other. This information is essential during Product Customisation — to subtly coerce the customer to design feasible products —, Assembly Search — to restrict the combinatorial space of possible component combinations — and, intuitively, Adaptive Operator Guidance. Assembly Stage Detector Preparation does not need such information, as it does not aim to generate examples of combined components. Connection medium is the information that specifies how the connection type is realised in the physical world. Example of mediums are screws and threaded component sockets. Connection medium is needed by Assembly Stage Detector Preparation — as the generated images must contain as many physical details as possible —, Adaptive Operator Guidance — as an operator must know, e.g., whether, how many and where screws should be placed — and, needless to say, Component Manufacturing. Product Customisation does not necessarily need this level of detail. Assembly Search does not need it at all, as it would bring additional and not needed collision detection hindrances. All CAD+ models specify the position and orientation of their connection points (see, for instance, Fig. 3). Connection points are also grouped together: two components can combine with each other by means of two connection groups if and only if these have the same type, all points can virtually overlap without collisions, and that their respective normal vectors are inverted.

One final remark regarding the Virtual Environment is due. As we will thoroughly explain in Section 6.1.1, Assembly Search visually compares its generated product with the partial view stored in Product Specifications. As both processes share the same Virtual Environment, the two assembled and customised images would be taken from the same virtual camera position and orientation.

Fig. 5 The assembled product



6 Assembly Search

Assembly Search explores the combinatorial space of assembly stages to retrieve those which, form loose components, can ultimately lead to the final product which satisfies the constraints defined by Product Specifications. The output of Assembly Search is an Assembly Graph which will be subsequently used by Adaptive Operator Guidance. Assembly Search is composed of two sequential sub-processes, namely Monte Carlo Graph Search (MCGS) and Graph Completion (see Fig. 6).

Monte Carlo Graph Search is the core process of Assembly Search. It leverages the Virtual Environment to manipulate the components to determine whether a certain assembly action is feasible. MCGS' output is a forward-directed graph, that is, all its possible paths attempt to assemble the product (either correctly or wrongly). Subsequently, Graph Completion enriches the forward-directed graph to make it suitable for Adaptive Operator Guidance, e.g. by creating edges allowing partial product disassembly.

6.1 Monte Carlo Graph Search

MCSG builds upon Monte Carlo Tree Search (MCTS), a model-based reinforcement learning algorithm which showed its power especially in the Game Artificial Intelligence domain [54, 58]. The main difference between MCTS and MCGS is that — since MCGS' transition function is deterministic (see Section 6.1.1) — the latter relies on a node similarity function to determine whether its expansion tasks attempt to generate already existing nodes. In that case, the algorithm does not duplicate the nodes, rather it allows nodes to have more than one parent, hence creating a graph.

6.1.1 Markov Decision Problem

Each state encodes a unique assembly stage. The state is composed of a tuple representing three regions: *bucket*, *picked* and *assembled*. Each component, uniquely identified, can be in one and only one region at a time. The initial state is the only one in which all components are in the bucket. MCGS can perform four parametrised actions:

$$\begin{aligned}
&- \; first\ placement(id)\\
&- \; pick(id)\\
&- \; orient(x_{deg},\ y_{deg},\ z_{deg})\\
&- \; join(p_{id},\ p_g,\ a_{id},\ a_g)
\end{aligned} \tag{1}$$

First placement is the only action that can be performed in the initial state and is not performed anywhere else. Then, MCGS will execute the remaining three actions iteratively and in that order (pick, orient, join, pick, ...) unless a terminal state is reached.

Pick moves a component from bucket to picked. Orient sets a 3D orientation to the picked component. Join connects the picked component (p_{id}) with another already present in assembled (a_{id}), by specifying the two components' connection groups (p_g and a_g). The successful join action moves the picked object to assembled. Bucket solely contains component ids. Picked contains the picked component id, its available connection groups and, after the orient action, the set orientation. Assembled contains all the assembled ids, their orientations, all the join group pairs, and all the remaining available connection groups. First placement corresponds to the concatenation of $pick(id)$, $orient(0, 0, 0)$ and $join(id, \_, \_, \_)$, that is, join with nothing.

The reward function is depicted in Fig. 7. Given a state and action in input, the function returns a reward signal and determines whether the input state is terminal or not. If the action is not a join, the function returns the "no reward" signal and the current state is non-terminal. Otherwise, the function determines the signal by leveraging the Virtual Environment.

First, the Virtual Environment places p_{id}, with its specified orientation, in the 3D space so that its p_g overlaps a_{id}'s group a_g. Then, by following the reversed direction of the normal associated to p_g , the Virtual Environment sets p_{id} in motion, with constant speed, moving away from its starting position ("assemble and disassemble").

The component will move along a straight line. It can then either collide with another assembled component

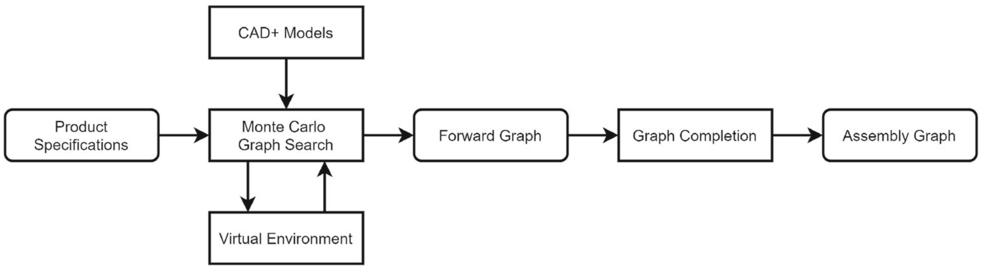


Fig. 6 The diagram representing the Assembly Search process

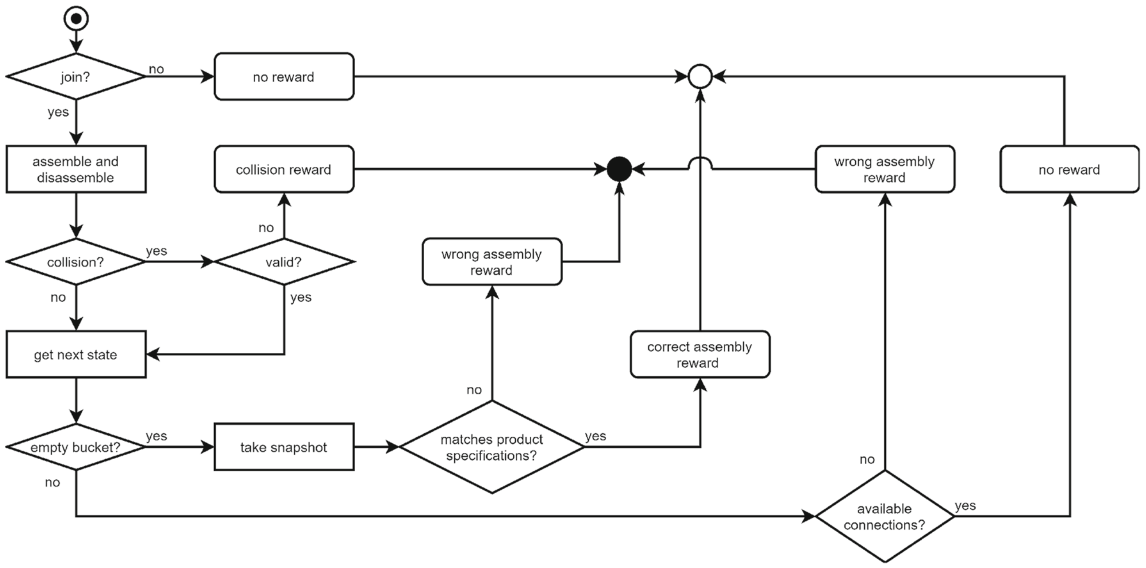


Fig. 7 MCGS' reward function. The concentric black-white circle corresponds to the current state the action is performed, that is the state the reward signal is calculated upon. The black circle, which can be reached only in case join is performed, means that the current state is also a terminal state (unsuccessful join action). The non-filled circles means that the current state is non-terminal (either a successful join or any other action)

or reach a pre-defined limit distance ("collision?"). If a collision is not "valid", that is, it does not occur between p_{id} and a_{id} in a limited region around p_g and a_g (possible, for instance, in case of unavoidable floating point approximations), then the function returns the "collision reward" signal and the input state is terminal (invalid join action).

Algorithm 1 State similarity function.

```
1 function $state\ similarity(state_1,\ state_2)$
2   if $leading\ action_1 \neq leading\ action_2$ then return $false$
3   if $bucket_1 \neq bucket_2$ then return $false$
4   if $picked_1 \neq picked_2$ then return $false$
5   if $assembled_1 \neq assembled_2$ then return $false$
6   if $picked\ orientation_1 \neq picked\ orientation_2$ then return $false$
7   if $available\ connections_1 \neq available\ connections_2$ then return $false$
8   if $connection\ pairs_1 \neq connection\ pairs_2$ then return $false$
9   return $true$
```

In case of no or valid collision, the Virtual Environment re-places p_{id} in its assembled position and retrieves the newly available connection groups ("get next state"). If the next state's bucket is not empty ("empty bucket?") and no available connection group remains ("available connections?"), then spare components are left behind. Therefore, the reward function returns the "wrong assembly" signal and the input state is terminal. If the bucket is not empty and there are available connections, then there are spare parts that can still be connected. Therefore, the returned signal will be "no reward" and the input state is non-terminal.

If the bucket is empty, then a full product is assembled. It is now time to evaluate whether it adheres the Product Specifications. The Virtual Environment generates a snapshot of the assembled product ("take snapshot") and this is compared — currently via Structural Similarity Index Measure (SSIM) [68] with window size w — with Product Specification's product partial look ("matches product specifications?"). If the similarity score is above a pre-defined threshold τ, then the virtually assembled product is considered adhering the specifications. This leads to a "correct assembly" signal, the input state is non-terminal and, intuitively, the next state is the terminal goal state. Otherwise, the function returns the "wrong assembly" signal and the input state is terminal.

Figure 8 presents a fragment of the transition and reward function for a fictional three-component product to be assembled. It depicts the initial state (S0, at the top) and two terminal states (S9 and S14). S9 is the result of a wrongly executed join action ("wrong assembly" reward signal), as the next state (S11, showed below the dashed line to indicate that it was solely generated to determine the reward signal) has a non-empty bucket and no available connections. S14, instead, is the result of a correctly executed join action.

The graph initially splits into three branches, one for each component in the bucket. It is also possible to see that the first two branches eventually merge at S10,

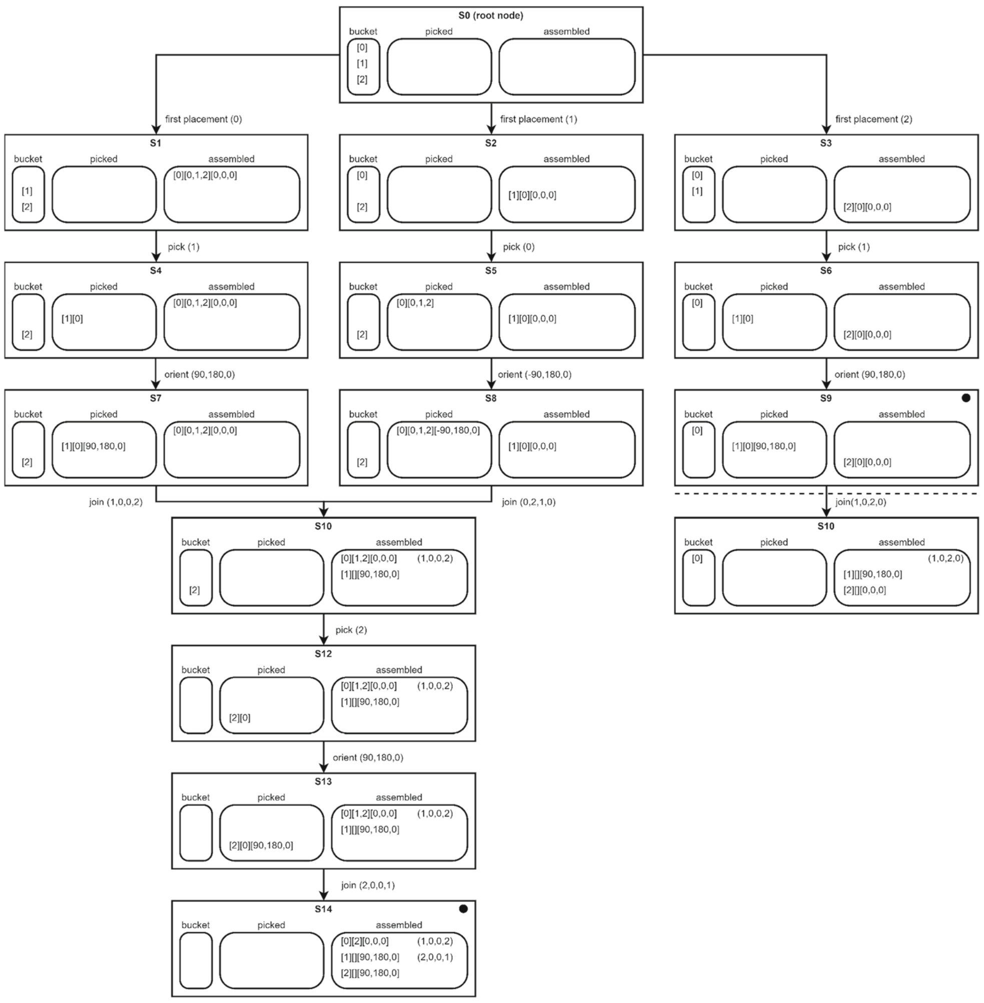


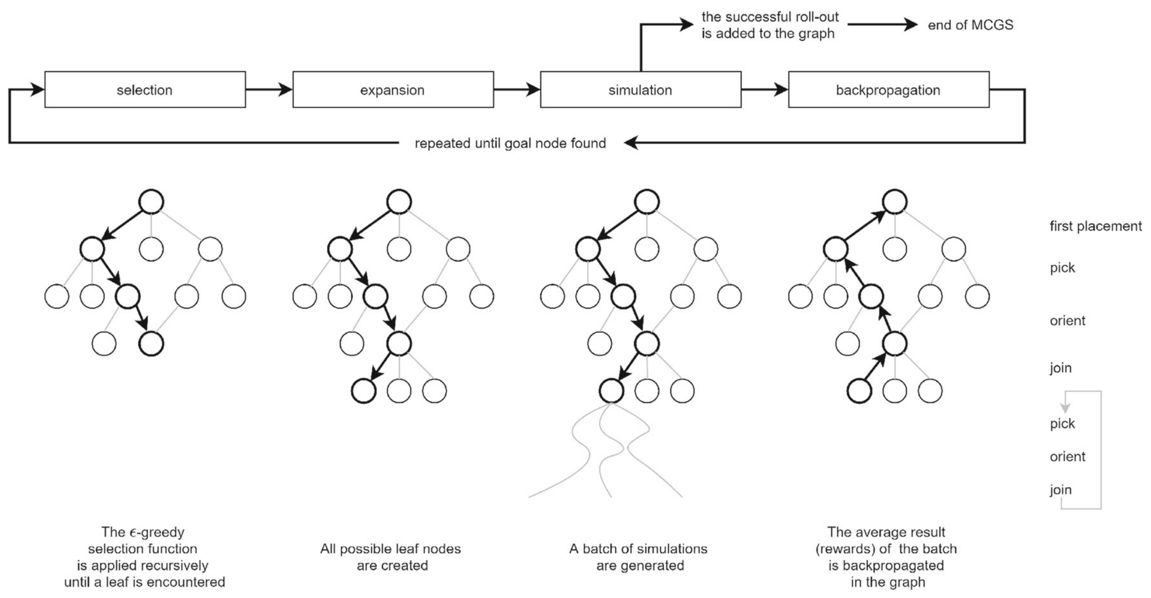
Fig. 8 An example of transition function and related graph in accordance with Section 6.1.1. Each node, depicted by a rectangle, represents a state. Each state label has prefix "S" followed by a number. Fifteen states are depicted. States with a solid black circle on their upper right corner represent terminal states for the Monte Carlo Graph Search algorithm. S10, depicted below a dashed line, represents a state which was only estimated by the reward function: it is part of the transition function but the algorithm would never reach this state. Each state defines the three bucket, picked and assembled regions. Each region maintains the relevant information of each component: id for the bucket region; id and, if specified, available connection groups and orientation for the picked region; id, available connection groups and orientation for the assembled region. The component information is encapsulated in square brackets and is represented as a horizontal vector. The assembled region also holds a list of component id-connection group pairs encapsulated by round brackets. Each edge connecting two nodes represents an action. Each action label specifies its parameters in accordance with the action space defined in Eq. 1. Some details have been omitted in order not to overcomplicate the figure

after their respective join actions. This is achieved by means of the state similarity function, which pseudo-code can be found in Algorithm 1. The function is rather straightforward: two states (i.e. the next states of S7 and S8) are equal if their regions and the parameters they hold are the same. The so-called leading action is the action that generated the function's input states (join in this example).

As it can also be seen, the orientation check is done only for components in the picked region, as the assembled ones might be in different orientations depending on the order of the assembly, yet represent the same partially assembled product. What is important to check for the assembled region, however, is that the available connections and the assembled connection pairs are the same.

6.1.2 MCGS algorithm

The MCGS algorithm — the schematic representation of which can be found in Fig. 9 — performs its iterations and generates the Forwad Graph. Each node is composed of the state and its *value*, computed and updated by the algorithm's backpropagation step. The edges correspond to the parameterised actions. MCGS is executed only once,



the successful roll-out is added to the graph
end of MCGS
selection
expansion
simulation
backpropagation
repeated until goal node found
first placement
pick
orient
join
pick
orient
join
The ϵ-greedy selection function is applied recursively until a leaf is encountered
All possible leaf nodes are created
A batch of simulations are generated
The average result (rewards) of the batch is backpropagated in the graph

Fig. 9 The schematic of Monte Carlo Graph Search

with each iteration beginning at the root node (initial state). The algorithm terminates when the terminal goal state/node is found.

As MCGS operates in a deterministic world, the canonical Upper Confidence Bound (UCB) selection algorithm can lead to a nearly exhaustive search strategy. Therefore, MCGS' selection step relies on the ϵ-greedy policy. The selection phase terminates when a leaf node, i.e. a node with no outgoing edges, is encountered.

Once a leaf node is reached, the subsequent expansion phase occurs. Expansion generates nodes based on the action to be performed. If it is a pick or first placement, Expansion generates as many nodes as the components in the bucket. If the action is orient, the expansion strategy generates nodes which picked orientation is such that any of its connection group normals are inverted with any available and compatible (i.e. same connection type) group in the assembled region. In other words, orient-centred expansion generates nodes which always lead to a feasible (yet collision ignorant) join action. Finally, the expansion for join generates as many nodes as the compatible p_g-a_g groups for each p_{id}-a_{id} pair.

Once the nodes are expanded, a random one is selected uniformly, and from there the Simulation step starts. Simulation generates s roll-outs by (1) adhering to the ordered and iterated pick-orient-join sequence, (2) generating nodes uniformly sampled from the feasible ones (i.e. Simulation utilises the same rationale used by Expansion). The s roll-outs $O = \{o_1, o_2, \ldots o_s\}$ would then be executed to retrieve their respective reward signals $r_i \; \forall \, o_i \in O$.

As soon as a roll-out $o_i \in O$ receives a "correct assembly" signal, o_i is added to the graph and MCGS terminates. Otherwise, the final step of MCGS, backpropagation, occurs. First, the gain G of O is calculated as the related rewards' arithmetic average

$$G = \frac{1}{s} \sum_{i=1}^{s} r_i$$

backpropagation is then performed via every-visit Monte Carlo update rule [58]

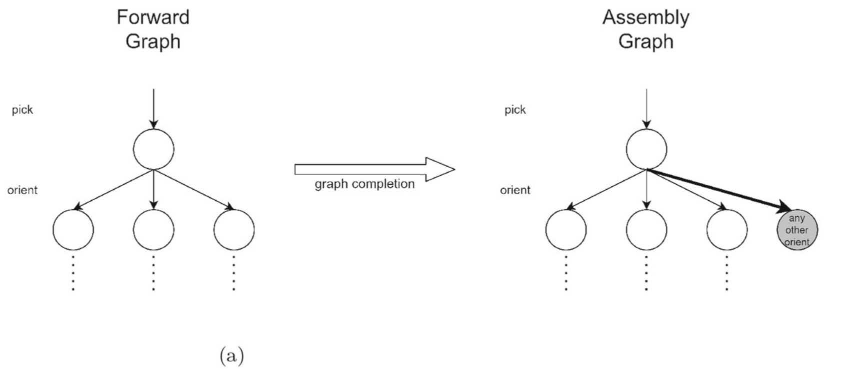
$$V_i \leftarrow V_i + \alpha(G - V_i)$$

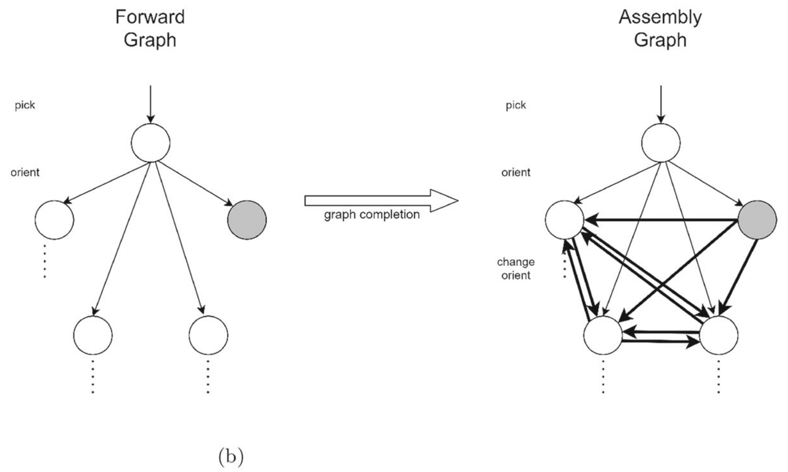
by discounting G, for each backward step, by a parameter γ

$$G \leftarrow \gamma G$$

6.2 Graph completion

The Forward Graph, output of MCGS, is further augmented in three simple steps. The first one regards the creation of an additional orientation node, which symbolises *any other orientation*, to the same level as those generated after the orient action (see Fig. 10a). The second augmentation creates *change orientation* edges (see Fig. 10b). The edges connect orientation nodes of the same level with each other, to create a nearly complete sub-graph. *Any other orientation* is the only node which does not have any incoming *change orientation* edges. The third augmentation adds *undo pick* edges from all nodes in between pick actions (see Fig. 10c). Undo pick reverts the first placement edge as well.





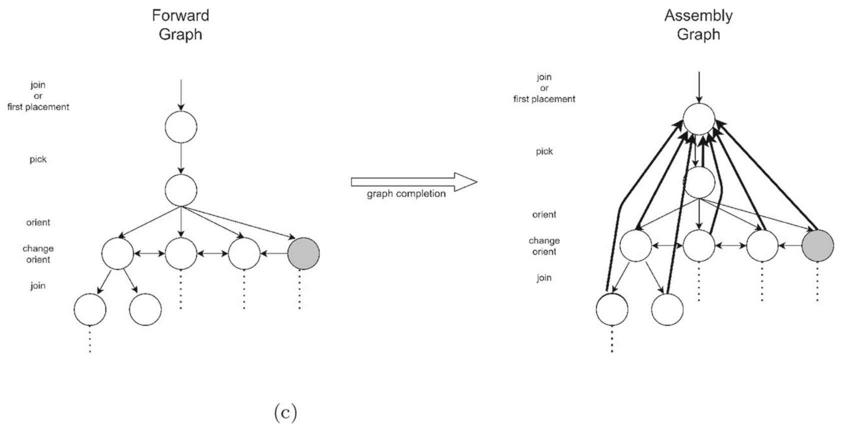


Fig. 10 The three steps of Graph Completion. (a) “Any other orientation” node creation. (b) “Change orientation” edge creation. (c) “Undo pick” edge creation

7 Assembly stage detector and its preparation

The Adaptive Operator Guidance process aims to autonomously detect the current assembly stage in order to provide the most relevant instruction to the human operator. The process relies on a module — Assembly Stage Detector — to observe the assembly theatre and to retrieve the necessary information required to map the physical assembly stage with the nodes of Assembly Graph.

Assembly Stage Detector is configured in three submodules (see Fig. 11). The first, Position Detector, identifies the components and locates them with respect to the state’s bucket/picked/assembled. The other two, Feature Extractor and Orientation Feature Dataset, retrieve the orientation of the picked component by extracting its visual features

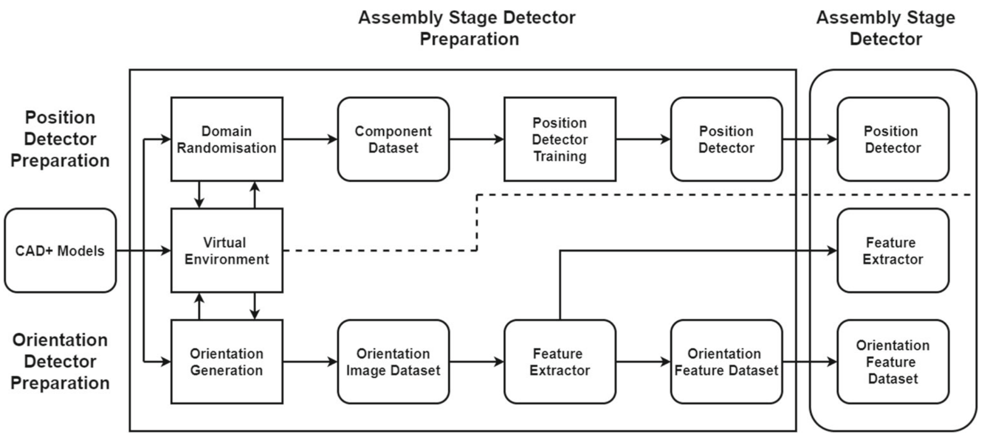

Assembly Stage Detector Preparation
Assembly Stage Detector
Position Detector Preparation
CAD+ Models
Orientation Detector Preparation
Domain Randomisation
Virtual Environment
Orientation Generation
Component Dataset
Position Detector Training
Position Detector
Orientation Image Dataset
Feature Extractor
Orientation Feature Dataset
Position Detector
Feature Extractor
Orientation Feature Dataset

Fig. 11 The detailed schematics of ViTroVo's Assembly Stage Detector Preparation process

and mapping them with a previously generated orientation-visual feature dataset.

7.1 Position detector preparation

The Position Detector is realised using a RetinaNet object detection network [34] with a ResNet-50 backbone [19], as implemented by [14]. The network, pre-trained on the COCO dataset [35], was further trained using a Component Dataset of synthetic images with the Adam optimisation algorithm [29].

Component Dataset comprises of S synthetic images generated in ViTroVo's Virtual Environment via domain randomisation [62, 63]. The Virtual Environment generates random 3D scenes composed of three elements: a background plane (simulating the assembly table), assembly components, and distractor objects. Scenes are generated by uniformly sampling between $[obj_{min}, obj_{max}]$ objects (components or distractors) which are then positioned in the scene by randomising their position, orientation, scale, colour and texture. The distractors items can either be a cube, a cylinder, a sphere or a capsule. The texture is uniformly sampled from a set of t textures. A single image of size $h \times w$ was captured from every scene, with accompanying meta-data including the type and 2D bounding box of every component. Figure 12 showcases five examples of such generated images.

7.2 Orientation detector preparation

Orientation detection is performed by means of an image retrieval methodology.

First, an Orientation Image Dataset of images depicting isolated components with different orientations is generated using the Virtual Environment. A total of 64 images per component are generated — all combinations of 0, 90, 180 and 270° of rotation around its three axes. However, components can be symmetrical over certain axes, leading to the same image. Therefore, highly similar images (detected via structural similarity) are detected and grouped together. Figure 13 depicts the generated asymmetrical images for our case study's LED board component.

Subsequently, the images are fed to a pre-trained VGG-16 neural network (excluding the last layer) which acts as a Feature Extractor [55]. The result is an Orientation Feature Dataset.

Finally, during actual assembly, the orientation of a picked component is determined by the following steps: (1) a cropped image of the picked component is obtained using the bounding box derived by the Position Detector, (2) the cropped image is fed to the VGG-16 network (Feature Extractor) to retrieve the related feature vector, (3) the Euclidean distance of the picked component's feature vector with those stored in the Orientation Feature Dataset is calculated, (4) the orientation related to the

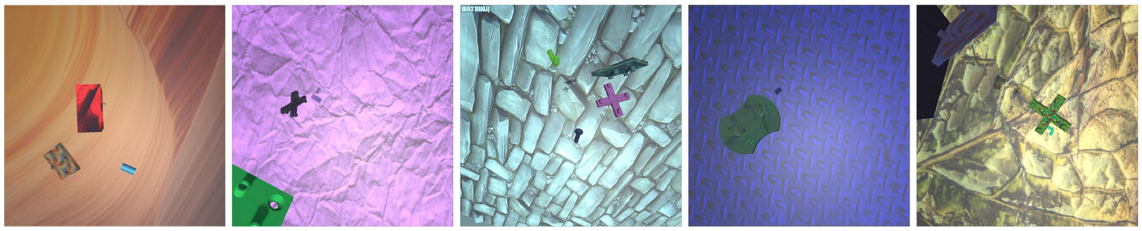


Fig. 12 Examples of domain randomised images used to perform the supervised learning of Assembly Stage Detector's Component Detector sub-module

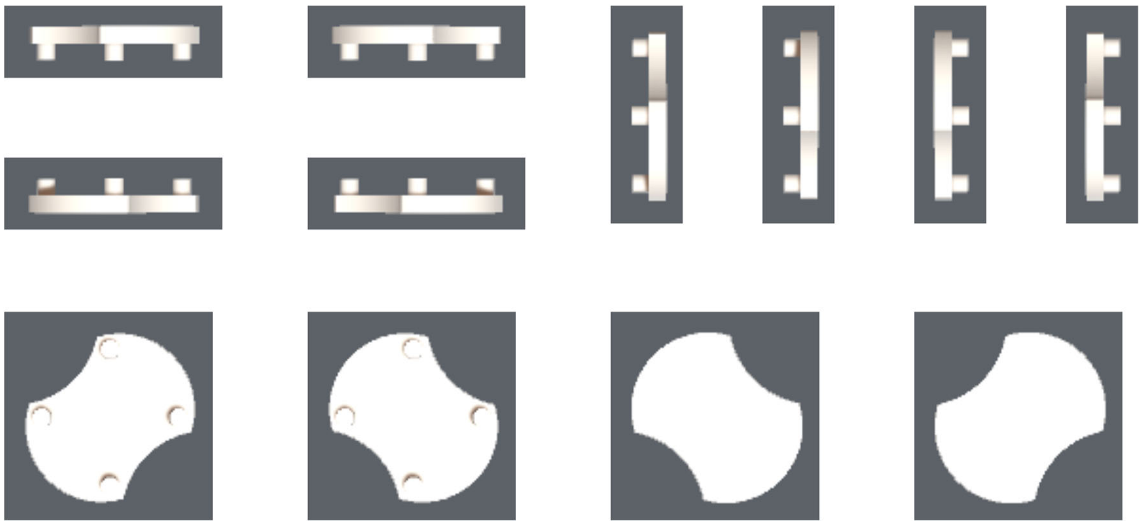


Fig. 13 An example of the 12-out-of-64 symmetry-free orientation images related to one of the LED board component of our case study

dataset's vector with closest distance is returned as detected orientation.

8 Adaptive Operator Guidance

The in vivo Adaptive Operator Guidance (AOG) is the final aggregator process of ViTroVo. In here, the framework and the human operator interact with each other so that the physically manufactured components are combined to form the end-user's idealised product. Figure 14 depicts the schematics of AOG. Two information flows exist: one going from the Assembly Graph to the Operator (ViTroVo-to-Operator), and one going in the reversed direction (Operator-to-ViTroVo). The latter is in charge of making sure AOG *knows* not only at which assembly stage the operator is in, but also how the operator has thus far experienced the task at hand. The former presents the most relevant assembly instruction to be executed.

Algorithm 2 Graph localisation.

```
1  function $graph\ localisation(\mathcal{G},\ n,\ P,\ \hat{s})$
2    $node\ list\ =\ breadth\ first\ search(\mathcal{G},\ n)$
3    $candidate\ nodes\ =\ \emptyset$
4    for each $node\ \in\ node\ list$ do
5      if $state\ similarity(node.state,\ \hat{s})\ is\ true$ then
6        $candidate\ nodes \leftarrow concatenate(candidate\_nodes,$
           $node)$
7      end if
8    loop
9    if $|candidate\ nodes|\ =\ 0$ then return $n$
10   if $\exists\, m\ \in\ candidate\ nodes\ :\ m \in P$ then return first $m$
11   return $candidate\ nodes[0]$
```

8.1 Operator-to-ViTroVo

The Graph Localisation process, in charge of keeping the Assembly Graph's current node and the physical assembly stage aligned, takes in input both the processing outcome of Assembly Stage Detector (ASD) and, if existing, Operator Input.

A snapshot of the physical assembly theatre is fed to ASD. The Position Detector network returns the identified components and their bounding box. Based on pre-defined specifications of the in vivo bucked, picked and assembled regions, the corresponding in vitro component positioning is retrieved. In case of a picked component, ASD retrieves the orientation of the object.

The ASD-driven Graph Localisation process is outlined in Algorithm 2. It takes in input the Assembly Graph $\mathcal{G}$, the current node n, the current path from n to goal node P, and Assembly Stage Detector's output $\hat{s}$. The algorithm is designed based on a somewhat fair assumption: the operator is in good faith, i.e. she tries to execute the instructions to the best of her capabilities. The algorithm first sorts the nodes via breadth-first search (line 2) starting from n. It then builds a list of candidate nodes which state matches $\hat{s}$ (lines 3–8). The matching is achieved via the *state similarity* Algorithm 1 introduced in Section 6.1.1 (line 5). The list of candidate nodes is then analysed (lines 9–11). If the list is empty then no candidate node is found, hence the last known current node is prudently returned n (line 9). If the list contains nodes that are also in P then the first of these, i.e. the node closest to n (result of breadth-first search), is returned (line 10). Otherwise, the algorithm returns the first candidate node (line 11).

The ASD-based Localisation process is not infallible. For this reason, the Operator-to-ViTroVo flow allows the

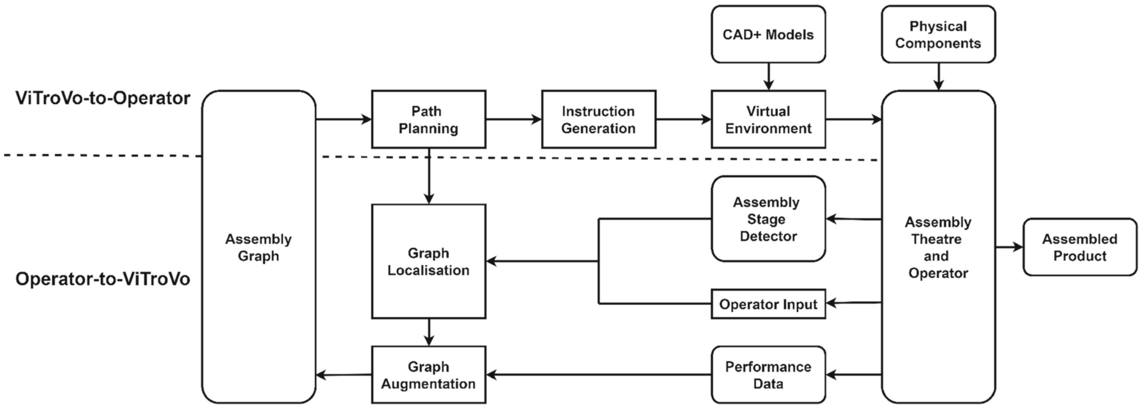


Fig. 14 The details of ViTroVo's in vivo Adaptive Operator Guidance process

operator to proactively retrieve the next/previous path-planned instruction via Graphical User Interface (Operator Input). The Graph Localisation process based on Operator Input is straightforward.

Ultimately, the Operator-to-ViTroVo flow gathers additional Performance Data and stores it along the edge connecting the current node with the localised one. Currently, Adaptive Operator Support records the duration of the executed instruction and the operator's average emotional state (happiness, sadness, anger and neutrality). Emotion detection is achieved by means of the pre-trained, mini-Xception network [3, 47].

8.2 ViTroVo-to-Operator

The flow is organised as a simple sequence of processes. First, Path Planning determines the sequence of nodes and edges from the (localised) current node to goal node. As the edges are enriched (weighed) with performance data, the process relies on the Bellman-Ford algorithm [4] to generate, e.g., shortest paths, minimum duration paths, sadness-minimum paths, and so on. Subsequently, Instruction Generation extracts the relevant information of the triplet current node, next node and connecting edge and sends it to the Virtual Environment. The latter, finally, utilises the related CAD+ models and additional information (e.g. connection medium for the join action) to visualise the animated instruction. A snapshot of a rendered join instruction can be seen in Fig. 15.

9 Initial evaluation

The five-component fictional Product Specifications introduced in Section 4 was fed to the two in vitro processes Assembly Search and Assembly Stage Detector Preparation. The hyperparameters of their respective sub-processes can be found in Table 2.

Assembly Search ran on a non-dedicated Intel® Core™ i7-9850H CPU (2.60GHz, 6 Cores, 12 Logical Processors) with 32GB. It terminated in approximately five hours and generated a Forward Graph composed of 2062 nodes and 2106 edges. The subsequent Graph Completion process led to an Assembly Graph of 2306 nodes and 12426 edges. Table 3 reports graph sizes throughout Graph Completion, whilst Fig. 17 gives a macroscopic view of the two Assembly Search graphs.

We then conducted a preliminary, qualitative evaluation aimed to estimate the effectiveness of Adaptive Operator Guidance. The process ran on the same computing machine used by Assembly Search. We set an assembly theatre composed of (1) a desk, with the three bucket/picked/assembled regions, (2) one monitor displaying the animated instructions, (3) one monitor displaying additional information (e.g. detected emotion values) and the two previous/next instruction buttons (Operator Input), (3) a Logitech C920 PRO HD webcam (1080p), mounted at the middle of the desk and 50cm elevated from it, which stream is sent to Assembly Stage Detector, (4) a LED strip, along the frame

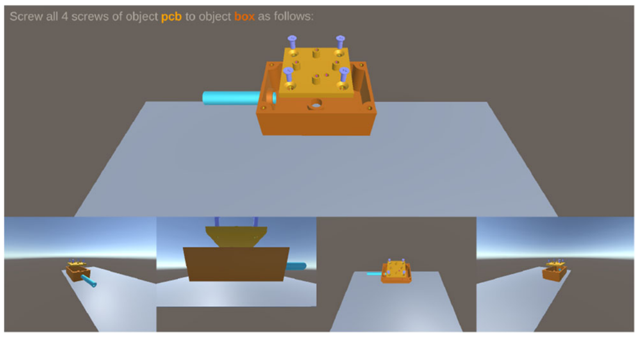


Fig. 15 An example of generated join instruction. The information related to the connection medium is used by the Virtual Environment to also render the number of needed screws and their position. The animation highlights the components to be connected (in orange), moves the PCB to its target position, and also shows how screws should be fastened (clockwise rotation)

Table 2 Hyperparameter settings used for the five-component case study

Name	Description	Value	Location
no reward	Reward signal	0	Reward Function
collision reward		–1	Reward Function
wrong assembly reward		–0.1	Reward Function
correct assembly reward		10	Reward Function
w	SSIM window size for correct assembly reward signal	351	Reward Function
τ	Threshold for correct assembly reward signal	0.8	Reward Function
ϵ	ϵ-greedy exploration policy	0.1	MCGS algorithm
s	Number of roll-outs	10	MCGS algorithm
α	Every-visit Monte Carlo update rule	0.1	MCGS algorithm
γ	Gain discount rate	0.99	MCGS algorithm
b	Batch size	4	Position Detector Training
e	Learning epochs	100	Position Detector Training
r	Learning rate	1e−5	Position Detector Training
S	Number of synthetic images	15.371	Component Dataset
$[obj_{min},\ obj_{max}]$	Number of components/distractors per synthetic image	[10, 50]	Component Dataset
t	Number of textures	50	Component Dataset
$h \times w$	Synthetic image size	720 × 720	Component Dataset

holding the 1080p camera, to better illuminate the assembly stage, and (5) a Logitech C170 (1024x768) webcam, positioned in front of the operator, to record her affective state (see Fig. 16). The stage was installed at the Brainport Industry Campus in Eindhoven, The Netherlands [21]. We recruited three participants in September 2020. Unfortunately, the restrictions due to the COVID-19 pandemic are still hampering our ambition to conduct a thorough empirical evaluation.

The participants did not know how the final product would have had to look like and never saw the components before. As soon as the experimenter positioned all components in the bucket region, they began to follow the given instructions. They all successfully assembled the final product. They found the animations easy to understand. None of them paid attention to the instruction's additional text. They all agree that the set orientation instruction is needless, as that information is also shown during the subsequent join instruction. Assembly Stage Detector had difficulties in retrieving the orientation of the picked stand and, for one participant, the picked antenna as well. In these circumstances, the two operators pressed the "next instruction" button. Possibly, the main reason is due to the rather low-quality of the gathered images (see Fig. 4). The captured components were very blurred, to the point that the stand threading — the only feature capable to suggest an orientation — is hardly visible even for a human eye. On the other hand, component detection and its region mapping was highly successful. Graph Localisation correctly redirected a participant who erroneously picked the wrong component (the undo pick instruction was shown).

Aside for the previous/next instruction buttons, the participants showed no interest in the monitor displaying

Table 3 Forward-to-assembly graph number of nodes and edges progress generated by Graph Completion

Graph	Completion Step	Number of	
		Nodes	Edges
Forward Graph		2062	2106
	"Any other orientation" node creation	2306	2350
	"Change orientation" edge creation	2306	10922
	"Undo pick" edge creation	2306	12426
Assembly graph		2306	12426

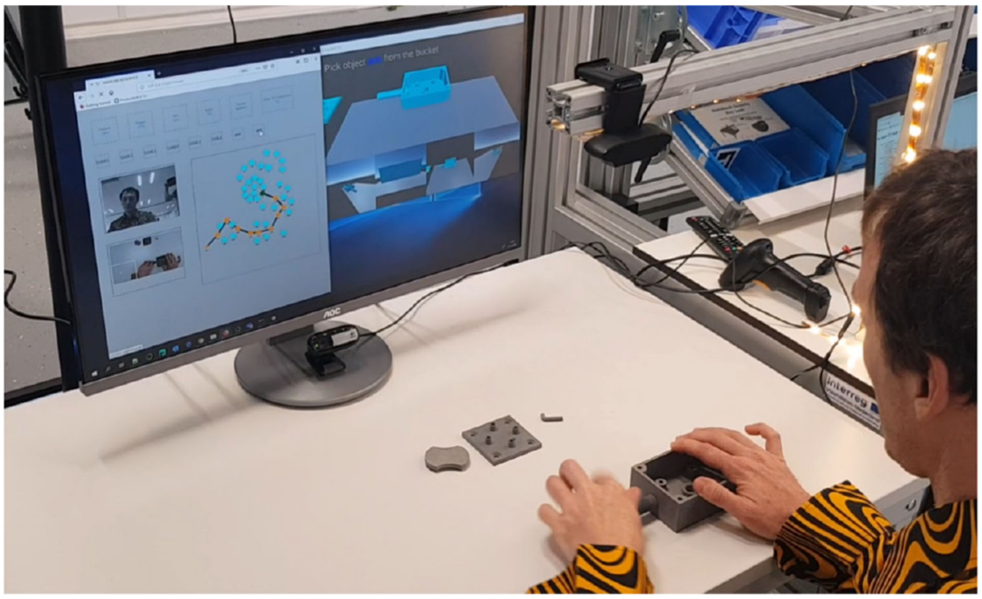


Fig. 16 A snapshot of the ViTroVo assembly theatre

additional information. Ironically, two participants thought the monitor was solely needed by the experimented for debugging purposes, given its rather poor graphical appeal (Fig. 17).

10 Discussion, limitations and future work

ViTroVo's architecture (see Fig. 1) was designed to tackle the three research questions introduced in Section 1.

RQ1 was successfully tackled by the framework's in vitro Assembly Search module (see Section 6). More specifically, Monte Carlo Graph Search generates a Forward Graph from loose components to final product by avoiding the preliminary disassembly step.

RQ2 was tackled by a plethora of modules. Undoubtedly, the in vivo Adaptive Operator Guidance module (see Section 8) is the core component in charge of realising the interaction with the operator. Its design into a looped information flow (ViTroVo-to-Operator and Operator-to-ViTroVo) allow for a continuous monitoring of the operator's instruction executions, assembly status localisation within the graph, and hence retrieval of the next, most appropriate instruction. Adaptive Operator Guidance cannot exist without Assembly Graph, output of Assembly Search. Additionally, the module rather successfully relied

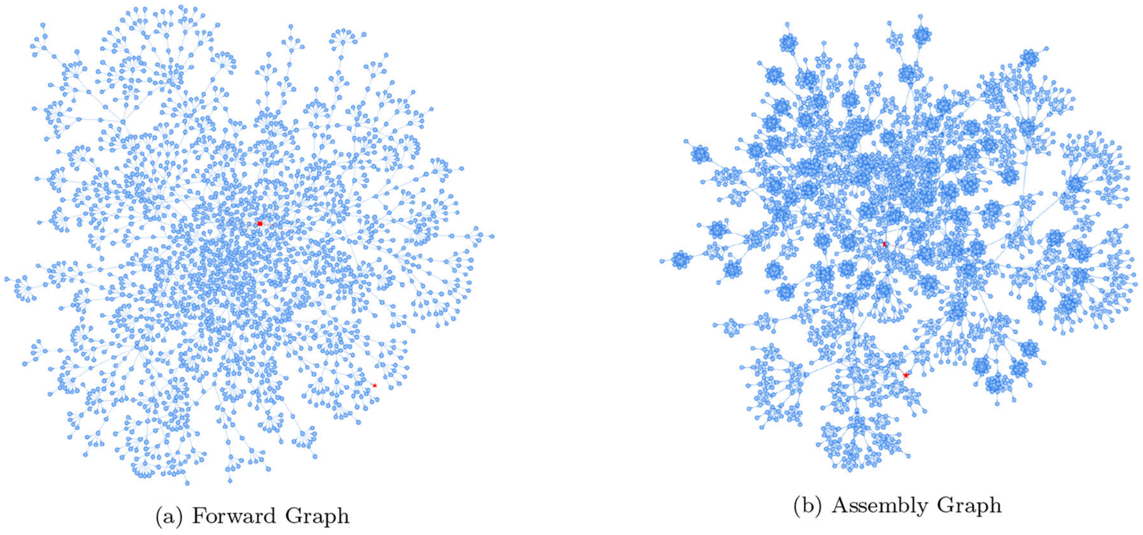


Fig. 17 The two graphs generated by Assembly Search for our five-component case study. The root node is depicted by a red square, the goal node by a red star

on Assembly Stage Detector, a vision-based component capable of *understanding* the current assembly status and hence obtain an automated Graph Localisation task. Most remarkably, Assembly Stage Detector was trained in vitro via domain randomisation (see Section 7), that is, by means of synthetically generated images of the product components.

RQ3 was tackled by equipping Adaptive Operator Guidance with (1) a vision-based emotion recognition module, part of Performance Data gathering (Operator-to-ViTroVo flow), (2) Graph Augmentation, which stores the detected emotion values as weights on the Assembly Graph's traversed edges (Operator-to-ViTroVo flow), and finally (3) Path Planning (ViTroVo-to-Operator flow) which, by relying on the Bellman-Ford algorithm, computes the shortest path in a weighted graph. In this way, ViTroVo can generate paths which, for instance, anger is minimised or happiness is maximised (the happiness values would be stored with their sign inverted).

Ultimately these modules, together with Product Customisation and Component Manufacturing, are effortlessly linked together by a versatile component representation — CAD+ models — and management — a Virtual Environment — used throughout the whole highly customised manufacturing journey, albeit with slightly different purposes and configurations (see Section 5). We advocate the standardisation of CAD+ models for Highly Customised Manufacturing to seamlessly transition from in vitro to in vivo phases. The standardisation must enable the automated augmentation which, in our study, was necessarily done by hand, such as the creation of collider shapes.

We are convinced ViTroVo paves the way for Highly Customised Manufacturing. Nevertheless, it can certainly be improved. What follows is a scrutiny of its performance, component per component, with the aim to identify potential advancements.

Assembly Search's greatest challenge lies in effectively explore the combinatorial space of assembly configurations by solely relying on Product Specification's partial information, in our case study corresponding to the outer look of the final product.

The decision to model Assembly Search as a Markov Decision Process was a straightforward interpretation of its required output, the Assembly Graph: nodes would correspond to the process' states, edges to its actions, and the certain outcome of a specific action performed in a specific state would correspond to a deterministic transition. It was therefore natural to rely on reinforcement learning to algorithmically solve the related Markov Decision Problem (MDP). Reinforcement learning allows Assembly Search not only to seek for the optimal action plan to assemble the product, it also allows to explore the search space, to discover wrong assembly sequences, hence to generate a slightly redundant assembly graph, which would arguably be much more prone to localise the physical assembly stage in case of erroneously executed instructions. Within the model-based reinforcement learning approaches, the choice to use Monte Carlo Graph Search was mainly motivated by the need to solve a specific MDP related to a specific product, rather than training an AI agent capable of assembling *any* product. Moreover, the reward function in use allows for early stops along assembly paths in case of collision and wrong assembly reward signals (see Section 6.1.1 and Fig. 7). We argue that the Markov Decision Process so-defined is quintessential to Assembly Search, whilst MCGS could potentially be replaced by other search algorithms, for instance Evolutionary Computation or Random Forest Search, as the designed reward function can easily be used as a fitness/cost function. Future work on such comparisons are expected.

The case study we used to validate our approach was mainly based on the idea that Highly Customised Manufacturing would leverage global repositories of interchangeable components. The other fundamental characteristic, that is the need to assemble unique products composed of customer-designed parts, was indirectly considered in this work. We are quite confident our framework is rather robust against customer-designed parts, providing of course these are specified via CAD+ models, as Assembly Search was designed to solely leverage the product components and the partial knowledge over the final product. Moreover, the in vitro phase follows the Product Customisation process. It is in this very process that ViTroVo would bring constraints to the customer regarding her custom-designed parts, among which is the specification of connection types and medium.

We believe the main causes behind the apparent long duration of Assembly Search execution (five hours) are not to be attributed to MCGS itself. First and foremost, the process was run on a non-dedicated and, arguably, non-high-end hardware. Furthermore, most of the execution time spent during MCGS corresponded to the "assemble and disassembly" step performed by the Virtual Environment during reward signal calculation. It is more than likely that headless simulations would decrease the computational time. Our choice for the Unity Software Suite was mainly based on the ease to create template-based animated instructions. Future investigations on other simulators, e.g. Gazebo [15] or Unreal Engine [11] are due. Another straightforward improvement would come from the parallel execution of multiple MCGS algorithms which would nonetheless operate on the same assembly graph. Aside all that, the execution of MCGS led to a rather large assembly graph, which of course required computational time. This was essentially due the lack of knowledge related to intermediate assembly steps. The reward function returns meaningful signals only in case of collision or when all components are somehow assembled together. Partial assembly information should be retrieved by the preliminary Product Specification process. The MCGS algorithm was designed to monolithically assemble a product by starting from its root node in each iterations. This search strategy would inevitably face scaling issues in presence of a higher number of components/connection points. A way to overcome such issue is combining MCGS with divide-et-impera approaches such as Group Genetic Algorithm [13], which would lead to (genetic-driven) Assembly Graphs of (MCGS-generated) sub-assembly graphs.

In addition to improve the computational performance and scalability of Assembly Search, future work would centre its attention at converting the Assembly Graph into an AND/OR graph — most likely a straightforward task when MCGS is combined with divide-et-impera search approaches —, to make the output of Assembly Search compatible with, e.g., Assembly Line Balancing and teamed human-robot assembly. Assembly Search should also allow the integration of domain knowledge (e.g. assembly heavier components first, safety/functional assessment once electrical components are combined). The necessary requirement of MCGS is a model-based, deterministic Markov Decision Process. This does not impede to grant higher degrees of agency to the Virtual Environment. For instance, and in accordance with the related work on motion planning, the Virtual Environment could autonomously retrieve non-straight join paths, allowing thus the search for more complicated products to be assembled.

The main limitation of our framework are Assembly Stage Detector's Orientation Detector and Graph Localisation. Although we note that these were impacted by poor image quality — see Fig. 4c and e — it appears evident that orientation detection should be improved. On the other hand, the Domain Randomisation approach resulted in a Position Detector model which produced highly accurate and robust detections, despite it being trained on solely synthetic data without any real-world examples. The Assembly Stage Detector-Graph Localisation pair would most likely benefit from (1) an additional detector capable to recognise which action is being performed and (2) a form of memory that remembers the visited nodes, so that the good faith assumption can be better leveraged. Finally, Graph Localisation should cease to be greedy. The memory could in fact allow for a probabilistic localisation process. This process could even increase the cooperation with the operator, for instance, by letting her ultimately select the current assembly stage, should more than one candidate (highly likely) node be found.

In our current architecture, Performance Data is used to augment the Assembly Graph. Future work must delve into the task of operator modelling. This could allow, for instance, to determine the level of expertise of the operator and to predict which future actions might negatively affect her state.

The Instruction Generation process is not free from improvements too. As we understood in our pilot study, there is little-or-no need to present orient instructions. Hence, and in accordance with what said regarding operator modelling, further work on Instruction Generation should delve, e.g., into experience-tailored [2], multi-modal instructions [32], or even gamification [36].

To conclude, we introduced the ViTroVo architecture which shows to be highly promising in becoming one of the foundations for Highly Customised Manufacturing. ViTroVo brings a plethora of new possibilities in the smart manufacturing domain, such as global repositories of compatible/interchangeable components, but also repositories of assembly sub-graphs and therefore reusable partial instructions. Possibly, the framework could become an online service, capable to retrieve assembly instructions of *any* customised product and directly for the customer, whilst individual companies physically manufacture and independently deliver the required components. Additionally, ViTroVo could be used as an educational tool: the framework would leverage the operator model to generate fictional products that would force operators improve their skill repertoire by, e.g., performing assembly instructions which are historically known to be problematic.

11 Conclusions

Highly Customised Manufacturing (HCM) is a topic of growing importance within the smart manufacturing domain. If on one side HCM has the potential to establish new manufacturing patterns and closer relations between customers and companies, on the other side it poses challenges to the manufacturing task, such as the increase in flexibility of the assembly theatre, the compilation of assembly instructions for possibly unique products, and stress-related risks for human operators.

We introduced ViTroVo, a framework capable of (1) autonomously generating a graph of assembly steps without full knowledge on the final product and (2) adaptively presenting assembly instructions to a human operator. Based on a five-component product case study, our framework not only succeeded in retrieving the assembly steps, it also guided, in an initial empirical evaluation, three recruited humans throughout the assembly task. Although we are well aware of the improvements our framework needs and what future developments it can undergo, we believe the architecture of ViTroVo could constitute the foundations of highly customised manufacturing.

Acknowledgements The work could have not been possible without the cooperation and support of the authors' colleagues Peter Laloli, Ioannis Tollios, Mauro Comi and Marit Baalbergen. The authors thank their colleagues Gu van Rhijn and Tim Bosch (and Peter, twice) for having taken part into the empirical evaluation despite the COVID-19 threats, and Productive 4.0's consortium partner Signify.

Funding The work was partly funded by the Electronic Components and Systems for European Leadership's (ECSEL's) Productive 4.0 project [22] and the "Toeslag voor Topconsortia voor Kennis en Innovatie (TKI)" of the Ministry of Economic Affairs of The Netherlands [48].

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Alcácer V, Cruz-Machado V (2019) Scanning the industry 4.0: A literature review on technologies for manufacturing systems. Eng Sci Technol Int J 22(3):899–919
2. Antifakos S, Michahelles F, Schiele B (2002) Proactive instructions for furniture assembly. In: International conference on ubiquitous computing. Springer, pp 351–360
3. Arriaga O, Valdenegro-Toro M, Plöger P (2017) Real-time convolutional neural networks for emotion and gender classification. arXiv:1710.07557
4. Bang-Jensen J, Gutin GZ (2008) Digraphs: theory, algorithms and applications. Springer Science & Business Media
5. Borges (2021) Borges Website. https://www.borges.xyz/. Last accessed 9 August 2021
6. Breque M, De Nul L, Petridis A (2021) Industry 5.0: towards a sustainable, human-centric and resilient european industry. LU: European Commission, Directorate-General for Research and Innovation, Luxembourg
7. Browne J, Dubois D, Rathmill K, Sethi SP, Stecke KE et al (1984) Classification of flexible manufacturing systems. FMS Mag 2(2):114–117
8. De Mello LSH, Sanderson AC (1990) And or graph representation of assembly plans
9. Doyle-Kent M, Kopacek P (2019) Industry 5.0: Is the manufacturing industry on the cusp of a new revolution? In: Proceedings of the international symposium for production research 2019. Springer, pp. 432–441
10. ElMaraghy HA (2005) Flexible and reconfigurable manufacturing systems paradigms. Int J Flex Manuf Syst 17(4):261–276
11. Epic Games I (2021) Unreal engine. https://www.unrealengine.com/en-US/. Last accessed 9 August 2021
12. Evjemo LD, Gjerstad T, Grøtli EI, Sziebig G (2020) Trends in smart manufacturing: Role of humans and industrial robots in smart factories. Current Robot Reports 1(2):35–41
13. Falkenauer E (1998) Genetic algorithms and grouping problems. Wiley, Hoboken
14. Fizyr (2021) Retinanet network repository. https://github.com/fizyr/keras-retinanet. Last accessed 9 August 2021
15. Open Robotics (2021) Gazebo simulator. http://gazebosim.org/. Last accessed 9 August 2021
16. Funk M, Dingler T, Cooper J, Schmidt A (2015) Stop helping me-i'm bored! why assembly assistance needs to be adaptive. In: Adjunct proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing and proceedings of the 2015 ACM international symposium on wearable computers, pp 1269–1273
17. Gjeldum N, Salah B, Aljinovic A, Khan S (2020) Utilization of industry 4.0 related equipment in assembly line balancing procedure. Processes 8(7):864
18. Guo X, Zhou M, Abusorrah A, Alsokhiry F, Sedraoui K (2020) Disassembly sequence planning: a survey IEEE/CAA Journal of Automatica Sinica
19. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR), vol 2016-Decem. IEEE, pp 770–778. https://doi.org/10.1109/CVPR.2016.90
20. Hořejší P, Novikov K, Šimon M (2020) A smart factory in a smart city: Virtual and augmented reality in a smart assembly line. IEEE Access 8:94330–94340
21. Industries B (2020) Brainport industries campus. https://www.brainportindustriescampus.com/en/. Last accessed 31 December 2020
22. Infineon Technologies AG (Coordination Office, 2020) Productive 4.0 eu project. https://productive40.eu/. Last accessed 9 August 2021
23. Inter IKEA Systems BV (2021) IKEA online planner. https://www.ikea.com/nl/en/planners/. Last accessed 9 August 2021
24. Johnson-Roberson M, Barto C, Mehta R, Sridhar SN, Rosaen K, Vasudevan R (2017) Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks. In: 2017 IEEE international conference on robotics and automation (ICRA), pp 746–753
25. Kaipa K, Morato C, Zhao B, Gupta SK (2012) Instruction generation for assembly operations performed by humans. In: International design engineering technical conferences and computers and information in engineering conference. American Society of Mechanical Engineers, vol 45011, pp 1121–1130
26. Kang HS, Lee JY, Choi S, Kim H, Park JH, Son JY, Kim BH, Do Noh S (2016) Smart manufacturing: Past research, present findings, and future directions. Int J Precis Eng Manuf Technol 3(1):111–128
27. Kerin M, Pham DT (2019) A review of emerging industry 4.0 technologies in remanufacturing. J Clean Prod 237:117805
28. bin Khairai KM, Sutarto AP, bin Abdul MN (2020) The influence of stress on industrial operator's physiology and work performance. Jurnal Optimasi Sistem Industri-Vol 19(2):82–90
29. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv:1412.6980
30. Kusiak A (2018) Smart manufacturing. Int J Prod Res 56(1-2):508–517. https://doi.org/10.1080/00207543.2017.1351644
31. Lambert AJ (2003) Disassembly sequencing: a survey. Int J Prod Res 41(16):3721–3759
32. Lampen E, Teuber J, Gaisbauer F, Bär T, Pfeiffe T, Wachsmuth S (2019) Combining simulation and augmented reality methods for enhanced worker assistance in manual assembly. Procedia CIRP 81:588
33. Le DT, Cortés J., Siméon T. (2009) A path planning approach to (dis) assembly sequencing. In: 2009 IEEE international conference on automation science and engineering. IEEE, pp 286–291
34. Lin TY, Goyal P, Girshick R, He K, Dollár P. (2017) Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision, pp 2980–2988
35. Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollàr P., Zitnick CL (2014) Microsoft COCO: Common Objects in Context. In: Fleet D, Pajdla T, Schiele B, Tuytelaars T (eds) Computer Vision – ECCV 2014. Springer International Publishing, pp 740–755

36. Liu M, Huang Y, Zhang D (2018) Gamification's impact on manufacturing: Enhancing job motivation, satisfaction and operational performance with smartphone-based gamified job design. Human Factors and Ergonomics in Manufacturing & Service Industries 28(1):38–51. https://doi.org/10.1002/hfm.20723
37. Mahanti A, Bagchi A (1985) And/or graph heuristic search methods. J ACM (JACM) 32(1):28–51
38. Mattsson S, Fast-Berglund Å (2016) How to support intuition in complex assembly? Procedia CIRP 50:624–628
39. Mattsson S, Fast-Berglund Å, Åkerman M (2017) Assessing operator wellbeing through physiological measurements in real-time—towards industrial application. Technologies 5(4):61
40. Mattsson S, Fast-Berglund Å, Li D (2016) Evaluation of guidelines for assembly instructions. IFAC-PapersOnLine 49(12):209–214
41. Mattsson S, Li D, Fast-Berglund Å, Gong L (2017) Measuring operator emotion objectively at a complex final assembly station. In: Advances in neuroergonomics and cognitive engineering. Springer, pp 223–232
42. Morato C, Kaipa KN, Gupta SK (2013) Improving assembly precedence constraint generation by utilizing motion planning and part interaction clusters. Comput Aided Des 45(11):1349–1364
43. Mourtzis D, Doukas M (2014) The evolution of manufacturing systems: From craftsmanship to the era of customisation. In: Handbook of research on design and management of lean production systems. IGI Global, pp 1–29
44. Müller J (2020) Enabling technologies for industry 5.0, results of a workshop with europe's technology leaders. Directorate-General for Research and Innovation
45. Murali PK, Darvish K, Mastrogiovanni F (2020) Deployment and evaluation of a flexible human–robot collaboration model based on and/or graphs in a manufacturing environment. Intell Serv Robot 13(4):439–457
46. Nahavandi S (2019) Industry 5.0—a human-centric solution. Sustainability 11(16):4371
47. Bhadana N (2021) Emotion detection network repository. https://github.com/nileshbhadana/emotion_detection. Last accessed 9 August 2021
48. Rijksdienst voor Ondernemend Nederland (2020) Pps-toeslag onderzoek en innovatie. https://www.rvo.nl/subsidie-en-financieringswijzer/pps-toeslag-onderzoek-eninnovatie. Last accessed 9 August 2021 (website in Dutch)
49. Prakash A, Boochoon S, Brophy M, Acuna D, Cameracci E, State G, Shapira O, Birchfield S (2019) Structured domain randomization: bridging the reality gap by context-aware synthetic data. In: 2019 International conference on robotics and automation (ICRA). IEEE, pp 7249–7255. https://doi.org/10.1109/ICRA.2019.8794443
50. Rashid MFF, Hutabarat W, Tiwari A (2012) A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. Int J Adv Manuf Technol 59(1-4):335–349
51. Riegel J, Mayer W, van Havre Y (2021) Freecad (version 0.19). http://www.freecadweb.org. Last accessed 9 August 2021
52. Sanderson AC, de Mello LSH, Zhang H (1990) Assembly sequence planning. AI Mag 11(1):62–62
53. Signify Holding (2021) Philips my creation. https://www.mycreation.lighting.philips.com/en/. Last accessed 9 August 2021
54. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489
55. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556
56. Sivasankaran P, Shahabudeen P (2014) Literature review of assembly line balancing problems. Int J Adv Manuf Technol 73(9-12):1665–1694
57. Su Y, Mao H, Tang X (2020) Algorithms for solving assembly sequence planning problems. Neural Comput Applic 1–10
58. Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press
59. Tao F, Qi Q (2017) New it driven service-oriented smart manufacturing: framework and characteristics. IEEE Trans Syst Man Cybern Syst 49(1):81–91
60. Tao F, Qi Q, Liu A, Kusiak A (2018) Data-driven smart manufacturing. J Manuf Syst 48:157–169
61. Unity Technologies (2021) Unity software suite. https://unity.com/. Last accessed 9 August 2021
62. Tobin J, Fong R, Ray A, Schneider J, Zaremba W, Abbeel P (2017) Domain randomization for transferring deep neural networks from simulation to the real world. Iros 0–7
63. Tremblay J, Prakash A, Acuna D, Brophy M, Jampani V, Anil C, To T, Cameracci E, Boochoon S, Birchfield S (2018) Training deep networks with synthetic data: Bridging the reality gap by domain randomization. arXiv:1804.06516
64. Tseng HE, Tang CE (2006) A sequential consideration for assembly sequence planning and assembly line balancing using the connector concept. Int J Prod Res 44(1):97–116
65. Wang B, Wang G, Sharf A, Li Y, Zhong F, Qin X, CohenOr D, Chen B (2018) Active assembly guidance with online video parsing. In: 2018 IEEE conference on virtual reality and 3D user interfaces (VR). IEEE, pp 459–466
66. Wang J, Liu J, Zhong Y (2005) A novel ant colony algorithm for assembly sequence planning. Int J Adv Manuf Technol 25(11-12):1137–1143
67. Wang Y, Liu J (2010) Chaotic particle swarm optimization for assembly sequence planning. Robot Comput Integr Manuf 26(2):212–222
68. Wang Z, Bovik AC, Sheikh HR, Simoncelli EP (2004) Image quality assessment: from error visibility to structural similarity. IEEE Trans Image Process 13(4):600–612
69. Watson G, Butterfield J, Curran R, Craig C (2010) Do dynamic work instructions provide an advantage over static instructions in a small scale assembly task? Learn Instr 20(1):84–93
70. Xiong J, Hu Y, Wu B, Duan X (2015) Minimum-cost rapid-growing random trees for segmented assembly path planning. Int J Adv Manuf Technol 77(5-8):1043–1055
71. Zhang C, Chen D, Tao F, Liu A (2019) Data driven smart customization. Procedia CIRP 81:564–569
72. Zheng P, Sang Z, Zhong RY, Liu Y, Liu C, Mubarok K, Yu S, Xu X et al (2018) Smart manufacturing systems for industry 4.0: Conceptual framework, scenarios, and future perspectives. Front Mech Eng 13(2):137–150