The final publication is available at Springer via
https://doi.org/10.1007/978-3-319-61276-8_21

Sensor-based loops and branches for playback-programmed robot systems

Michael Riedl, Eric M. Orendt, and Dominik Henrich

Lehrstuhl für Angewandte Informatik III
Universität Bayreuth, D-95440 Bayreuth, Germany
michael.riedl@uni-bayreuth.de,
http://robotics.uni-bayreuth.de

Abstract. Usually, programming robot systems is expensive and complex and only profitable for companies with big lot sizes. Hence these systems do currently not play a big role in small or medium sized enterprises. This work extends the programming paradigm published in [1] that is based on the playback programming method, so that also sensor information can be used to generate more complex robot programs by means of playback programming in addition to the already existing functionality. This is achieved by developing a concept for sensor-based loops and branches that fits well to the programming paradigm. Finally, the enhanced programming system is evaluated in a user study with experts and non-experts with respect to its intuitiveness. The user study is divided into a part that tests the user interface and a part that evaluates the system as a whole, so that possible weak spots of the system or the user interface can be detected and can be taken into account in further work with this programming system.

Keywords: playback programming, sensor-based control structures, intuitive robot programming

1 Introduction

This article deals with sensor-based control structures for playback-programmed robot systems. Therefore the programming system from [1] is extended so that it can also deal with control structures, which in our case are loops and branches. An illustration of the programming system is shown in Figure 1. On the one hand, branches are helpful, if a certain action should be executed depending on a detected object, e.g. sorting objects depending on their color. On the other hand, loops are useful in a playback programming environment if a task has to be executed as long es a certain object is detected by the system, e.g. picking up parts from a conveyor belt and placing them in a box as long as parts are arriving.

At the beginning, this paper gives an overview over the related work that is already done in this field, the next part deals with sensor-based control structures in general and loops and branches in particular and the last section shows the

The final publication is available at Springer via
https://doi.org/10.1007/978-3-319-61276-8_21

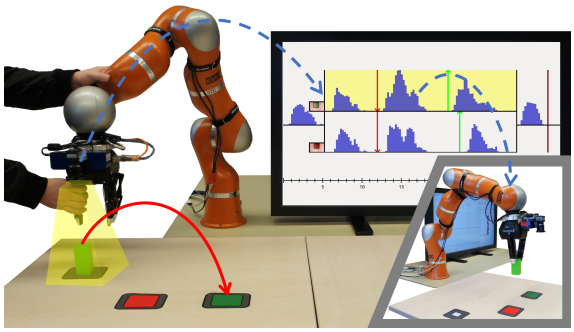

Fig. 1: An illustration of the programming system for sensor-based playback programming. It shows the programming aspect, the representation aspect and the reproduction aspect. The blue arrows show the transformation from the demonstration layer to the representation layer and from the representation layer to the reproduction layer and the red arrow shows the task that should be executed by the robot.

outcome of a user study that has been done to evaluate the intuitiveness and usability of the developed programming system.

2 Related work

Different approaches to robot programming are discussed in [2]. It distinguishes between on-line- and off-line programming approaches, where off-line programming techniques are textual programming, graphical programming and programming using CAD and on-line programming techniques are programming by demonstration and playback programming. The *playback programming* approach is distinguished to other approaches by the fact, that robot manipulators are programmed by an operator that guides them through the desired trajectory. This guiding can be done manually, with the help of a teach pendant or a master robot. After the trajectory is recorded, the robot plays back exactly the same path as taught. In [3], robot programming is divided into on-line programming, off-line programming and programming using augmented reality. The work further divides the on-line programming approaches into sensor guided on-line programming and operator assisted on-line programming. The off-line programming part compares different off-line programming systems and the programming using augmented reality part describes possible ways of integrating augmented reality into robot programming systems. Another survey on robot programming techniques can be found in [4]. It divides robot programming in automatic programming methods and manual programming methods.

Since we use playback programming, we review the related work in this field, too. In [5] the programming technique is firstly described by the terms *teaching by showing* and *guiding*, which is the simplest form of the playback programming where the trajectory is taught by the operator and exactly played back afterwards. An extension, called *extended guiding*, enables the operator to record

The final publication is available at Springer via
https://doi.org/10.1007/978-3-319-61276-8_21

the trajectory relatively to a coordinate system and then change the coordinate system when the operator is going to play back the trajectory. The first appearance of the term playback programming is in [6], where it is described as a programming method where the robot is guided through the desired trajectory either by hand, with the help of a master robot or with a teach pendant. The playback programming technique counts among on-line programming methods as described in [2] and [3] and among manual programming methods described in [4].

A last big part of research that is related to our programming approach is the intuitive robot programming section. The EU project SMErobot developed different intuitive robot programming methods, e.g. programming a robot by drawing a path onto the workpiece in [7], by using Programming by Demonstration with post processing for welding in [8], by using a combination of voice recognition and guiding in [9] and by using a combination of voice recognition and a accelerometer-based hand-held device in [10]. Besides SMErobot, another article that unites playback programming with intuitive robot programming can be found in [11], where a welding robot for shipyards is programmed by using the walk-through programming technique that is a synonym for playback programming technique. One last approach for intuitive robot programming is shown in [12] and [13], where a one-shot programming by demonstration approach is proposed which is capable of adapting its motion to slightly different scenarios. This approach is evaluated in [14] regarding intuitiveness and it is used to compare the outcomes of the user study made with our programming system.

The achieved results of this paper are based on a playback programming system using Gantt-charts [1], which is briefly explained in the next section.

3 Playback programming using Gantt-charts

This part gives a short overview on the programming system from [1] that we will extend. The central part of the programming system is the playback programming technique to allow also non-professional users to program robots. It aims to eliminate different disadvantages of this technique like the non-editability of recorded trajectories. Besides that, the visual representation of the recorded trajectories is also a central aspect of the system. A Gantt-chart like illustration is used to show the operator a schematic representation of the recorded trajectories. The representation from [1] was slightly modified within the scope of this work to give the user more detailed information on the motion of the robot. The block-representation was replaced by a velocity profile that displays the velocity of the tool center point for each configuration in the trajectory, so the user does not only see when the robot is moving but also how fast it is moving. Illustrations of the Gantt-chart representation are shown in Figure 1, Figure 2 and Figure 3. Another important feature that was introduced in [1] is the edit functionality for playback-programmed trajectories. It enables the user to manipulate the already recorded trajectory in the Gantt-chart with drag-and-drop, copy, paste, cut and delete. This programming paradigm is extended within the scope of this work.

The final publication is available at Springer via
https://doi.org/10.1007/978-3-319-61276-8_21

4 Sensor-based loops and branches

Depending on a sensor input, loops and branches add more possibilities to the already existing playback programming system. To give a formal description on how to add loops and branches to playback programmed trajectories, a definition of the term trajectory is needed. A *trajectory* $\mathbf{q}(t)$ of a robot with N joints is for our purposes defined as $\mathbf{q}(t) = [q_1(t), q_2(t), \cdots, q_N(t)]^T$ with the joint angles $q_i(t)$ of the i-th joint at time t, an integer amount of joints $N \in \mathbb{N}$, and $1 \leq i \leq N$. Hence, for every point in time $t > 0$, an explicit configuration for the N joints of the robot exists.

4.1 Loops

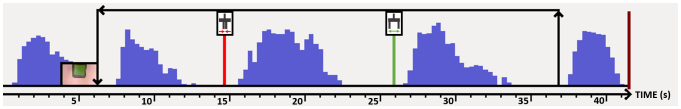

Fig. 2: The visual representation of a loop in the Gantt-chart. The pictogram shows the reference stimulus that is the loop condition. Additionally the position of it shows, whether it is a while- or a do-while-loop.

We use the concept of loops in imperative programming languages and extend it to the playback programming technique. The visual representation of loops in the Gantt-chart is shown in Figure 2. A loop consists of a loop condition and a loop trunk. The different types of loops emerge from the different points in time, when the loop condition is evaluated. Inserting a loop into a trajectory $\mathbf{q}(t)$ seems easy at first sight, one just has to define the range $[t_1; t_2]$ with $t_1 \leq t_2$ as the loop trunk and add the loop condition, which is in our case the comparison between an image that was taken at programming time and an image that is taken at runtime, either at the beginning (t_1) or at the end (t_2) of the loop trunk. But a problem accrues when the execution of the trajectory comes to an end of a loop trunk. Then it would have to jump back to the beginning of it. This motion is not recorded, so a transfer motion needs to be inserted into the trajectory. In [1], different methods for generating transfer motions within playback programmed trajectories are shown. Another problem that accrues when camera images are used as sensor input is, that the robot has to stand still while the image is taken. Since the loop condition is always evaluated at the beginning or the end of the loop trunk and images are only taken at these points, we define as a precondition, that the velocity of all joints has to be 0 at both ends of the loop trunk. Therefore, $\dot{q}_i(t_1) = \dot{q}_i(t_2) = 0$ must apply for $1 \leq i \leq N$.

In addition, it has to be dealt with the loop condition. All three loop types, i.e. for-loop, while-loop and do-while-loop are represented by this concept. For the for-loop, a counter is used that is compared with the predefined maximum of loop iterations and afterwards incremented before the execution of the loop. Therefore, the loop-counter and the predefined amount of iterations is attached to the trajectory $\mathbf{q}(t)$ at point t_1. While and do-while loop differ from each

The final publication is available at Springer via
https://doi.org/10.1007/978-3-319-61276-8_21

other in the point, where the loop condition is evaluated. For the while-loop, the loop-condition and therefore the point of evaluation is attached to $\mathbf{q}(t)$ at point t_1, so that it is evaluated before the execution of the loop trunk, and for the do-while-loop, it is attached to $\mathbf{q}(t)$ at point t_2, so that it is evaluated after the execution of the loop trunk. For the evaluation of the loop-condition, a compare function is used to determine whether the stimulus is similar to the prerecorded one or not. This function is defined as

$$f(S_1, S_2) = \begin{cases} \textit{true}, \text{ if } S_1 \text{ and } S_2 \text{ similar} \\ \textit{false}, \text{ else} \end{cases}$$

with stimuli S_1 and S_2. It compares both stimuli and returns a boolean value.

4.2 Branches

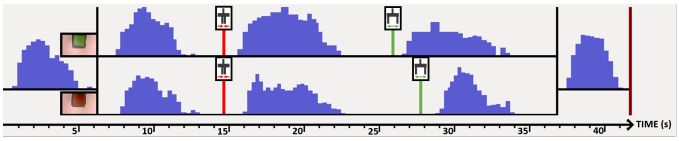

Fig. 3: The visual representation of a branch in the Gantt-chart of the programming system. The small pictograms on the left of each branch show the reference stimulus for the branch.

Analogous to the loops, we used the branch concept of imperative programming languages to find a concept for branches in playback-programmed trajectories $\mathbf{q}(t)$. The visual representation of branches in the Gantt-chart is shown in Figure 3. Let t_b be the point in time where the branch should be inserted into $\mathbf{q}(t)$. Then, the constraint $\dot{\mathbf{q}}(t_b) = 0$ needs to be fulfilled for the same reasons as for the loop constraint. Branches consist of N conditions $C \in \{C_1, C_2, \cdots, C_N\}$ and $N+1$ trajectories $T \in \{T_1, T_2, \cdots, T_N, T_d\}$, where T_d is the optional default trajectory. A branch $B = (C, T)$ is a tuple of a condition and a trajectory, where the i-th condition is connected to the i-th trajectory. No condition exists for T_d, because it is executed if none of the N conditions evaluate to *true*.

A condition C_i consists of a reference stimulus that is compared to the stimulus that is received during runtime with the help of the compare function defined in subsection 4.1. The branch that should be executed is the branch with the highest similarity value of all branches whose conditions evaluate to *true*. If no condition evaluates to *true*, the default branch T_d is executed. If T_d does not exist, the whole branching is skipped and $\mathbf{q}(t)$ is executed with $t > t_b$.

5 User study

The intuitiveness of the presented programming system was evaluated in a user study from the 17th until the 25th of May 2016 at our lab with 20 participants. The evaluation was done with the methods and questionnaires published in [14] to get a result that is comparable with other studies.

The final publication is available at Springer via
https://doi.org/10.1007/978-3-319-61276-8_21

5.1 Evaluation

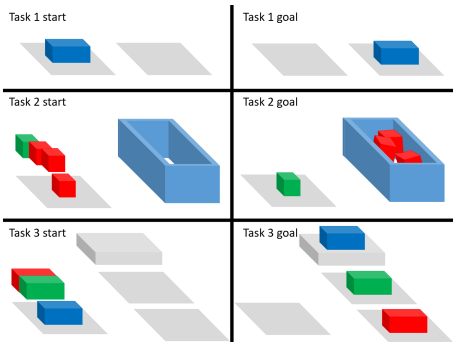

Fig. 4: The three tasks that had to be processed by the participants illustrated schematically as start and goal condition.

The presented programming paradigm is implemented in a prototypical way to test and to evaluate the concept regarding intuitiveness. Every participant had to fulfill three tasks that are shown in Figure 4 and in a video[1]. To generalize from a specific application domain, all tasks are designed to be fulfilled with colored cuboid. The first task is a simple pick-and-place operation to make the participant familiar with the programming system. The participant has to grasp the block with the robot and place it at the marked goal area. The core of the second task is a sensor-based loop. The participant should demonstrate a motion that grasps the red block and throws it into a box as long as a red cube lies at the start area. A green cube is used as a stop-block. As a last task, the participants should implement a branching with three branches depending on the color of the cuboid.

Three questionnaires and one observation sheet have to be filled out for every participant, where the first questionnaire has to be answered before the processing of the tasks. The second questionnaire is divided into two parts to evaluate the graphical user interface (GUI) and the programming system as a whole separately. It is filled out separately for each task while the task is processed. In parallel, the observer fills out the observation sheet. The third questionnaire has to be answered by the participant after all tasks are fulfilled. The detailed procedure and goals of the questionnaires can be found in [14].

5.2 Outcomes

All 20 participants were between 20 and 36 years old with a mean value $M = 25.5$ and a standard deviation $SD = 4.3$. They were divided into two groups, experts and non-experts, whereas experts are defined as people that already worked with robots and non-experts are defined as people that have never done anything with a robot. With this definition, both groups consisted of 10 participants.

Since EN ISO 9241 and IBIS [15] define the intuitiveness of a system as a combination of effectiveness, efficiency and satisfaction, all three aspects were evaluated within the scope of this work. The **effectiveness** was examined for both, the GUI and the programming system separately. A value between 0 and

[1] http://www.ai3.uni-bayreuth.de/resypub/files/riedl2017a.Sensorbased.loops.and.branches.for.playbackprogrammed.robot.systems.mp4

The final publication is available at Springer via
https://doi.org/10.1007/978-3-319-61276-8_21

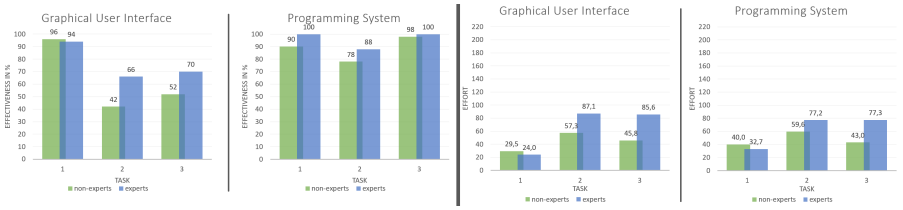

Fig. 5: The left two diagrams show the results of the measuring of the effectiveness for each task and group divided into GUI and programming system in a whole. High values represent high effectiveness. The right two diagrams show the results of the measuring of the effort for each task and group divided into GUI and programming system in a whole. Low values represent low effort.

100 is the result for the effectiveness, whereas a high value represents a high effectiveness. The results are shown in the left two diagrams in Figure 5. The chart shows, that the average effectiveness is higher for experts than for non-experts. The overall mean value for the effectiveness of the GUI over all groups and tasks is 70%, for the programming system it is 92.3%. This shows, that, the GUI has a lack of effectiveness, especially for the tasks two and three, but the programming system in a whole has already a good effectiveness.

The **efficiency** was also raised for the GUI and the programming system separately. The results were values between 0 and 220, where lower values correspond to a lower effort and therefore a higher efficiency. The right two diagrams in Figure 5 show the results. The mean value over all groups and tasks was $M = 54.9$ ($SD = 47.5$) for the GUI and $M = 55.0$ ($SD = 39.5$) for the programming system. Since the standard deviation is relatively big, one can see, that the effort is perceived differently from participant to participant, but in average, the mean value is literally translatable to *a bit to somewhat exhausting*.

The **satisfaction** was evaluated with the QUESI-questionnaire from IBIS [15] in five categories. All categories have values between 1 and 5, whereas higher values represent a higher satisfaction. The results are shown in Figure 6. The mean value over all categories and groups is $M = 3.6$ ($SD = 0.4$). According to IBIS, this value has to be compared to other studies using QUESI, so that a statement on the satisfaction can be made. In [14], the programming by demonstration approach from [12] and [13] was evaluated with the same method and it reached a QUESI-value of 3.6, which is the same as the one our system received. This can be interpreted in a way, that our developed programming system is as satisfying and intuitive as the one in [12] and [13].

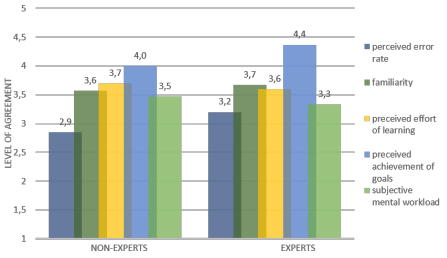

Fig. 6: The rated level of agreement of both groups for each of the five criteria of the QUESI-questionnaire.

6 Conclusion

The contribution of this work is the extension of the playback programming paradigm from [1] by the aspect of sensor-based loops and branches. To achieve this, the concepts of loops and branches in imperative programming languages is applied to the Gantt-chart represented and playback-programmed trajectories. The user study shows, that the developed programming system can be seen as intuitive compared to other intuitive robot programming approaches.

Acknowledgments. This work has partly been supported by the Bayerische Forschungsstiftung (BFS) under the project FORobotics.

References

1. M. Riedl, J. Baumgartl, and D. Henrich. Editing and synchronizing multi-robot playback programs. *47th International Symposium on Roboics - ISR2016*, 2016.
2. M. Hägele, K. Nilsson, and J. N. Pires. Industrial Robots. In *Springer Handbook of Robotics*. Springer, 2008. 978-3-540-23957-4.
3. Z. Pan, J. Polden, N. Larkin, S. van Duin, and J. Norrish. Recent progress on programming methods for industrial robots. *Robotics and Computer Integrated Manufacturing*, 28(2):87–94, 2012.
4. G. Biggs and B. Macdonald. A survey of robot programming systems. In *Proc. of the Australasian Conference on Robotics and Automation, CSIRO*, page 27, 2003.
5. T. Lozano-Perez. Robot Programming. *Proceedings of the IEEE*, 71(7):821–841, 1983.
6. D. Ardayfio. *Fundamentals of Robotics*. Mechanical Engineering. Marcel Dekker, Inc., 1987. 978-0824774400.
7. J. N. Pires, T. Godinho, and R. Araujo. Using digital pens to program welding tasks. *Industrial Robot: An International Journal*, 34(6):476–486, 2007.
8. C. Meyer, R. Hollmann, C. Parlitz, and M. Hägele. Programmieren durch Vormachen für Assistenzsysteme - Schweiß- und Klebebahnen intuitiv programmieren. *it Information Technology*, 49(4):238–246, 2007.
9. J. N. Pires et al. Programming-by-demonstration in the coworker scenario for SMEs. *Industrial Robot: An International Journal*, 36(1):73–83, 2009.
10. P. Neto et al. High-level programming and control for industrial robotics: using a hand-held accelerometer-based input device for gesture and posture recognition. *Industrial Robot: An International Journal*, 37(2):137–147, 2010.
11. M. H. Ang Jr., W. Lin, S.-Y. Lim. A walk-through programmed robot for welding in shipyards. *Industrial Robot: An International Journal*, 26(5):377–388, 1999.
12. C. Groth and D. Henrich. One-shot robot programming by demonstration by adapting motion segments. In *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1068–1075, Dec 2014.
13. C. Groth and D. Henrich. One-shot robot programming by demonstration using an online oriented particles simulation. In *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 154–160, Dec 2014.
14. E. M. Orendt et al. Robot programming by non-experts: Intuitiveness and robustness of one-shot robot programming. In *2016 IEEE 25th International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2016.
15. A. Wegerich, D. Löffler, and A. Maier. Handbuch zur IBIS Toolbox. 2012.