

# bradscholars

## Automated experience-based learning for plug and produce assembly systems

Item Type	Article
Authors	Scrimieri, Daniele;Antzoulatos, N.;Castro, E.;Ratchev, S.M.
Citation	Scrimieri D, Antzoulatos N, Castro E et al (2017) Automated experience-based learning for plug and procedure assembly systems. International Journal of Production Research. 55(13): 3674-3685.
DOI	<a href="https://doi.org/10.1080/00207543.2016.1207817">https://doi.org/10.1080/00207543.2016.1207817</a>
Rights	© 2017 Taylor & Francis. This is an Author's Original Manuscript of an article published by Taylor & Francis in International Journal of Production Research in 2017, available online at <a href="https://doi.org/10.1080/00207543.2016.1207817">https://doi.org/10.1080/00207543.2016.1207817</a> .
Download date	2026-03-10 21:41:18
Link to Item	<a href="http://hdl.handle.net/10454/17716">http://hdl.handle.net/10454/17716</a>

To appear in the *International Journal of Production Research*  
Vol. 00, No. 00, 00 Month 20XX, 1–15

## Automated experience-based learning for plug and produce assembly systems

Daniele Scrimieri\*, Nikolas Antzoulatos, Elkin Castro, Svetan M. Ratchev

*Department of Mechanical, Materials and Manufacturing Engineering, University of Nottingham,  
Nottingham, NG7 2RD, UK*

*(Received 00 Month 20XX; accepted 00 Month 20XX)*

This paper presents a self-learning technique for adapting modular automated assembly systems. The technique consists of automatically analysing sensor data and acquiring experience on the changes made on an assembly system to cope with new production requirements or to recover from disruptions. Experience is generalised into operational knowledge that is used to aid engineers in future adaptations by guiding them throughout the process. At each step, applicable changes are presented and ranked based on: 1) similarity between the current context and those in the experience base; 2) estimate of the impact on system performance. The experience model and the self-learning technique reflect the modular structure of the assembly machine and are particularly suitable for plug and produce systems, which are designed to offer high levels of self-organisation and adaptability. Adaptations can be performed and evaluated at different levels: from the smallest pluggable unit to the whole assembly system. Knowledge on individual modules can be reused when modules are plugged into other systems. An experimental evaluation has been conducted on an industrial case study and the results show that, with experience-based learning, adaptations of plug and produce systems can be performed in a shorter time.

**Keywords:** plug-and-produce; adaptation; automated assembly; learning; knowledge engineering.

### 1. Introduction

The adaptation of an automated assembly system is normally carried out by engineers through a series of adjustments, performed either manually or by means of software. This process relies on the engineers' skills and experience acquired by experimenting with the machine and, because of its trial and error nature, it is difficult to predict and plan. The required knowledge cannot be easily formalised in order to be reused and be transferable among engineers and to new production environments.

Knowledge on system adaptations helps to guarantee higher responsiveness to disruptions and changes by facilitating problem analysis and supporting decision-making processes. A method for capturing this knowledge is necessary to enable its reuse. Sharing engineers' experience on machines and adaptation processes is also useful for training purposes. Specifically, an engineer performing an adaptation can be provided with details on what other engineers did in similar contexts and the outcome of their actions.

Adaptation knowledge plays a pivotal role in the reuse of equipment. This is fundamental for building cost-effective and sustainable manufacturing solutions, in particular with regard to assembly systems. However, lack of experience with reuse and adaptation, as well as insufficient planning and support during the adaptation itself prevent the achievement of this objective (Harms, Fleschutz, and Seliger 2008; Järvenpää 2012). Hence, manufacturing companies often decide not to adapt existing assembly systems because of the complexity and inefficiency of the process.

---

\*Corresponding author. Email: daniele.scrimieri@nottingham.ac.uk

Despite the development of plug and produce systems to apply autonomously logical adaptations, there is not yet a complete solution to manage effectively the overall adaptation process, which is still heavily human-driven. Although there exist tools and methods for addressing domain-specific problems, no general integrated framework comprising such tools has been developed. Therefore, there is a need for an adaptation framework providing methodologies, information models and automated learning mechanisms to capture adaptation knowledge, support decision-making and facilitate knowledge sharing and reuse.

### 1.1 *Proposed approach*

This paper presents an approach to capturing automatically information on adaptations being performed and using it to produce reusable adaptation knowledge. Adjustments made to the system are recorded and evaluated in the current system state, and stored in an experience base. The evaluation of an adjustment consists of measuring the system performance before and after the change. This allows to determine whether a change has had a positive impact. Adjustments that failed are recorded as well. This is useful in order to avoid reapplying them.

The experience base of adaptations represents the input for the self-learning system. When an engineer adapts a production system for which an experience base has been built, the self-learning system is applied on the current machine state and experience base to recommend applicable adjustments to the engineer. The self-learning algorithm searches the experience base for the most similar adaptation contexts and the best adjustments. A ranked list of adjustments is presented, based on the available experience for the given adaptation context. The engineer will then choose one, apply it and evaluate the resulting state. If the effect is not satisfactory, they can try out a different adjustment from the list.

From a knowledge engineering perspective, the benefits of using our self-learning technique over the manual construction of a knowledge-base for an adaptive manufacturing system are twofold:

- (1) The generated knowledge is automatically built directly from the data. Therefore, there is no need for a knowledge engineer eliciting knowledge from shop floor operators and building a knowledge base. This task is, in fact, heavily dependent on the understanding of the knowledge engineer and their ability in unifying and representing the acquired knowledge.
- (2) The generated knowledge is general, i.e. it is not limited to cover a fixed set of previously seen adaptation scenarios. Instead, it can aid operators in new adaptations, as long as experience on past similar cases has been captured.

This paper is an extension of the work presented at the 15th IFAC Symposium on Information Control Problems in Manufacturing (Scrimieri et al. 2015). The manuscript is organised as follows. Section 2 contains related work on learning and adaptation. Section 3 describes how experience on adaptations is captured and represented. Section 4 presents a technique for learning how to adapt based on the acquired experience. Section 5 describes an experimental evaluation carried out on an industrial case study. Section 6 contains some conclusions.

## 2. Related work

### 2.1 *Adaptive manufacturing systems*

Several manufacturing paradigms and strategies, as well as physical and logical enablers have been proposed to allow rapid system changes (ElMaraghy 2009). Flexible manufacturing systems can adapt to produce anticipated product variations. This level of adaptability is achieved a priori during design as opposed to when new production requirements arise. In contrast, reconfigurable manufacturing systems can adapt to changing volumes of production and unanticipated product

variants. Their modular and adjustable structure enables to change the hardware or control software and offer new capacities and functionalities (ElMaraghy 2006, 2009; Koren et al. 1999).

Two types of adaptations can be identified: static and dynamic. *Static* adaptations are performed “off-line” (when the system is not running) and consist of long-term physical changes (e.g. installation of new equipment). *Dynamic* adaptations are performed “on-line” (when the system is running) and consist of short-term changes to alter, for example, scheduling or routing. Static adaptations are normally planned by human experts or through automatic planning methods, while dynamic adaptations can be carried out autonomously by self-organising systems able to react to changing environments (Westkämper 2006). A capability-based adaptation methodology to match product requirements to available system capabilities is proposed by Järvenpää (2012), with the aim of supporting both human planning and reactive systems.

The multiagent and holonic paradigms are often used to design solutions for dynamic adaptation, by making logical or parametric changes online as the result of distributed decision-making processes involving agents or holons (Leitão and Karnouskos 2015; Shen et al. 2006). Self-managing evolvable assembly systems (Frei et al. 2009) can self-adapt at production time by modifying configuration parameters of individual modules or the distribution of tasks among modules. The system is controlled by a set of distributed agents that can form coalitions which can change dynamically to adapt to new conditions. The automatic identification and configuration of new devices is inspired by the plug-and-produce holonic system of Arai et al. (2001). A plug-and-produce solution with data interpretation and exchange based on industrial ethernet networks for the reconfiguration of robot systems is presented by Reinhart and Krug (2012). Agent evaluation and re-assignment of roles is a key issue for distributed adaptive systems (Zhu, Feng, and Pickering 2013). Cooperation mechanisms among agents are reviewed by Demasure et al. (2014). Agent-based decentralised solutions are also adopted in smart grids (Linnenberg, Wior, and Fay 2013). The self-organising behaviour of evolvable production systems is analysed in Neves et al. (2014a), with the aim of providing design and configuration alternatives for different products.

Successful EU-funded research projects that developed multiagent systems and plug-and-produce technology include IDEAS (Onori et al. 2012; Neves et al. 2014b; Hasan, Onori, and Wikander 2014), GRACE (Pereira, Rodrigues, and Leitão 2013; Cristalli et al. 2013) and PRIME (Antzoulatos et al. 2014). In IDEAS, plug-and-produce self-reconfiguration is accomplished using only multiagent control programmed in intelligent embedded systems, without any higher-level coordination. The mechatronic architecture of IDEAS employs control boards designed specifically for multiagent applications. In GRACE, a multiagent system operates at all stages of production, acting as both a distributed control system and a manufacturing execution system, and sharing process critical information between these two layers. This helps to integrate process and quality control. In PRIME, multiagent control is used for plug-and-produce module integration in production environments with legacy equipment. Through a common agent communication language and a standard interface, a network of heterogeneous control systems from different equipment suppliers can be deployed on the same production line.

Agent-based manufacturing systems share some distinguishing characteristics with cyber-physical systems, namely their distributed nature, with interacting and cooperating entities, and their adaptability and intelligence. In fact, a multiagent system represents a possible implementation of the cyber part of a cyber-physical systems. Cyber-physical systems are a key component of the fourth industrial revolution (Industry 4.0) and can achieve the vision of *smart factories* (Lee 2015).

## 2.2 Learning in manufacturing systems

Adaptation knowledge evolves over time with the introduction of new products and technology. It can be kept up-to-date by continuous learning. Knowledge acquired in a manufacturing organisation

can be measured with learning curves (Jaber 2011). Learning curves can estimate costs based on the cumulative volume of production. The assumption is that costs decrease as cumulative production increases. This is justified by the fact that the more an organisation produces the more it learns how to produce and becomes efficient. If a product to manufacture is similar to another one produced in the past, the old process may be adapted. Resources employed and knowledge built for the production of the old product can be transferred to the new one. Therefore, the starting costs will be lower than those that would be incurred if there was no prior knowledge.

Experimentation is another form of learning. Changes to the process are made and the results are evaluated and compared to the expectations. This is radically different from the learning-by-doing underlying learning curve theories, where learning seems to happen autonomously. Foguem et al. (2008) formalise experience feedback in the context of continuous improvement of industrial systems. The experience feedback process is modelled with conceptual graphs and an ontology is built to define experiential elements. The aim is to transform experience feedback into explicit knowledge and know-how that can be shared among the involved actors.

Examples of automated learning in manufacturing systems include a reinforcement learning technique for reconfiguration games that captures the effects of a sequence of reconfiguration decisions (McDonnell, Joshi, and Qiu 2005), and a learning method for adaptive controllers of manufacturing cells using a cerebellar model articulation controller network (Sun, Chang, and Yih 2005).

### 2.2.1 *Learning during ramp-up*

A particular stage of the life-cycle of production systems in which learning plays an important role is the ramp-up phase. This phase represents the period of time it takes for a new production system to reach high levels of production and product quality (Koren et al. 1999). It has the following characteristics (Surbier 2010):

- The production capacity is low;
- There are frequent disturbances regarding the equipment, the quality of the product and the supply chain;
- The initial level of knowledge about the product and the process is low.

A conceptual framework for production ramp-up and a review of decision-support models are presented by Glock and Grosse (2015). When the ramp-up phase starts, the production process is not entirely understood and discrepancies hinder efficiency. The initial process recipe is not yet suitable for large volumes of production. Although changes are required, they may conflict with learning. Terwiesch and Xu (2004) consider learning as the process of eliminating the discrepancies between process specification and execution during ramp-up. They analyse the trade-off between applying process changes immediately and accomplishing learning tasks. A ramp-up strategy called “copy-exactly” is proposed, which freezes the process recipe for a certain period of time (i.e. it does not allow changes) to enable learning.

Fjällström et al. (2009) analyse the types of information needed to deal with critical events during ramp-up. The authors consider problem, domain and problem solving information in a case study in the automotive industry. They find out that problem and domain information are the most important. In addition, the domain information required to respond to a critical event is not exclusively related to the category of the event. For example, handling a critical event about a product may require domain information not only about the product itself, but also about the process and the equipment.

Oates, Scrimieri, and Ratchev (2012) and Scrimieri, Oates, and Ratchev (2015) describe some automated techniques specifically designed for supporting engineers during the ramp-up of an assembly system. Two k-nearest neighbour algorithms are used to classify machine states, indicating which adjustment to perform and which value to set. These works address in particular the following

challenges:

- The dimensionality of the data constantly changes. In fact, the number of parameters used to describe machine states changes as modules or sensors are added or removed;
- The training set may not be large enough to generalise from;
- The experience should be transferable among similar machines.

### 3. Experience capture in plug and produce systems

A plug and produce assembly system can be described as a collection of stations or modules for assembling or checking parts. Parts are processed sequentially or in parallel by a certain number of modules. The sequence of modules a part goes through is determined by the assembly process and the scheduling policy. Modules can be replaced with others having similar functionality and interfaces in case of breakdown or to adapt to a new process. Also, new modules can be added to increase production volumes.

Modules are the smallest entities we are interested in when learning. When they are moved from one system to another we want to be able to transfer the experience base built on them. The analysis of the operation of a single module independently from the others does not normally provide us with all the information we need to measure its performance. This is evident in systems where the check of an assembly operation happens in a subsequent step by means of a dedicated module. In this case, we can measure the quality of the output of a module and assess its operation only when the check takes place. Our object of study is given by production systems performing intermediate quality checks on the parts being assembled, in addition to a last check on the final product. We call *subsystem* a group of one or more modules for assembly and checking a part. A subsystem could be reused in a different production system or replicated within the same system.

The process executed by a subsystem can be configured by setting subsystem-specific parameters in a *process recipe*. In addition, there can be a process recipe for the entire system, which defines system-level parameters affecting more than one subsystem, possibly all. For instance, the layout of the pallets carried by a conveyor belt and processed by every subsystem can be a system parameter specified in the system recipe. A KPI (key performance indicator) is defined for each subsystem and for the whole system to measure their performance. We assume that a KPI function takes non-negative values and is such that the higher the value, the better the performance. Usually a KPI is a function of sub-assembly or assembly quality and throughput.

#### 3.1 Event generation

We consider assembly systems equipped with programmable logic controllers, or equivalent industrial control systems, generating real-time signals which are then transformed into high-level data structures (*events*), handled by our software. Events contain information on cycle times, results of checks or measurements, adjustments performed, modules added or removed, product item or pallet being processed, and anything else that can be monitored and can be useful to characterise the system state or measure its performance. In fact, events are used to capture the state of the system at specific points and identify changes being made.

We differentiate two types of events: *status events*, conveying information about the current status of the system, and *adjustment events*, describing changes to the process recipe. In addition, we identify a sub-type of status events, called *disruptive events*, generated when a disruption occurs. In general, by disruption we mean a significant variation in a monitored variable. Disruptions can be detected by statistical process control or condition monitoring techniques. Usually they are symptoms of suboptimal conditions or developing failures and require human intervention.

The assembly system used in the experimentation (Section 5) is equipped with sensors for

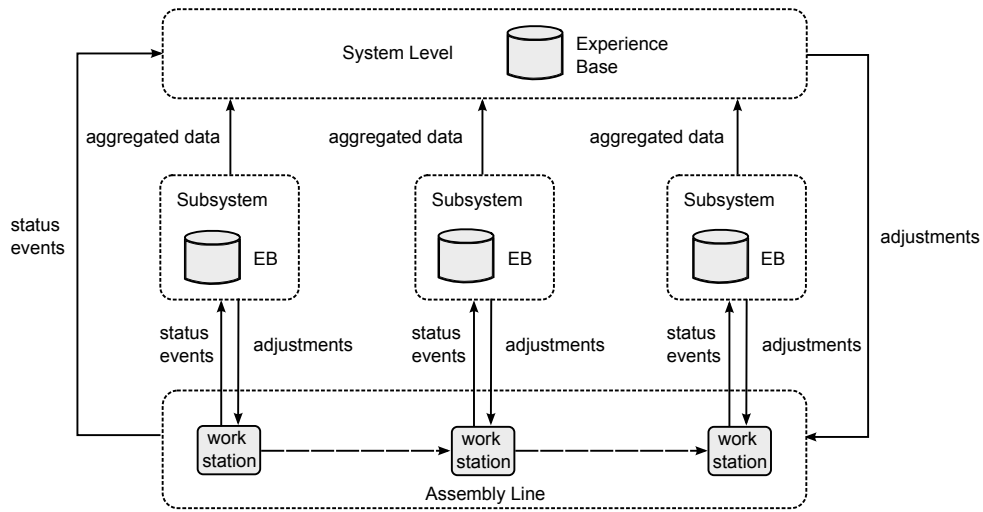


Figure 1. Overview of the architecture of the adaptation framework showing the system and subsystem layers, the experience bases, and the flow of data (Scrimieri and Ratchev 2014).

detecting the presence of parts on a pallet at the beginning of the process, and for checking their assembly after each operation. Sensor data is used to generate status events containing the results of the checks carried out on each part. Other events are generated by the software. For instance, when changing opening or closing angles of grippers an adjustment event is generated. The learning technique described in this paper does not depend on the generated events and the KPI function. The only requirement is that the experience creation process, detailed in the next subsection, must be able to retrieve status information from events in order to represent the machine state and calculate the KPI.

Let us call *attribute* each process parameter and monitored variable whose detected values are transmitted by status events. Attributes can be associated with individual modules, subsystems or the entire system. The set of available attributes depends on the modules and sensors present at a specific moment. If an attribute is not available, its value is undefined.

### 3.2 Machine state and experience representation

In our framework, experience is the representation of changes applied to an assembly system. It describes experiments made and their outcome. Its creation is triggered by adjustment events. At the subsystem level, an experience instance is related to a change to a subsystem, and includes the effect of the change on the subsystem. At the system level, experience is the result of either a subsystem or system change and includes the effect of the change on the whole system. An outline of the system architecture with system and subsystem experience bases is shown in Figure 1.

The creation of an experience instance (Figure 2) starts when an adjustment event is received. At this point, the state of the machine is captured using the status information contained in the latest status events. The KPI is then calculated on this state. Depending on the adjustment type, when the adjustment has taken effect the state of the machine is captured again using the new status information received. The new KPI value is calculated and a new experience instance is created in the experience base.

Suppose that we have a system consisting of  $m$  subsystems. For each  $1 \leq i \leq m$ , let  $\text{Attr}_i = \{a_1, \dots, a_{n_i}\}$  be the set of all attributes that can be used to characterise the state of subsystem  $i$  over a certain period of time, and let  $D^j$  ( $1 \leq j \leq n_i$ ) be the domain of attribute  $a_j$ .  $\text{Attr}_i$  also includes all the system-level attributes which are relevant to subsystem  $i$ . We represent a state  $S$  of subsystem  $i$  at a certain point in time by a mapping from  $i$ 's attributes to their respective values,

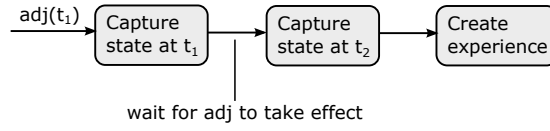


Figure 2. Experience creation. The state of the machine before ( $t_1$ ) and after ( $t_2$ ) an adjustment  $adj$  is captured and the KPI is calculated on both these states.

i.e. by a function  $S : Attr_i \rightarrow \bigcup_j D^j \cup \{null\}$  such that:

$$S(a_j) = \begin{cases} d \in D^j & \text{if attribute } a_j \text{ is present in state } S \\ null & \text{otherwise} \end{cases}$$

In addition, let  $D^S$  denote the set of attributes present in state  $S$ , i.e. all attributes  $a_j$  such that  $S(a_j) \in D^j$ . We denote the set of states of subsystem  $i$  by  $State_i$ . The KPI of subsystem  $i$  is a function defined on  $State_i$ .

A subsystem enters a new state every time any of its attributes changes or a module is added, removed or replaced. When a new module is inserted, the associated attributes become available, although not necessarily all at the same time. When a module is removed, the associated attributes are no longer available. A transition from  $S$  to  $T$  will be written as  $S \rightarrow T$ . During its operation, a subsystem enters a sequence of states  $S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_l$ , where for each  $1 \leq k \leq l-1$  there exists a  $j$  such that one of the following two conditions holds:

- (1)  $a_j \in D^{S_k} \cap D^{S_{k+1}}$  and  $S_k(a_j) \neq S_{k+1}(a_j)$  (i.e.  $a_j$  has changed value in the transition  $S_k \rightarrow S_{k+1}$ );
- (2)  $a_j \in D^{S_k} \Delta D^{S_{k+1}}$  (i.e.  $a_j$  has been introduced or removed in the transition  $S_k \rightarrow S_{k+1}$ ).

An adjustment is specified by a type, which indicates what the change is about (e.g. pressure, conveyor belt speed, cam angles, pick-and-place positions), and a value, which depends on the type. Let  $Adj_i$  be the set of adjustments for subsystem  $i$ . An adjustment  $adj \in Adj_i$  has the form  $(t, v)$ , where  $t$  is the type and  $v$  is the value.

### 3.2.1 Subsystem experience

We can now define an experience instance  $exp$  for subsystem  $i$  as follows:

$$exp = (adj, S, C, S')$$

where

- $adj \in Adj_i$ ;
- $S \in State_i$  is the state when  $adj$  is made;
- $C \subset Attr_i$  is the set of attributes that caused a disruption in a certain period of time before performing  $adj$ ;
- $S' \in State_i$  is the first state after  $adj$  has been performed and all available attributes have been updated.

### 3.2.2 System experience

Let  $SysAdj$  be the set of system-level adjustments and, for  $1 \leq i \leq m$ , let:

- $S_i \in State_i$  be the state in which subsystem  $i$  is when  $adj$  is made;
- $C \subset Attr_i$  be the set of attributes that caused a disruption in subsystem  $i$  in a certain period of time before performing  $adj$ ;

- $S'_i \in \text{State}_i$  be the first state of subsystem  $i$  after  $adj$  has been performed and all available attributes have been updated.

A system-level experience instance  $sysexp$  has the following form:

$$sysexp = (adj, S, C, S')$$

where  $adj \in \text{SysAdj} \cup (\bigcup_i \text{Adj}_i)$ ,  $S = \bigcup_i S_i$  and  $S' = \bigcup_i S'_i$ . Note that system attributes are also specified (with the same value) in all subsystem states in which they are relevant. Analogously to subsystems, the system KPI is calculated on a system state, given by the union of the subsystem states.

#### 4. Learning and generation of adaptation recommendations

We have presented a general method for capturing and representing experience on a plug and produce system. What is still missing is a context-aware automatic procedure employing stored experience to derive successful adaptations systematically. When needed, the engineer should be able to query the self-learning system and be provided with a list of recommended adjustments, ranked from best to worst. Two phases can be identified in the experience-based adaptation mechanism:

- (1) Event generation and experience capture during an adaptation, and creation of an experience base (Section 3).
- (2) System adaptation using the self-learning system and an experience base, as presented in this section.

The problem of finding an appropriate adjustment to perform in a specific system state can be seen as a *classification problem*. This is a typical problem in machine learning, where a system learns how to classify instances based on a given set of examples, the *training set*. Each example contains an input vector and a discrete output value that specifies the class of the example. The system receives the training set and tries to generalise the association between input vectors and output classes. The output of this process is a *classifier* program. When given a new instance not included in the training set, the classifier predicts its correct class. A variety of learning methods have been developed for classification, such as artificial neural networks, instance-based learning, support vector machines and decision trees (see, for example, Marsland 2009).

In the experience model we have defined, given an experience base at either the subsystem or system level, the initial states and adjustment types of its instances represent the examples of the training set and the adjustment type of an experience instance is the example's class. The instances to be classified are the states entered by the assembly machine throughout an adaptation in which the engineer requests assistance. The classification process guides the engineer in finding an appropriate adjustment in each of these states. Note that it is important not only to find the optimal setting of parameters, but also the sequence in which making changes as, in general, there could be constraints on applicable changes in a certain state.

##### 4.1 Experience base search

Our self-learning technique searches the experience base using a variant of the *K-nearest neighbour* algorithm (kNN) (Aha, Kibler, and Albert 1991). In kNN, given an instance to be classified, the aim is to find the  $k$  nearest instances in the training set, minimising a certain distance measure. The distance measure expresses the similarity between examples, in the sense that the smaller the distance between two examples, the more similar they are supposed to be. The class to be assigned to an instance is the most common among the  $k$  neighbours or is chosen by adopting some voting

scheme.

In addition to similarity between system states, our approach to classification uses performance estimates based on KPI values calculated on the experience. The resulting measure is not itself a distance function, though (i.e. a non-negative and symmetric function satisfying the triangle inequality). The method is equally applicable at both subsystem and system level.

#### 4.1.1 Distance function

We define the distance  $d(S_1, S_2)$  between system states  $S_1$  and  $S_2$  over attributes  $a_1, \dots, a_n$ . Since we handle both numerical and categorical attributes, we use the *heterogeneous euclidean-overlap metric* (Wilson and Martinez 1997):

$$d(S_1, S_2) = \sqrt{\sum_{i=1}^n (d_i(S_1, S_2))^2},$$

where  $d_i(S_1, S_2)$ , for  $1 \leq i \leq n$ , is the distance between  $S_1$  and  $S_2$  on attribute  $a_i$ :

$$d_i(S_1, S_2) = \begin{cases} 1 & S_1(a_i) = \text{null} \text{ or} \\ & S_2(a_i) = \text{null}, \\ \text{overlap}(S_1(a_i), S_2(a_i)) & a_i \text{ is nominal,} \\ \text{rndiff}_i(S_1(a_i), S_2(a_i)) & \text{otherwise.} \end{cases}$$

The function *overlap* gives 0 if its arguments are the same, otherwise 1. The function *rndiff<sub>i</sub>* (range normalised difference), calculated if attribute  $a_i$  is numerical, is defined as:

$$\text{rndiff}_i(x, y) = \frac{|x - y|}{\max(a_i) - \min(a_i)},$$

where  $\max(a_i)$  and  $\min(a_i)$  are, respectively, the maximum and minimum values of  $a_i$  observed in the training set.

Neighbours are found using the distance function, which calculates distances on all attributes. Redundant, irrelevant, interacting or noisy attributes have a negative effect on the similarity search. To deal with these issues, many weight-setting methods have been proposed (Wettschereck, Aha, and Mohri 1997). A scheme tailored to our problem would assign higher weights to attributes that have caused a disruption either in the state where an adjustment is being made or in the experience. With such a scheme, states exhibiting the same disruption would be considered closer than those that do not.

#### 4.1.2 Similarity-performance function

We are interested not only in finding the most similar experience instances, but also those containing the best adaptations, i.e. producing the highest KPI values. An experience instance with an initial state identical to the one we want to classify, but whose adjustment causes a performance drop, would not be helpful. Based on this observation, we also analyse the KPI of the final state in the experience.

Let  $E = (adj, S, C, S')$  be an experience instance,  $T$  be a state and  $kpi$  be a KPI function. We define the *similarity-performance function*  $e_{kpi}(E, T)$  as follows, for  $kpi(S') \neq 0$  (Scrimieri, Oates, and Ratchev 2015):

$$e_{kpi}(E, T) = \frac{d(S, T)}{kpi(S')}$$

If  $kpi(S') = 0$ , it means that *adj* made the system non-functional, so  $e_{kpi}(E, T)$  is defined to be infinitely large in this case. Apart from the similarity between states, this function takes into account the performance according to the experience acquired. The smaller the value of this function, the better the adjustment of the experience is expected to be in the state being examined.

## 4.2 Voting and ranking

Let  $T$  be a state to be classified and  $kpi$  be a KPI. Two cases must be distinguished.

$k = 1$  The ranked list of adjustments for  $T$  is generated by sorting the experience instances by  $e_{kpi}(E, T)$  in ascending order. The experience instance with the smallest value of  $e_{kpi}(E, T)$  represents the recommended adjustment type and value.

$k > 1$  A voting scheme must be applied to select a class among those of the  $k$  neighbours. The scheme that selects the most common class among the neighbours has the disadvantage that the most common class tends to be the most frequent in the training set. One typical approach is to use, as weights, the distances of the neighbours from the point to classify. We use  $e_{kpi}$ , which takes into account the distance of a neighbour as well as the effect of the corresponding adjustment on system performance. Let  $E_1, \dots, E_k$  be the  $k$  experience instances with the smallest values of  $e_{kpi}$ . The weight  $w(E_i)$  assigned to  $E_i$  is defined as:

$$w(E_i) = \frac{e_{kpi}(E_k, T) - e_{kpi}(E_i, T)}{e_{kpi}(E_k, T) - e_{kpi}(E_1, T)}$$

The instances  $E_1, \dots, E_k$  are grouped by class. The weights of the instances of each class are summed up. The class with the largest sum of weights represents the recommended adjustment type. The adjustment value for an adjustment type  $t$  is given by the weighted mean of the adjustments values of the instances (among  $E_1, \dots, E_k$ ) having class  $t$ , using  $w(E_i)$  as weights.

## 5. Experimental evaluation

The self-learning technique was implemented in a software package and tested and evaluated on a production system for the assembly of injection pens. The system comprises pick-and-place modules for changing the orientation of parts and putting them together, and inspection modules for checking the presence of parts, their assembly and the quality of the final product. Three subsystems can be identified, performing the following operations and the relevant checks:

- (1) Putting tanks into pen bodies;
- (2) Putting caps on pen bodies;
- (3) Revesing pens.

A product is accepted only if it passes all the checks, otherwise it is rejected. No rework occurs on bad parts because it is not considered cost-effective. In such a production system it is therefore extremely important to maximise the number of good parts. Throughput cannot be altered because cycle times are fixed. In fact, parts are moved by a synchronous conveyor system from station to station along a fixed path.

The objective of the experimentation was to evaluate how the self-learning technique can aid operators in system adaptations by ranking applicable adjustments. To this end, the system was adapted for the manufacture of a product variant having physical and geometrical characteristics slightly different to the original product. The adaptation involved mechanical changes (e.g. installation of different grippers) and adjustments of parameters related to grippers and checking devices

(e.g. opening and closing angles of grippers, pressure, part dimensions). This process may be quite challenging because assembly operations require very precise and repeatable positioning.

The adaptation for the assembly of the product variant was considered successful when a certain KPI on product quality was reached, as described below. In spite of measuring the quality of the product being assembled, the aim of the experimentation was not to increase product quality per se, but to adapt the machine to produce a new product variant at the required level of quality. At the beginning, the quality is not sufficient and different adjustments are made to increase it.

The system was first adapted one subsystem at a time, then as a whole. Two groups of 6 operators each were used in the experimentation, one for adaptation at the subsystem level and one for adaptation at the system level. Each group was in turn split into 2 groups of 3 operators each, one for adapting (the subsystems or the system) without assistance and one for adapting with the assistance provided by the self-learning framework. The subdivision into groups and their names are as follows:

	No assistance	With self-learning
Subsystem-level	Group A	Group B
System-level	Group C	Group D

The operators working at the subsystem level adapted all the subsystems, with no particular order. All the operators performed the adaptations individually and the original state of each subsystem or of the whole system was restored after each adaptation. All the operators had comparable skills and experience and did not communicate with each other during the experiment. The operators of groups B and D could get a list of ranked adjustments by the self-learning system at any time during the adaptation and they were free to apply any adjustment, whether or not in the list.

The experimentation consisted of 4 phases:

- (1) Each subsystem was adapted by group A. A subsystem-level experience base was built for each subsystem by capturing all the adjustments performed on it.
- (2) Each subsystem was adapted by group B, using the subsystem-level experience base built in the previous phase by group A.
- (3) The whole system was adapted by group C and a system-level experience base was built by capturing all the adjustments performed on it.
- (4) The whole system was adapted by group D, using the system-level experience base built in the previous phase by group C.

The average levels of performance reached and the number of adjustments required by groups A and B for completing the adaptation of each subsystem and by groups C and D for completing the adaptation of the whole system were compared. The performance of the whole system and of each subsystem was characterised in terms of product quality. A KPI was defined as the current number of good parts out of the total number of parts assembled during an adaptation. The KPI value was updated after checking the quality of each assembled part. Operators could read the current KPI value at all times, so that they could get feedback on their actions. The target KPI value that operators were expected to obtain to complete each adaptation process was 0.98.

The average KPI values obtained by groups A and B at each adaptation step of each subsystem are shown in Figures 3, 4 and 5, whereas those obtained by groups C and D for the whole system are shown in Figure 6. As it can be noted, the groups which received assistance from the self-learning framework could achieve higher KPI values at virtually every step. These experiments suggests that our self-learning technique enables operators to complete adaptations at the subsystem or system level in fewer steps, therefore saving time.

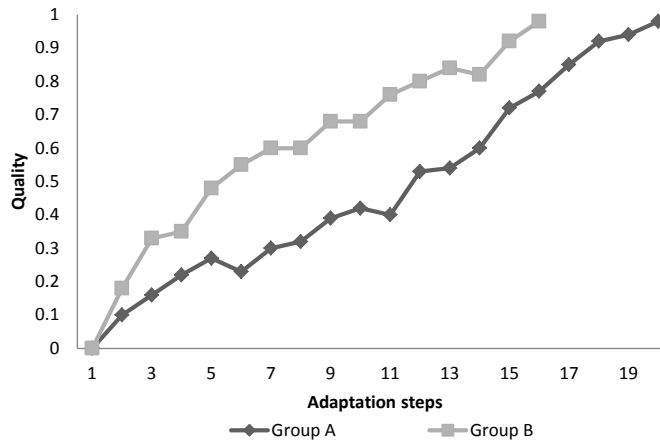


Figure 3. Average quality values obtained by groups A and B for subsystem 1.

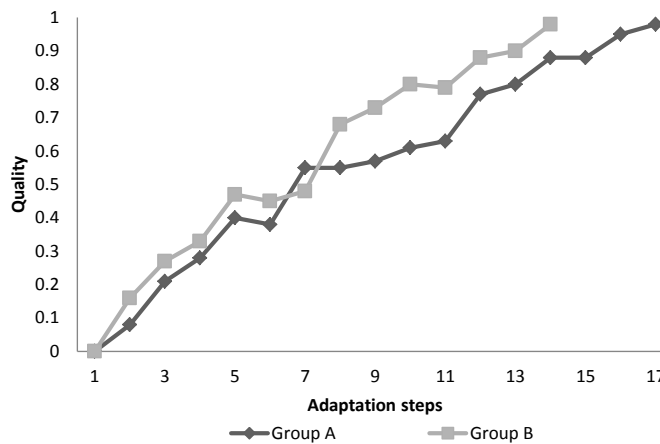


Figure 4. Average quality values obtained by groups A and B for subsystem 2.

## 6. Conclusion

An automated method for learning from experience captured on plug and produce assembly systems has been presented. Since knowledge is built directly from data, there is no need for a manual construction of a knowledge base. The experimental evaluation carried out on a modular assembly system indicates that the self-learning technique can effectively help operators to choose appropriate adjustments. The domain of application of the technique is, however, not limited to assembly systems and can be extended to other production systems.

The accuracy of the technique depends on the similarity-performance function, and thus on the distance and KPI functions. The use of a distance function is based on the assumption that the more similar two states are, the more likely an adjustment has the same effect on them. To refine the similarity-performance function, attributes should be weighted dynamically based on their relevance in the selection of a certain adjustment. Another interesting development of this function is the integration of a method to assess adaptations based on current production capabilities with a complex semantics that cannot be captured by the attributes of a machine state.

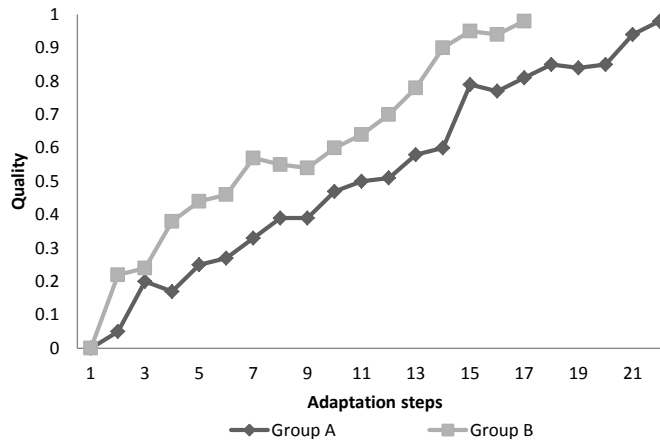


Figure 5. Average quality values obtained by groups A and B for subsystem 3.

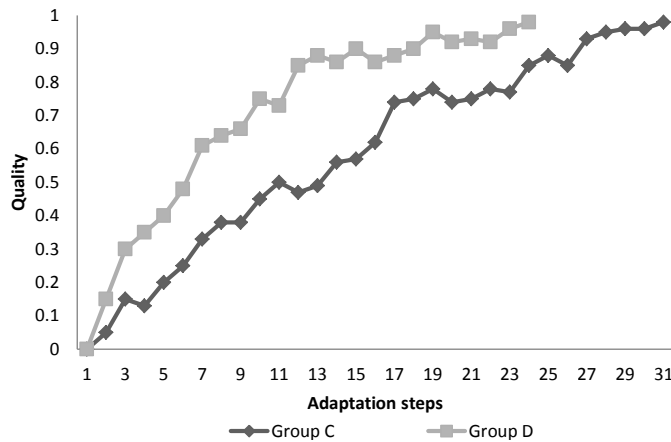


Figure 6. Average quality values obtained by groups C and D for the whole system.

## Funding

This work was supported by the European Union [grant agreement n. 314762].

## References

- Aha, D. W., D. Kibler, and M. K. Albert. 1991. "Instance-Based Learning Algorithms." *Machine Learning* 6 (1): 37–66.
- Antzoulatos, Nikolas, Elkin Castro, Daniele Scrimieri, and Svetan M. Ratchev. 2014. "A multi-agent architecture for plug and produce on an industrial assembly platform." *Production Engineering* 8 (6): 773–781.
- Arai, T., Y. Aiyama, M. Sugi, and J. Ota. 2001. "Holonc assembly system with Plug and Produce." *Computers in Industry* 46: 289–299.
- Cristalli, C., M. Foehr, T. Jäger, P. Leitão, N. Paone, P. Castellini, C. Turrin, and I. Schjolberg. 2013. "Integration of process and quality control using multi-agent technology." In *IEEE International Symposium on Industrial Electronics (ISIE)*, May, 1–6.
- Demesure, G., M. Defoort, A. Bekrar, D. Trentesaux, and M. Djema. 2014. "Cooperation mechanisms in multi-agent robotic systems and their use in distributed manufacturing control: Issues and literature review." In *Industrial Electronics Society, IECON 2014 - 40th Annual Conference of the IEEE*, Oct, 2538–2543.

- ElMaraghy, H. A. 2006. “Flexible and Reconfigurable Manufacturing Systems Paradigms.” *International Journal of Flexible Manufacturing Systems* 17: 261–276.
- ElMaraghy, H. A., ed. 2009. *Changeable and Reconfigurable Manufacturing Systems*. Springer.
- Fjällström, S., K. Säfssten, U. Harlin, and J. Stahre. 2009. “Information enabling production ramp-up.” *Journal of Manufacturing Technology Management* 20 (2): 178–196.
- Foguem, B. K., T. Coudert, C. Béler, and L. Geneste. 2008. “Knowledge formalization in experience feedback processes: An ontology-based approach.” *Computers in Industry* 59: 694–710.
- Frei, R., B. Ferreira, G. Di Marzo Serugendo, and J. Barata. 2009. “An architecture for self-managing evolvable assembly systems.” In *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, 2707–2712.
- Glock, Christoph H., and Eric H. Grosse. 2015. “Decision support models for production ramp-up: a systematic literature review.” *International Journal of Production Research* 53 (21): 6637–6651.
- Harms, R., T. Fleschutz, and G. Seliger. 2008. “Knowledge Based Approach to Assembly System Reuse.” In *Proceedings of the 9th ASME Biennial Conference on Engineering Systems Design and Analysis*, 295–302.
- Hasan, Baha, Mauro Onori, and Jan Wikander. 2014. “Assembly Features Utilization to Support Production System Adaptation.” In *Technological Innovation for Collective Awareness Systems: 5th IFIP WG 5.5/SOCOLNET Doctoral Conference on Computing, Electrical and Industrial Systems, DoCEIS 2014, Costa de Caparica, Portugal, April 7-9, 2014. Proceedings*, edited by Luis M. Camarinha-Matos, Nuno S. Barrento, and Ricardo Mendonça, 85–92. Springer Berlin Heidelberg.
- Jaber, M. Y., ed. 2011. *Learning Curves: Theory, Models, and Applications*. CRC Press.
- Järvenpää, E. 2012. “Capability-based Adaptation of Production Systems in a Changing Environment.” Ph.D. thesis, Tampere University of Technology, Finland.
- Koren, Y., U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. van Brussel. 1999. “Reconfigurable Manufacturing Systems.” *CIRP Annals - Manufacturing Technology* 48: 527–540.
- Lee, Jay. 2015. “Smart Factory Systems.” *Informatik-Spektrum* 38 (3): 230–235.
- Leitão, Paulo, and Stamatis Karnouskos, eds. 2015. *Industrial Agents: Emerging Applications of Software Agents in Industry*. Morgan Kaufmann.
- Linnenberg, T., I. Wior, and A. Fay. 2013. “Analysis of potential instabilities in agent-based smart grid control systems.” In *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, Nov, 7422–7427.
- Marsland, S. 2009. *Machine Learning: An Algorithmic Perspective*. Chapman and Hall/CRC.
- McDonnell, P., S. Joshi, and R. G. Qiu. 2005. “A learning approach to enhancing machine reconfiguration decision-making games in a heterarchical manufacturing environment.” *International Journal of Production Research* 43 (20): 4321–4334.
- Neves, P., L. Ribeiro, J. Dias-Ferreira, M. Onori, and J. Barata. 2014a. “Exploring reconfiguration alternatives in self-organising evolvable production systems through simulation.” In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, July, 511–518.
- Neves, Pedro, Luis Ribeiro, Mauro Onori, and José Barata. 2014b. “Performance Assessment in Self-organising Mechatronic Systems: A First Step towards Understanding the Topology Influence in Complex Behaviours.” In *Technological Innovation for Collective Awareness Systems: 5th IFIP WG 5.5/SOCOLNET Doctoral Conference on Computing, Electrical and Industrial Systems, DoCEIS 2014, Costa de Caparica, Portugal, April 7-9, 2014. Proceedings*, edited by Luis M. Camarinha-Matos, Nuno S. Barrento, and Ricardo Mendonça, 75–84. Springer Berlin Heidelberg.
- Oates, R. F., D. Scrimieri, and S. Ratchev. 2012. “Accelerated Ramp-Up of Assembly Systems through Self-learning.” In *Proceedings of the 6th IFIP WG 5.5 International Precision Assembly Seminar (IPAS 2012)*, 175–182. Springer.
- Onori, Mauro, Niels Lohse, Jose Barata, and Christoph Hanisch. 2012. “The IDEAS project: plug & produce at shop-floor level.” *Assembly automation* 32 (2): 124–134.
- Pereira, A., N. Rodrigues, and P. Leitão. 2013. “Data collection for global monitoring and trend analysis in the GRACE multi-agent system.” In *IEEE International Conference on Industrial Technology (ICIT)*, Feb, 1240–1245.
- Reinhart, G., and S. Krug. 2012. “Automatic Configuration (Plug & Produce) of Robot Systems Data-Interpretation and Exchange.” In *Enabling Manufacturing Competitiveness and Economic Sustainability*, edited by Hoda A. ElMaraghy, 147–152. Springer Berlin Heidelberg.
- Scrimieri, Daniele, Nikolas Antzoulatos, Elkin Castro, and Svetan M. Ratchev. 2015. “Automated

- Experience-Based Learning for Plug and Produce Assembly Systems.” In *15th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2015)*, .
- Scrimieri, D., R.F. Oates, and S.M. Ratchev. 2015. “Learning and reuse of engineering ramp-up strategies for modular assembly systems.” *Journal of Intelligent Manufacturing* 26 (6): 1063–1076.
- Scrimieri, D., and S.M. Ratchev. 2014. “A k-Nearest Neighbour Technique for Experience-Based Adaptation of Assembly Stations.” *Journal of Control, Automation and Electrical Systems* 25 (6): 679–688.
- Shen, Weiming, Qi Hao, Hyun Joong Yoon, and Douglas H. Norrie. 2006. “Applications of agent-based systems in intelligent manufacturing: An updated review.” *Advanced Engineering Informatics* 20 (4): 415–431.
- Sun, Y. L., T. M. Chang, and Y. Yih. 2005. “Learning-based adaptive controller for dynamic manufacturing cells.” *International Journal of Production Research* 43 (14): 3011–3025.
- Surbier, L. 2010. “Problem and interface characterization during ramp-up in the low volume industry.” Ph.D. thesis, Institut polytechnique de Grenoble, France.
- Terwiesch, C., and Y. Xu. 2004. “The copy-exactly ramp-up strategy: trading-off learning with process change.” *IEEE Transactions On Engineering Management* 51 (1): 70–84.
- Westkämper, E. 2006. “Factory Transformability: Adapting the Structures of Manufacturing.” In *Reconfigurable Manufacturing Systems and Transformable Factories*, edited by A.I. Daschenko, 371–381. Springer Berlin / Heidelberg.
- Wettschereck, D., D. W. Aha, and T. Mohri. 1997. “A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms.” *Artificial Intelligence Review* 11: 273–314.
- Wilson, D. R., and T. R. Martinez. 1997. “Improved heterogeneous distance functions.” *Journal of Artificial Intelligence Research* 6: 1–34.
- Zhu, Haibin, Luming Feng, and R. Pickering. 2013. “Agent Evaluation in Distributed Adaptive Systems.” In *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct, 752–757.