Visual Programming of Robot Tasks with Product and Process Variety

Dominik Riedelbauch and Sascha Sucker

Abstract

In flexible manufacturing settings, automation is shaped by ever changing conditions (e.g. varying part feeding locations, highly customizable products). Quick adaptation of robot systems is mostly achieved by visual end-user robot (re-)programming. In this paper, we discuss the explicit integration of anticipated product and process variety into visually programmed tasks. We contribute a task model which captures a user-defined range of task variants. To this end, parts are specified in terms of approximate locations and generalized parts families. Workspace exploration and combinatorial assignment planning enable online adaptation to unknown environments. Our experiments show that this adaptation capability can increase the economical efficiency of cobot use.

Keywords

Flexible production • Visual programming • Intelligent robots

D. Riedelbauch (✉) · S. Sucker
Lehrstuhl für Angewandte Informatik III, Universität Bayreuth, Bayreuth, Germany
e-mail: dominik.riedelbauch@uni-bayreuth.de

S. Sucker
e-mail: sascha.sucker@uni-bayreuth.de
URL: https://robotics.uni-bayreuth.de

© The Author(s) 2023

T. Schüppstuhl et al. (eds.), *Annals of Scientific Society for Assembly, Handling and Industrial Robotics 2022*, https://doi.org/10.1007/978-3-031-10071-0_20

1 Introduction and Related Work

Contrasting to traditional mass production, manufacturing demands have shifted towards shorter innovation cycles and small-batch production. This has raised the demand for *flexible manufacturing* systems that can quickly be adapted to customized products by domain-experts in small and medium enterprises [6]. When additionally considering recent advances in collaborative robotics towards flexible *partial automation*, adaptation of robot programs to various sources of variety are needed [1, 4]: *Product variety* is needed to manufacture different product instances from a product family by assembling parts with varying features (e.g. color) to suite individual customer demands [9]. In this field, we particularly focus on *process-specific variations* [7] that additionally yield *process variety*. Relevant robot task parameters that may change with process-specific variations are e.g. pickup or placement locations, or even the ordering of process steps [1].

Visual end-user robot programming is an established approach to cope with such variety [6]. Corresponding approaches [14, 17–19] are mostly based on skill frameworks. Those let users combine skills with human-readable semantics into tasks (e.g. [15]) even for human-robot collaboration [16, 18]. Modularity and intuitive usability support convenient (re-)programming and, in consequence, quick adaptation. In contrast, our contribution seeks to reduce recurrent programming efforts by applying the visual programming paradigm to a task model that intrinsically encodes a subset of feasible variations (e.g. different part types or locations) and adapts online (Fig. 1). We hypothesize that this would further contribute to the economic efficiency of intelligent robot systems.

Corresponding **task models with variety** have also been addressed in literature. Among them, especially precedence graphs and hierarchical AND/OR Trees are frequently used in intelligent robot systems (e.g. [4, 13, 16]). They seek to encode all feasible assembly sequences [10], hence focussing on process variety. Similarly, hierarchical models emphasizing product variety [7, 9, 11], approaches at the intersection of assembly and product family oriented goals [5], and ontologies to exchange production data under variety [8] have been proposed. They commonly decompose products into functional entities [11] until inseparable, constituent components referred to as *primary generic products* [9] or *parts*

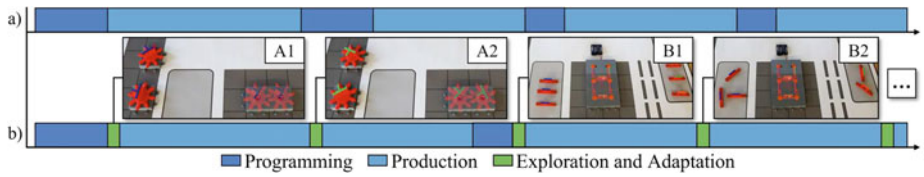


Fig. 1 Visual programming enables frequent end-user robot task adaptation to customer demands in flexible manufacturing (**a**). We seek to reduce programming efforts by online adaptation (**b**). To this end, we propose to explicitly encode different situations with product variety (A1, A2) or process variety (B1, B2) in a single task model

families [7] are reached. A group of feasible variants for assigning a part in concrete product instances is associated with each component. Analogously, groups of feasible locations can be expressed with spatial relations [14], or more specifically with areas in the workspace [18]. Taking inspiration from this group notion for feasible part types and locations, we propose end-user programming of assembly task models with skills accepting parts families and partly known locations as input. This way, parameters can be partially left underspecified at modelling time to create a single task model for several instances of the task. Consider e.g. a pick-and-place task that involves fetching five `bolts` from the imprecise location `conveyor` and putting them into a `box`—with our approach, a single task model is sufficient to robustly conduct this kitting task for any positions and orientations of bolts on the conveyor, and for any size of bolts.

Once a skill is executed, one of the physically present entities with precisely known parameters as sensed by the robot must be assigned to the symbolic part description in the task model. Establishing a link between symbolic parts and the world is referred to as the **anchoring problem** [3]. This in particular includes deciding between multiple sensed entities that equally match an ambiguous part description (e.g. bolts of different sizes all being of type `bolt`). Related approaches perform anchoring with *local decisions* [4, 14, 18]. Ambiguity is here resolved in the scope of a skill without considering subsequent process steps, e.g. by choosing from all matching entities the one closest to the robot [14], or by drawing randomly [18]. However, such decisions can render the overall process infeasible (Fig. 2): Despite being suitable for the currently considered skill, an entity may be strictly required by some subsequent skill with more strongly constrained input parts. Choosing the "wrong" entity will thus lead to an error when trying to anchor this subsequent skill. Therefore, we propose an algorithmic procedure with *global decisions* which considers the constraints of all skills during the anchoring process.

All in all, our contribution is twofold: (i) We propose a task model and visual programming procedure with robot skills accepting parts families and flexible locations rather than definitely specified, uniquely identified parts as input parameters. (ii) We show a

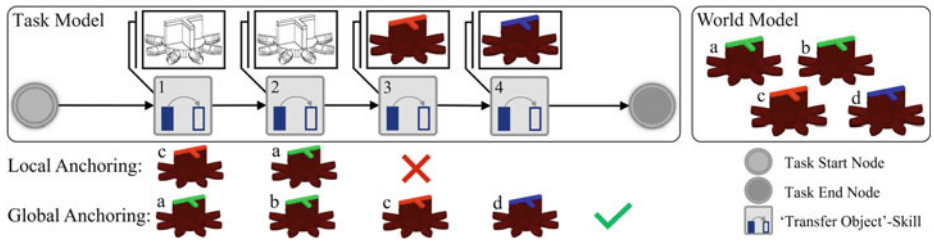


Fig. 2 Our task models may be underspecified, e.g. by skills accepting any kind of `gear` (1 and 2) for adaptation to sensed parts in a world model (a-d). Locally correct anchoring decisions, e.g. assigning `red_gear` c to skill 1, can render the process infeasible when subsequent skills have strictly specified input parts (3 and 4)

computationally efficient method for anchoring and executing such task models in unknown environments with ambiguous parts.

2 Our Approach

An overview of our approach is shown by Fig. 3. Users will first use a visual programming task editor to create a precedence graph model (Sect. 2.2) capturing different instances of the task (Sect. 2.3). After that, the robot workspace is prepared by supplying concrete parts. The task model provides partly underspecified information about the types and approximate locations of parts to be expected when executing the task (Sect. 2.1). From this information, a path to explore points of interest in the workspace with a camera attached to the robot hand is calculated. A world model is then built by active vision, i.e. by approaching each point of interest and performing object recognition. The world model enables the computational process of plan instantiation for the perceived situation in the workspace (Sect. 2.4): Detected entities in the world model are assigned to parts referenced in the task model with an *assignment planner* solving the anchoring problem. Together with the task model, the resulting assignment solution is passed to a task sequencer. The sequencer applies a scheduling algorithm to the task model and finishes skill parametrization by replacing underspecified parameters with precise information from the world model. The resulting operation sequence is finally passed to a skill execution engine. After task completion, further materials can be supplied, and the plan instantiation process can be re-iterated starting from the workspace exploration step without manually adapting the task model.

2.1 Part Types and Locations

We describe parts in terms of their type and location in the workspace. To this end, a *part type* is an entry taken from a tree-shaped part type ontology. This ontology is a required input to the approach. It captures "is-a"-relations between a set of nodes $O = \{o_1, o_2, \ldots, o_{|O|}\}$. Leaf nodes $P \subset O$ denominate *concrete part types* as which parts in the physical world can be classified. We assume a CAD model given for each $o \in O$ for the purpose of

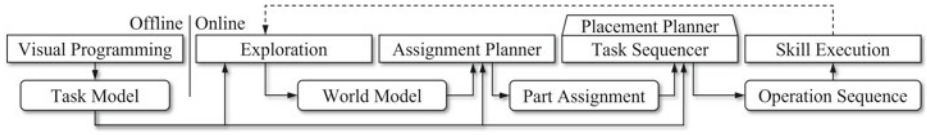


Fig. 3 Our approach adapts generalized task models emerging from a visual programming procedure by means of active workspace exploration, assignment planning, task sequencing, and skill execution

grasp and placement planning. When ascending from leaf nodes upwards towards the root node, encountered inner ontology nodes encode increasingly generic part descriptions. The ontology thus encodes parts families with an increasing level of generalization over part types. An example inspired by the benchmark domains used in our experiments is shown by Fig. 4. Here, different gear and conductor leaf part types are summarized under the more general terms `gear` and `conductor`. The approach is intuitively adapted to other domains by specifying a corresponding tree with several levels of generalized part types. Formally, the ontology is characterized by the function is_a : $O \times O \rightarrow \{\text{TRUE}, \text{FALSE}\}$ with $\text{is\_a}(o, o') = \text{TRUE}$ whenever $o = o'$ or o is a child of o'. In all other cases, $\text{is\_a}(o, o')$ is FALSE.

Regarding the *part location*, we distinguish two cases: A location can be known precisely and, hence, be specified by a rigid body transform ${}^{\text{w}}T_{\text{part}} \in \mathbb{R}^{4\times4}$ indicating the object translation and rotation with respect to some world frame w. This is e.g. the case for object recognition results, for parts provided on workpiece carriers etc. In the second case, a part location is not given precisely, but only within a certain tolerance. These two concepts can be captured by a unified formalization: Let $L = \{l_1, l_2, \ldots, l_{|L|}\}$ denote a set of locations relevant to the task. A location $l_i \in L$ may describe the precise position and orientation of some place where parts are usually located (e.g. the output slot of a parts feeder). Let $L^{\text{prec}} \subseteq L$ denote these precisely known locations, each associated with a rigid body transform $\text{pose}(l_i) \in \mathbb{R}^{4\times4}$ ($l_i \in L^{\text{prec}}$). In addition to these precisely known locations, elements of L may also describe a 2-dimensional area on the workbench surface, a 3-dimensional volume defining the interior of a box etc. We will see in Sect. 2.3 how L emerges from the visual programming process. For the planning process (Sect. 2.4), each location $l_i \in L$ is associated with a *location function* $\text{is\_at}_{l_i} : O \times \mathbb{R}^{4\times4} \rightarrow \{\text{TRUE}, \text{FALSE}\}$. These functions are designed to output $\text{is\_at}_{l_i}(o, {}^{\text{w}}T_{\text{part}}) = \text{TRUE}$ for a part type $o \in O$ and transformation ${}^{\text{w}}T_{\text{part}} \in \mathbb{R}^{4\times4}$ if and only if some part of type o with pose described by ${}^{\text{w}}T_{\text{part}}$ is at the location denominated l_i. Our system currently supports is_at_{l_i} functions for comparing equality of precise positions, and for checking whether parts lie in planar workspace areas considering

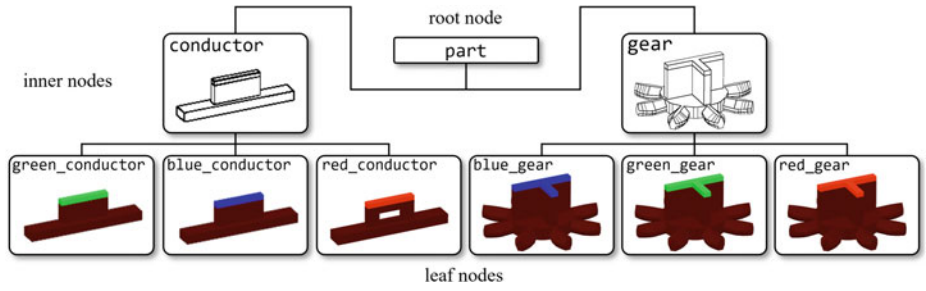


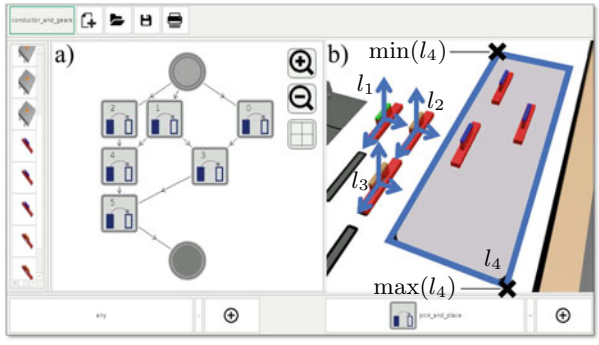
Fig. 4 A part ontology tree encodes "is-a"-relations to group different part leaf types into more generic type descriptions represented by inner tree nodes

their axis-aligned bounding boxes aabb(o) (Fig. 5). The formalism allows for integrating more complex location specifications in future work (e.g. spatial relations between parts).

2.2 Task Models with Degrees of Freedom

Our goal is programming tasks that can be adapted to product and process variety at execution time. To this end, we first define the notion of part templates which capture boundary conditions that parts used in a task must satisfy. A *part template* $p = (p^{\text{type}}, p^{\text{loc}})$ combines an arbitrary node $p^{\text{type}} \in O$ from the part type ontology with a location $p^{\text{loc}} \in L$. It describes a part with parameters that are possibly only partly known during the visual programming procedure, e.g. a conductor that may be either red, green, or blue and that lies at any position within a larger area on the workbench. Part templates enable task models with a certain degree of generality regarding part types and locations: In our framework, each task $(T, \prec_T)$ is composed of partially ordered *operations* $T = \{\tau_1, \tau_2, \ldots, \tau_{|T|}\}$. The partial order $\prec_T$ defines assembly precedence relations between operations, i.e. some operation $\tau_i \in T$ must be done before $\tau_j \in T$ $(i \neq j)$ if and only if $\tau_i \prec_T \tau_j$. This task model is well known from the assembly planning domain [10] and suited for flexible production settings. We further describe each operation with a pair $\tau_i = (p_i, l_i)$ of a part template p_i and a part goal location $l_i \in L$. The model thus covers any sort of operation where a part is transferred to a new location by the robot. This comprises basic pick-and-place actions as well as operations during which the transfer requires more sophisticated robot control (e.g. force-supervised gear meshing, see Sect. 3).

Task models as defined above are underspecified, and each part template must be anchored to a physical entity when the task is executed (Sect. 1). To this end, the robot builds a *world*



$\text{is\_at}_{l_1}(o, {}^{\text{w}}T_{\text{part}})$:
  return
    ${}^{\text{w}}T_{\text{part}} \approxeq \text{pose}(l_1)$
  ...
$\text{is\_at}_{l_4}(o, {}^{\text{w}}T_{\text{part}})$:
  $o_{\min} = {}^{\text{w}}T_{\text{part}} \cdot \min(\text{aabb}(o))$
  $o_{\max} = {}^{\text{w}}T_{\text{part}} \cdot \max(\text{aabb}(o))$
  return
    $o_{\min} \geq \min(l_4) \wedge$
    $o_{\max} \leq \max(l_4)$

Fig. 5 Our task editor (left) combines icon-based precedence graph modelling (**a**) with part creation in a virtual workspace (**b**). The modelling process outputs task models with associated operators to compare locations and part types (right)

model $W = \{\hat{p}_1, \ldots, \hat{p}_{|W|}\}$ containing all entities perceived on camera images. Entities are encoded by part states. Contrasting to part templates, *part states* $\hat{p} = (\hat{p}^{\text{type}}, \hat{p}^{\text{loc}})$ combine an ontology leaf node $\hat{p}^{\text{type}} \in P$ and a precise location $\hat{p}^{\text{loc}} \in L^{\text{prec}}$ as detected by object recognition. We say that an operation $\tau_i \in T$ may be applied to a part state $\hat{p} \in W$ if and only if $\hat{p}$ *satisfies* the part template p_i. Validation of this connection between part templates and states is achieved with a satisfies-function (Eq. 1).

$$\text{satisfies}(\hat{p}, p) = \begin{cases} \text{TRUE} & \text{if is\_a}(\hat{p}^{\text{type}}, p^{\text{type}}) \ \wedge \ \text{is\_at}_{p^{\text{loc}}}(\hat{p}^{\text{type}}, \text{pose}(\hat{p}^{\text{loc}})) \\ \text{FALSE} & \text{otherwise} \end{cases} \tag{1}$$

2.3 Visual Programming

Users create task models by interacting with a graphical editor shown in Fig. 5. To this end, it is first necessary to specify part templates for each part to be used during the task. A new template can be added by choosing its part type and initial part location. The user is in charge of selecting from the part type ontology appropriately so that the desired level of task generalization is reached. The selection of locations is supported by a virtual representation of the workspace. In the virtual workspace, a *workspace layout* as introduced in our prior work [16] offers pre-defined regions to be chosen as part locations (e.g. l_4 in Fig. 5, left). For each area defined by the layout, a location function based on the area corner vertices is instantiated and added to the location set L (Sect. 2.1). If the user prefers to specify part poses precisely (l_1, l_2, l_3 in Fig. 5), additional location functions are defined by corresponding precise poses. Having specified all parts, pick-and-place operations may be added. Finally, the operations are connected with precedence relations using the icon-based editor component. Currently, the system is based on a single pick-and-place skill – suitable control algorithms are derived from annotations to the part type ontology (e.g. force-supervised `gear` meshing vs. position-controlled placement of our benchmark `conductor` parts). Yet further classes of skills, e.g. for visual inspection or presentation of parts to the user for collaborative steps, can be added in the future.

2.4 Plan Instantiation

Having modelled a task with operations $T = \{\tau_1, \ldots, \tau_{|T|}\}$, users need to prepare the workspace by supplying necessary parts to the robot. After an active vision exploration procedure (see [2] for an overview of applicable methods), the robot has all detected parts stored in its world model $W = \{\hat{p}_1, \ldots, \hat{p}_{|W|}\}$. The next step is solving the anchoring problem as introduced in Sect. 1, i.e. mating each part template p_i of operation τ_i with a part state $\hat{p}_j$ so that satisfies$(\hat{p}_j, p_i)$ holds. Assuming that the user has provided at least one part for each operation ($|W| \geq |T|$), this means $\mathcal{O}(|W|!)$ possible assignments. Enumerating and

testing those to find a valid solution, clearly, is a computationally infeasible combinatorial problem even for small $|W|$. However, we can apply efficient combinatorial optimization algorithms to this unbalanced assignment problem, e.g. the well-known Kuhn-Munkres algorithm [12] with $\mathcal{O}(|W|^3)$ runtime complexity:

Let $\mathbf{C} = (c_{i,j})$ denote a $|T| \times |W|$ cost matrix with a row for each part template and a column for each part state. Any wrong assignment of $\hat{p}_j$ to p_i is modelled to have infinite costs, whereas a correct assignment has no costs, i.e.

$$c_{i,j} = \begin{cases} 0 & \text{if satisfies}(\hat{p}_j, p_i) \\ \infty & \text{otherwise} \end{cases}, i \in \{1, \ldots, |T|\}, j \in \{1, \ldots, |W|\}. \tag{2}$$

Given $\mathbf{C}$, combinatorial optimization computes an optimal, injective assignment $f : \{1, \ldots, |T|\} \rightarrow \{1, \ldots, |W|\}$ which minimizes the total assignment costs $\sum_i c_{i,f(i)}$ ($i \in \{1, \ldots, |T|\}$). In our case, f says that part template p_i of operation τ_i must be associated with part state $\hat{p}_{f(i)}$ to incur the minimum cost assignment. By construction of $\mathbf{C}$, any solution involving a wrong assignment (cf. Fig. 2) leads to infinite overall costs. This means in practice that the user has not supplied all required parts to the workspace—in this case, our system outputs an error message to inform about missing parts. By contrast, a solution f with 0 overall costs means that each part template was matched with a suitable entity in the workspace. The process can then proceed to the task sequencing step.

The task sequencing procedure prepares a fully specified sequence of operations to be executed by the skill engine. For each operation $\tau = (p, l)$, a suitable input entity matching p is known from the above assignment f. We further use a grid-based placement planner that determines precise part goal locations whenever the operation goal location l is an area. Finally, the precedence graph is transferred into a sequence that complies with all "earlier-later" relations. The fact that we are using a graph structure as task model opens a range of future possibilities here: Aside from searching for an operation sequence that optimizes energy consumption or other secondary criteria, planning of collaborative action with a human-robot scheduler would also be feasible at this point in the process.

3 Experimental Validation

We have modelled four benchmark tasks (Sect. 2.3) which are designed to illustrate specific aspects of product and process variety (Fig. 6a): Product variety is represented by task S1, in which gears of arbitrary types (`red`, `blue`, `green`, cf. Fig. 4) are assembled with force-supervised robot control. Task S2 is a kitting task, where a connector of each type is added to a bundle of three. Tasks S3 and S4 replicate assembly tasks of electrical circuits with a serial/parallel connection. The tasks S2–S4 use region-based initial locations, thus enabling convenient part feeding by the user. Task S2 furthermore allows for the bundle to be placed anywhere within an area. We have executed each task with different workspace configu-

rations (e.g. S1 with different part types, S4 with orderly or arbitrarily placed connectors, cf. Fig. 1). Online adaptation and task execution in these differing settings was achieved successfully.

Moreover, a theoretical comparison of the effort needed for adaptation with our approach versus the traditional re-programming method was conducted. We say that a production cycle consists of executing a task N times, i.e. finishing N instances of a product. By introducing the *flexibility demand ratio* $FD = \frac{1}{N}$, we characterize the manufacturing setting, i.e. traditional mass production with hardly any adaptations for $FD \to 0$, decreasing lot sizes for $FD \to 1$, and one-off products for $FD = 1$. The adaptation effort per cycle of our approach depends on N, as each program execution is preceded by exploration and assignment planning—re-programming effort is not required during a cycle as the task models for S1–S4 have covered all necessary adjustments. During the experiments with our benchmark tasks, an exploration time of about 9 s was measured whereas the planning time was negligible. By our definition, the effort per cycle for adaptation by visual re-programming is independent of N and therefore constant. However, the re-programming time including loading and saving the task model depends on the degree of necessary changes. We have considered three cases where only one operation or corresponding part (minimum effort); half of the involved parts (medium effort); or all parts (maximum effort) need to be adjusted in the task model between consecutive cycles. Representative durations of these three re-programming types have been gathered by observing an expert operate our task editor (min. $\approx$ 31 s; med. $\approx$ 80 s; max. $\approx$ 110 s).

Figure 6b compares the time allocated for adaptation within a production cycle depending on FD. In general, our approach achieves better results than manual re-programming in highly flexible domains, i.e. for higher FD values, since task variants are widely encoded in

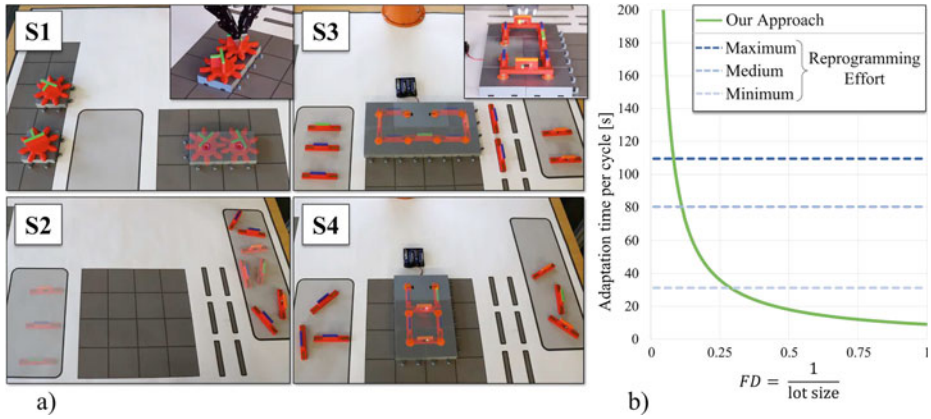


Fig. 6 Our experiments comprise different benchmark tasks S1-S4 (**a**, goal states are rendered transparently). Adaptation time measurements enable a comparison of our approach and manual re-programming for different lot sizes (**b**)

the task model. In particular, it performs better for lot sizes of three or less, even when considering re-programming with minimum effort. In other words, less adaptation effort is needed with our approach compared to manual re-programming for finishing three products—this confirms our hypothesis regarding economical efficiency (Sect. 1). For medium and maximum re-programming effort, this amortization threshold shifts towards larger lot sizes. However, the effort for exploring the workspace before each task iteration renders re-programming more efficient in mass production settings with relatively few changes. These quantitative results must of course be interpreted within the limits of our benchmark tasks. Yet, our analysis illustrates qualitative relationships that are transferable to other scenarios and applications.

4 Conclusion and Future Work

In this paper, we have contributed a visual programming and robot task execution approach that incorporates product and process variety. For this, part templates are specified as input to robot skills in terms of approximate locations and generalized parts families. This leads to partly ambiguous, underspecified task models capturing a set of task variants. Adaptation to concrete parts is achieved online by workspace exploration and combinatorial optimization to anchor ambiguous part templates to perceived concrete parts. Our experiments with a set of characteristic benchmarks show how this approach helps to reduce the (re-)programming effort of robots in flexible manufacturing settings.

We will address several limitations of the approach in future work: Currently, the task structure and number of processed parts are fixed. Further task variety could be achieved by augmenting the task model with constructs as loops for situation-dependent repetition of operations. Furthermore, we will extend the approach towards human-robot co-working by integrating multi-agent scheduling. Finally, our concept needs a comparison with other visual programming systems to evaluate the impact of generic part and location descriptions on usability.

Acknowledgements This work was partly funded by the Deutsche Forschungsgemeinschaft (DFG) under grant agreements He2696/20 (FlexCobot) and He2696/18 (VerbBot).

References

1. Bastidas-Cruz, A., Heimann, O., Haninger, K., Krüger, J.: Information requirements and interfaces for the programming of robots in flexible manufacturing. In: Annals of Scientific Society for Assembly, Handling and Industrial Robotics, pp. 183–192 (2020)
2. Chen, S., Li, Y., Kwok, N.: Active vision in robotic systems: a survey of recent developments. Int. J. Robot. Res. **30**(11), 1343–1377 (2011)

3. Coradeschi, S., Saffiotti, A.: An introduction to the anchoring problem. Robot. Auton. Syst. **43**(2–3), 85–96 (2003)
4. Darvish, K., et al.: A hierarchical architecture for human-robot cooperation processes. IEEE Trans. Rob. **37**(2), 567–586 (2021)
5. De Lit, P., Danloy, J., Delchambre, A., Henrioud, J.-M.: An assembly-oriented product family representation for integrated design. IEEE Trans. Robot. Autom. **19**(1), 75–88 (2003)
6. Dietz, T., et al.: Programming system for efficient use of industrial robots for deburring in SME environments. In: German Conference on Robotics, pp. 428–433. Munich (2012)
7. ElMaraghy, H.: Changing and evolving products and systems – models and enablers. In: ElMaraghy, H. (ed.). Changeable and Reconfigurable Manufacturing Systems, pp. 25–45. Springer, London (2009)
8. Giménez, D., Vegetti, M., Leone, H.P., Henning, G.P.: PRoduct ONTOlogy: defining product-related concepts for logistics planning activities. Comput. Ind. **59**(2–3), 231–241 (2008)
9. Hegge, H., Wortmann, J.: Generic bill-of-material: a new product model. Int. J. Prod. Econ. **23**(1–3), 117–128 (1991)
10. Homem de Mello, L., Sanderson, A.: AND/OR graph representation of assembly plans. IEEE Trans. Rob. Autom. **6**(2), 188–199 (1990)
11. Jiao, J., Tseng, M.: An information modeling framework for product families to support mass customization manufacturing. CIRP Annals **48**(1), 93–98 (1999)
12. Munkres, J.: Algorithms for the assignment and transportation problems. J. Soc. Ind. Appl. Math. **5**(1), 32–38 (1957)
13. Nottensteiner, K., et al.: A complete automated chain for flexible assembly using recognition, planning and sensor-based execution. In: International Symposium on Robotics (ISR), pp. 365–372. Munich (2016)
14. Paxton, C., Hundt, A., Jonathan, F., Guerin, K., Hager, G.: CoSTAR: instructing collaborative robots with behavior trees and vision. In: IEEE International Conference on Robotics and Automation, pp. 564–571. Singapore (2017)
15. Pedersen, M., Nalpantidis, L., Andersen, R., Schou, C., Bøgh, S., Krüger, V., Madsen, O.: Robot skills for manufacturing: From concept to industrial deployment. Rob. Comput.-Integr. Manuf. **37**, 282–291 (2016)
16. Riedelbauch, D., Henrich, D.: Fast Graphical Task Modelling for Flexible Human-Robot Teaming. In: Int. Symposium on Robotics, pp. 414–419. Munich (2018)
17. Schoen, A., Henrichs, C., Strohkirch, M., Mutlu, B.: Authr: A Task Authoring Environment for Human-Robot Teams. In: ACM Symposium on User Interface Software and Technology, pp. 1194–1208 (2020)
18. Senft, E. et al.: Situated Live Programming for Human-Robot Collaboration. In: ACM Symposium on User Interface Software and Technology, pp. 613–625 (2021)
19. Steinmetz, F., Wollschläger, A., Weitschat, R.: RAZER – A HRI for Visual Task-Level Programming and Intuitive Skill Parameterization. IEEE Rob. Autom. Lett. **3**(3), 1362–1369 (2018)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.