Editing and synchronizing multi-robot playback programs

Michael Riedl, Universität Bayreuth, michael.riedl@stmail.uni-bayreuth.de
Johannes Baumgartl, Universität Bayreuth, johannes.baumgartl@uni-bayreuth.de
Dominik Henrich, Universität Bayreuth, dominik.henrich@uni-bayreuth.de

Abstract

Controlling and programming multi-arm robots is cost-intensive and complex. As a result, such systems currently do not play a big role for small and medium sized enterprises. This paper tackles the task of intuitive multi-robot playback programming and introduces a new programming paradigm that allows teaching of both, single arm and multi-arm robots with only little knowledge about robotics. To achieve that, the common playback programming approach is extended to more than one robot and an editing mechanism is added to allow a graphical post processing of the generated trajectories. This editing mechanism contains both, manual editing the trajectories and temporal synchronizing the trajectories of the robots. Additionally, an example on how to use the new programming paradigm is given and also possible scenarios on how to use it and its limits are shown.

1 Introduction

Programming industrial robots is a tedious task that needs a lot of technical knowledge about the robot manipulator itself and the surrounding environment. This holds true especially for programming multi-arm robots. As a result, small or medium sized enterprises do usually not use robots to fulfill certain tasks, because the effort to program them bears no relation to the benefit they get from it. Usually it takes too long to teach tasks, or well trained personnel is needed to program industrial robots. The goal we aim for is, to simplify the programming of both, single-arm and multi-arm robots, so that no specialist is needed anymore.

In this paper, we present a new programming paradigm that enables to teach multi-arm robots by means of playback programming and afterwards manipulate the recorded trajectories. An illustration of the developed system is shown in Figure 1. The *playback programming* approach is distinguished to other approaches by the fact, that the robot manipulators are programmed by an operator that guides them through the desired trajectory. This guiding can either be done manually, with the help of a teach pendant or with a master robot. After the trajectory is recorded successfully, the robot plays back exactly the same path as taught [1]. A disadvantage of the playback programming approach is, that it is hardly possible to post process the recorded trajectory, e.g. delete unnecessary parts of it, and therefore the whole trajectory has to be reprogrammed if there is a mistake in the recorded one. To tackle this disadvantage, a visual representation of the recorded trajectories and an editing mechanism is introduced, that allows the user to edit and synchronize the taught trajectories afterwards. The visual representation is similar to a Gantt diagram. It has one timeline per robot component inside a common time scale. A *robot component* is that part of a system, that can be actuated independently from the other parts. A robot manipulator, a gripper or a conveyor are all examples for robot components, but the single joints of a robot manipulator are no robot components. This form of programming makes it easier for non-expert users to teach-in and synchronize multi-arm robots and therefore makes it interesting for small and medium sized enterprises. Even for expert users, the programming becomes much easier.

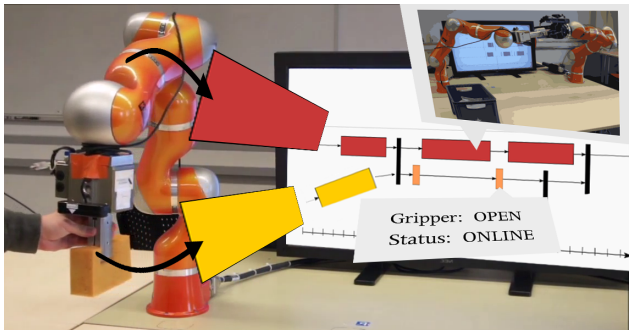

Figure 1 An illustration of the developed programming system, including a user-guided robot arm, a graphical representation of the recorded robot movements, and a synchronized two-arm task execution (left to right).

In the rest of the paper, a short overview over the related work to this topic is given. Afterwards, the new programming paradigm is described, which includes a method to program multi-arm robots by means of playback programming and then to edit and synchronize the generated trajectories in an intuitive way. In the last section, a short experiment shows, how to program an example robot sys-

tem, which tasks can be done by using our programming paradigm and what the limits are of our approach.

2 Related Work

There is various work about the playback programming method and multi-robot programming, but there is hardly any previous work about an intuitive way to edit and synchronize playback programs for multiple robots, that we want to introduce in this paper. Many different papers deal with robot programming itself and playback programming in particular.

In [2], a good overview over different programming methods for industrial robots is given. The overview distinguishes between online and offline programming techniques. If the program is created by directly teaching the robot, then the programming method is called *online programming*, whereas a technique that creates the robot program independent from the physical robot is called *offline programming*. It also explains the most common ones, e.g. textual programming methods, graphical programming methods or programming by demonstration. *Textual programming* methods are offline programming methods that use a textual programming language to create a robot program. *Graphical programming* methods are also offline programming methods, but they are using a CAD interface instead of a textual programming language to specify robot motions. The keyword *programming by demonstration* describes an online programming technique where the programmer teaches the robot by showing it the task once or several times while the robot observes the demonstration by means of sensors.

Pan et al. review current research progress on programming methods for industrial robots in [3]. They distinguish three major approaches: online programming, offline programming and programming using augmented reality. The online programming part deals with two programming methods, namely the operator assisted online programming and the sensor guided online programming. The operator assisted online programming techniques are all very similar to the playback programming approach, i.e. they all have in common, that one operator programs the robots either by direct guiding it or by using a teach pendant. In contrast to that, in sensor guided online programming, the operator does not guide the robot directly. Instead, it uses sensory information to generate the robot program. One example for that method is, that the programmer draws the desired path with a marker onto the work piece and the robot manipulator automatically follows the drawn path with its tool center point. The offline programming part shows the necessary steps to create an offline programming system and reviews afterwards already existing offline programming systems, both, generic ones and from robot manufacturers. Finally, the programming using augmented reality is discussed by showing different research projects that take this approach. *Augmented reality* is a method to extend a real world scene image by displaying computer generated 3D models in it. In the field of robot programming, it is used to achieve a new form of offline programming by displaying the virtual environment around the real work piece. Thus the work piece does not have to be modeled in the virtual environment. Another scenario to use augmented reality in robot programming is, that a model of the robot is displayed in the real world environment and then the combined real world and computer generated information is used to program the robot.

Another overview over robot programming systems is given by Biggs in [4]. The survey divides robot programming into automatic programming, manual programming and software architectures. *Automatic programming* describes the methods where the programmer has no direct control on the generated robot program, like programming by demonstration with a teach pendant, by directly guiding the robot or by teaching the robot with gestures, voice or vision. However, *manual programming* includes all methods where the programmer has to enter the behavior of the robot using e.g. textual programming or graphical programming. The last part, software architecture, is part of all programming systems to form the underlying structure of the system.

The playback programming method itself is described by Lozano-Perez [5] . He uses the terms “teaching by showing” and “guiding”, which can be used synonymously to playback programming in our case. Other synonyms for playback programming are “lead-through programming” or “walk-through programming”. The paper [5] describes the simplest form of playback programming, which is guiding the robot and recording the joint angles, so that the movement can be reproduced afterwards. It also mentions extensions to the playback programming method, called “extended guiding”. For this, the playback programming is combined with the possibility to use different reference coordinate systems. The movement is recorded relatively to one coordinate system and it could be played back in a different coordinate system afterwards. This can be used to process identical work pieces which do not necessarily need to have the same spatial orientation and position for every iteration of the task.

An example for playback programming can be found in the EU project SMErobot [6], which shows how to intuitively program a robot by using the playback programming method. The video shows the ease of programming a welding robot to fulfill a certain task. A worker without experience in robotics can program the welding system, because he just has to move the robot to the correct positions without having to write control code.

Ang et al. present a walk-through programmed robot for welding shipyards in [7]. This approach works without the help of a teaching pendant by just moving the robot in the desired direction. The procedure is specialized in welding in shipyards, it does not allow the post processing of the recorded trajectory and it is also designed for only one robot.

A system for welding and adhering is described by Meyer et al. in [8]. The term "programming by demonstration"

in this paper is equivalent to our term "playback programming". The procedure uses the guiding method to teach the desired trajectory and afterwards post processes the trajectory, so that the resulting trajectory is adjusted to the work piece. The post processing part is split up into two parts, an automatic part and a manual part. The automatic part compresses the trajectory representation by generating splines. This results in a smoothing of the trajectory where the user is able to set the parameters for the splines. In contrast, the manual part allows the user to directly manipulate the smoothed trajectory by moving the positions of the points within the simulation window using drag-and-drop functionality. It also uses a voice command system that allows the user to control the robot system with short voice commands and a personal digital assistant to visualize the environment and the movement of the robot and also to simplify the input of parameters for the programming method. Drawbacks are, that the system is limited to one robot manipulator and it lacks the visualization of the trajectory in a timeline representation. The trajectory is just shown in a simulation window. Furthermore, this approach does not support multi-arm robot systems and therefore does not have any synchronization mechanism implemented.

Park et al. describe in [9] the development of a dual arm robot manipulator and a teaching system. The teaching system allows the user to directly teach the arms of the robot by moving them, which is similar to our playback programming. It is possible to teach both at the same time, or to teach them separately, that leads to the ability to teach coordinated and uncoordinated tasks with the help of this teaching system. Drawbacks of this approach are, that the user is not able to edit the generated trajectories and that there is no special synchronization function implemented. The approach just plays back the recorded trajectories in exactly the same way as they were taught.

After dealing with robot programming in general and playback programming in particular, we now review previous work concerning programming of multi-arm robots and their synchronization.

Connoly describes in [10] the Motoman dual-arm robots. Besides talking about the advantages of two-arm robots in an industrial environment, the article focuses on how to program them. There is either the possibility to use textual programming with the help of a simulation application to verify the created program, or the possibility to use graphical programming. The graphical programming approach in this work uses predefined operations that the user can arrange in a sequence of operations to create the robot program. Pick-up or set-down positions are specified by manually moving the robot to the designated positions. The paper also talks about the possibility for real-time synchronization, but this remains unspecified.

Gan et al. present in [11] an offline programming systems for multi-robot cooperation systems. Additionally, current offline programming technologies are reviewed and the status of cooperative functions and motions used by the robot manufacturers are introduced. After the presentation of principles of robot path planning for cooperative motions, the developed offline programming system is shown. To achieve a synchronization between the robots, the paper classifies different cooperative motions, namely concurrent cooperation, coupled synchronous cooperation and combined synchronous cooperation. *Concurrent cooperation* is a motion where all robots have the same start time, but where no position or orientation constraints exist between the robots. In addition to the same start time, all robots execute identical motions in a *coupled synchronous cooperation*. Hence there is no relative motion between the different end-effector frames. Finally, *combined synchronous cooperation* is a motion, where all robots have the same start time and a slave robot executes an arbitrary motion relative to the end-effector frame of the master robot while it follows the movement of the master robot. Gan's approach uses a temporal synchronization at the beginning of each motion and afterwards a local synchronization between the end-effector frames of the robot manipulators.

Makris et al. describes an intuitive way to program dual-arm robots with the help of the human language and gestures in [12]. This technique decomposes complex operations into a sequence of simple operations such as grasp, approach, etc. The parameters needed to fulfill those simpler operations are entered via a user interface, voice commands or gestures. The movement between the two arms is synchronized by adding dual-arm operations to the simple operations like bi-grasp or bi-approach. In addition to the human language to program the robot, the operator can also use gestures to initiate the execution of approach or rotate tasks by stretching his arm into predefined directions. The predefined dual-arm operations in this approach lead to a limited form of local synchronization, because the motions of the robots are only synchronized when using these special tasks.

The next aspect of our paper deals with the visualization of the recorded robot trajectories.

The video clip at [13] shows the use of bar charts to visualize a movement of robots with the help of SolidWorks, a 3D computer-aided design (CAD) and computer-aided engineering (CAE) software developed by Dassault Systèmes [14]. It uses one bar per element that is involved in the movement. That way, it displays the movement of each joint and of each conveyor separately. One global timeline is used, so that the user can spot at what time which components are moving within the simulation. The chart shows a bar if the related element is moving and it displays no bar in case it stands still. The amount of bar charts gets bigger with every moving element, which leads to a unclearer visualization of the motion, because there are too many bars one has to follow.

To conclude the first part, one can say, that there is several work about playback programming and multi-robot programming, but hardly anything that combines playback programming with an editing mechanism or multi-robot programming with playback programming. And there is no work about mutli-robot playback programming in combination with an editing and synchronizing mechanism to

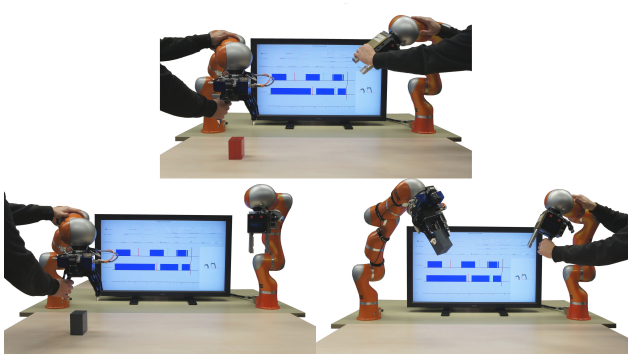

Figure 2 The three possibilities to program a multi-robot system by means of the playback programming paradigm. Top: parallel programming. Bottom left: sequential programming without playing back remaining robots. Bottom right: sequential programming while playing back already recorded trajectories.

post process the generated trajectories. That is, what we want to introduce in this paper and what is done throughout the next sections.

3 Programming Paradigm

The first part of this section explains the extension of the playback programming technique to multiple robot manipulators, afterwards, the representation of the trajectories, how they may be edited and how the synchronization mechanism works is shown.

3.1 Multi-robot playback programming

Playback programming for one robot manipulator is easy to handle and nearly self-explaining. The difficulty is to extend this playback programming technique to more than one robot manipulator. When extending the playback programming method to multiple robots, several problems emerge. The major problem is, how to coordinate the movement of the robot manipulators and especially how to allow the user some kind of rudimentary synchronization of the movements during the programming. The most important thing when it comes to synchronizing the robots is, that the user knows the relative position of each robot at all times. To achieve this, there is either the possibility to guide them concurrently through the desired trajectory (all at the same time), or to guide them sequential (one roboter after another). The latter option can be split up into two parts, one where all robots, that are currently not programmed, stand still, and another one, where all already programmed robots play back their trajectories simultaneous to the guiding. All in all, there are three options to program multi-robot systems, which are shown in Figure 2.

In the following, we take a look at the the effort needed to program a system with n robot manipulators: The first option is to program all robots at the same time, but then there is the problem, that it needs one operator per robot manipulator. So for n robot manipulators to program, one needs n persons to program the system. Furthermore, all involved operators need to know exactly what the final robot program should look like, none of the operators should make a mistake. Besides that, a positive aspect of the parallel programming approach is, that the users know the relative position of the robots to each other at all times.

The second option to program a multi-robot system by means of playback programming is, to program the robots sequentially, but in contrast to the last programming method, to not play back the already recorded trajectories when recording a new trajectory. This can be used, when the different robots have to fulfill independent tasks and therefore do not have to cooperate. This approach just needs one person to program the whole system. The problem with this approach is, that one rarely knows how the robots are located relatively to each other and therefore one cannot securely avoid collisions or teach collaborative tasks.

The last option is to program one robot manipulator after one another and play back already programmed trajectories while programming a new robot. This option combines the positive aspects of both previous options. With this option, the programmer knows at every time how the robot manipulators are located relatively to each other and therefore also have the advantage of programming all robot manipulators at the same time. Furthermore, the amount of persons to program the system is one, so it is held very low compared to programming all robot manipulators at once. A disadvantage of this approach is, that other robot manipulators are moving while the operator programs one manipulator. Therefore, other moving robots can be missed by the operator.

3.2 Internal and visual representation of trajectories

To achieve an easy manipulability, the internal representation of the trajectories is important. Internally, we use a layered structure to store different representations of the trajectories (Layer V, Layer I and Layer R). These representations can losslessly be transformed into each other. Therefore, all are equivalent. Layer V is used for the graphical representation in our programming system, layer I for the synchronization and editing tasks and layer R contains a representation that can be interpreted and executed by the robot controller. If one layer is manipulated, the other layers are automatically updated, so that all three layers contain the same information at any time.

Furthermore, the visual representation is an important aspect to allow the user an easy and intuitive manipulation and to provide him a quick overview on the trajectories. In the new programming paradigm, we chose to use bar charts similar to Gantt diagrams to represent the recorded trajectories (similar to video editing software). In a common time scale, there is one timeline per robot manipulator

which contains the representation of the trajectory. To give the user additional information, we designed the chart in a way that one can see, whether the robot manipulator moves at a certain point in time or whether it stands still. This is achieved by showing no bar when the manipulator stands still and by showing a bar when it is moving. Besides that, the bars can have different colors, depending on how the trajectory that belongs to the bar, was created.

An example for the visual representation used in our programming system is shown in Figure 3. We used the following colors for the bar: It is colored in blue if it was recorded regularly. In case it was calculated as a transition movement, the bar is colored in black. It is colored in red, if it is not possible to play back the underlying trajectory with the constraints set in the settings. In addition to that, the gripper commands like open the gripper or close the gripper are depicted in these bar charts as well. A green up-arrow stands for opening the gripper, a red down-arrow for closing it. Furthermore, the synchronization intervals (see Section 3.4) are also visually represented within our bar charts. A part of the trajectory that is executed synchronously is displayed with a grey background.

With all these graphical illustrations of the trajectory, it is easier for the user to get a quick overview on the movement of the robot manipulators without actually following the physical movement of the manipulators. Additionally, the visual layer may be supplemented by a graphical simulation of the robot movement.

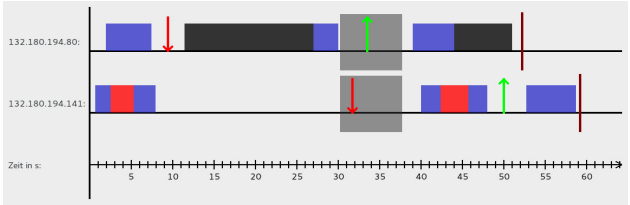

Figure 3 Example visual representation of the trajectory. Blue: recorded. Black: automatically generated transition movement. Red: not able to play back with current parameters. Green up arrow: open gripper. Red down arrow: close gripper. Grey rectangle behind timeline: synchronization interval. No visible block: robot manipulator stands still.

3.3 Editing of trajectories

Playback programming solely is not capable of creating complex robot programs, because the whole program needs to be programmed from scratch if the programmer makes one single mistake. Therefore, we developed an editing mechanism to edit already recorded trajectories in our programming system by copy, paste, cut and delete. With the help of this editing mechanism, small mistakes can easily be removed, and a single action can arbitrarily be repeated, as often as it is needed for the final program. Additionally, the multiple movements may be rearranged in a new order. The user selects a part of the visualized trajectory per drag-and-drop and afterwards chooses, what should be done with the selected part. One can choose between copy, cut and delete for a selected part of the trajectory. The three options do exactly what is expected from them. Copy copies the selected part to the clipboard, delete deletes the selected part from the Gantt chart, and cut first copies it to the clipboard and afterwards deletes it. Parts that were copied to the clipboard can be inserted at an arbitrary point in the trajectory of the same robot by selecting one point in the timeline and using the paste function to insert the copied part at this position.

One problem that emerges when deleting parts of the trajectory or inserting a new part into the trajectory is, that the robot manipulator needs to perform a transition movement at the start and end (bar edges) of the removed or inserted subtrajectory, because usually a discontinuity accrues at the edges of the insertion or deletion. There are different options to automatically create such a movement. One possibility is, to calculate a direct movement by using the constraints specified in the programming system, like maximum velocity and maximum acceleration. Another option is, to create a collision-free movement by using path planning and an environment model. It is also possible to let the user record a transition movement by means of playback programming with the help of the compliant robot by adjusting its impedance, so that the robot can easier be moved in the direction of the target point than in the opposite direction. There are certainly further ways to create such a transition movement, but in the end, it is important, that the resulting trajectory is valid and executable.

In our prototypical implementation, we chose to calculate the transition movement by creating a direct movement between both positions by using the maximum velocity and maximum acceleration specified in the settings of our programming system. In that case, we do not perform any collision test, but the generated transition movement ensures, that the whole trajectory remains valid.

3.4 Synchronizing of trajectories

Finally, our programming paradigm contains a synchronizing mechanism that allows the user to establish a temporal synchronization between the multiple robot components, so that certain movements are executed synchronously. This feature is important if tasks have to be coordinated, for example one robot manipulator holds a work piece and has to pass it to another robot manipulator, then the first one is not allowed to open its gripper before the second one closed its gripper. Another example is, that two robot manipulators have to grasp the same work piece and have to place it somewhere else, then the movement also needs to be executed synchronously, so that the two robot manipulators work as a parallel kinematics for a certain period of time. Both can be achieved with the help of the synchronization mechanism introduced in the following.

In the approach taken here, a *synchronization interval* is defined by its start and endpoint on the timeline. These interval limits concurrently refer to more than one robot

component. To visualize the intervals for the user, squared brackets or a grey background rectangle may be displayed within the Gantt diagram. Inside the intervals, the recorded trajectories of the involved robot components are played back exactly synchronous. The edges of a synchronization interval can coincide at one point in time to form a *synchronization point*. At a synchronization point, the involved robot components have to stop executing the trajectories until all synchronized robot components have reached that point. Afterwards, the robot components continue executing the trajectories independently from each other. This behavior is similar to the concept of a barrier in parallel computing.

With the above definition of synchronization intervals and synchronization points, it is reasonable to restrict the edges of intervals and points to a position in the timeline, where the associated robot components stand still. This constraint is needed to protect the robot components from damage, because if one component has to wait at an edge of a synchronization point/interval, for another robot component and it would be allowed to insert synchronization points/intervals within the trajectory, where the associated component is moving, then the robot component has to reduce its velocity to zero in no time. This may lead to a large force, which can cause damage to the robot component. Usually, this constraint still covers all needed synchronization intervals, because typically every robot manipulator stands still before and after opening or closing the gripper and at the beginning and the end of the movement.

4 Experimental Results

This section gives an overview on how to use our prototypical implementation of the introduced programming paradigm. It also shows which scenarios are realizable and how to program an example task. At the end of the section, the limits of the current implementation are shown.

4.1 Robot setup

Figure 4 shows the setup that we use to test the implementation. For our testing environment, we use two Kuka Lightweight Robot 4 [15] with seven degrees of freedom and a payload of 7 kg each. Both robot manipulators use the Kuka KRC2 controller and are connected to a computer that is used to execute the robot programs. Our programming system is able to run on any computer or laptop that is connected to the same network as the two computers that control the robot manipulators. In our case, we simply use one of the computers that are connected to the robot manipulators to run the programming environment. Each of the robots is equipped with a gripper that allows us to perform pick-and-place operations. One uses the 3-Finger Adaptive Robot Gripper from Robotiq [16], the other one uses the PG 70 2-Finger Parallel Gripper from Schunk [17]. The whole setup consists of four robot components, namely the two robot manipulators and the two grippers. Therefore the visual representation consists of four timelines, one for each robot component.

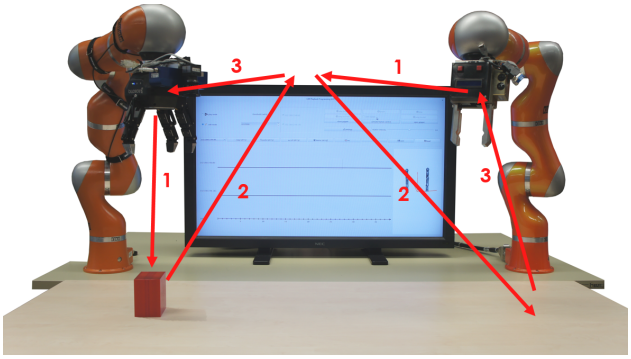

Figure 4 Robot setup with four robot components, namely two robot manipulators and two grippers in front of a big screen that is used to display the programming system. The arrows indicate the task that should be executed in section 4.2

4.2 Example application

This section shows, how to use the previously described programming system with our setup. It also describes example use cases for the system.

After starting all components of the setup, the robots are automatically registered at our programming system. A popup occurs and informs about the robot that is now available for programming purposes. In this popup, the user has to select the gripper connected to the newly registered robot manipulator and the position of the robot within the simulation widget by entering x, y and z coordinates. After completing this step, a new timeline is shown for the robot with an empty trajectory. This step has to be repeated for all robots that are part of the multi-robot system.

Once all robots are registered, the programming of the robots may begin. In this example we assume a left and a right robot. The task, that we want to show in detail is grasping a work piece with the left robot, then handing it over to the right one and afterwards placing this work piece at a desired position. This task is shown in detail in Figure 4. To show the difference between synchronization points and intervals, the passing of the work piece can be done in two different ways. The first option is, that the work piece is just handed over to the right robot manipulator and the left one moves back to its starting position after opening its gripper. The second option is, that after grasping the workpiece with the right robot, both robots perform a synchronous movement, similar to a parallel kinematic, while the work piece is grasped with both grippers. After finishing this motion, the left robot manipulator releases the work piece and moves back to its starting position, whereas the right robot manipulator finishes its trajectory by placing the work piece at the designated deposition spot. Here, one can see, that the first option needs a synchronization point that is located between closing the gripper of the right robot manipulator and opening the gripper of the left one. The sec-

ond option needs a synchronization interval, because the motion between closing the gripper of the right robot manipulator and opening the gripper of the left one needs to be executed synchronously, otherwise the work piece or the robot manipulators may be damaged.

To start programming the whole system, we first have to decide, which of the two options we want to implement. Depending on this decision, we choose the way to program the robots.

When choosing the first option, the appropriate way to program the whole system is, that we use the option to program the robots sequentially one after one another and play back the previously recorded sequences of the robots, that already have been recorded, as it is shown in the bottom right picture of Figure 2. First, we start recording the left robot manipulator and move it from its starting position to the position where it should grab the work piece. On the way to the work piece, the gripper needs to be opened. When the work piece is reached, we need to close the gripper and afterwards move the manipulator with the work piece held by the gripper to the position, where we want to pass the work piece to the second robot manipulator. After reaching this position, we open the gripper and move the robot manipulator back to its starting position and stop the recording of the first robot. After the recording is stopped, a visual representation of the recorded trajectory is shown in the timeline of the first robot manipulator. Now, we insert a synchronization point right before the gripper releases the work piece, so that, when we record the second robot manipulator, we have enough time to grab the work piece before the first one releases it. The next step is to record the second robot manipulator, which is in our case the right one. We choose the option to record the robot manipulator while the other robot manipulators play back their previously recorded trajectories. So, while we start moving the second robot manipulator from its starting position to the position where the passing of the work piece takes place, the first robot manipulator plays back its trajectory until it reaches the synchronization point. At this point, the first robot manipulator waits for the second one until it has grabbed the work piece in the passing area and until the robot programmer has confirmed the synchronization point. Then, the first one releases the work piece and moves back to its starting position, whereas the second one needs to be moved to the position where it should place the work piece. When reaching the deposition spot, the gripper of the second robot manipulator is opened and the work piece is placed at the desired position. Afterwards, the second robot manipulator has to be moved back to its starting position and then, the recording of the second robot manipulator can be terminated. Now, the timelines of the robot manipulators show the visual representation of the recorded trajectories, including the gripper movements and the synchronization point. These are all steps to program the multi-arm robot to hand over a work piece from one robot to the other one. Now, the whole program can be played back as often as it is desired and the robot manipulators will move the work piece from the starting position to the end position of the whole movement.

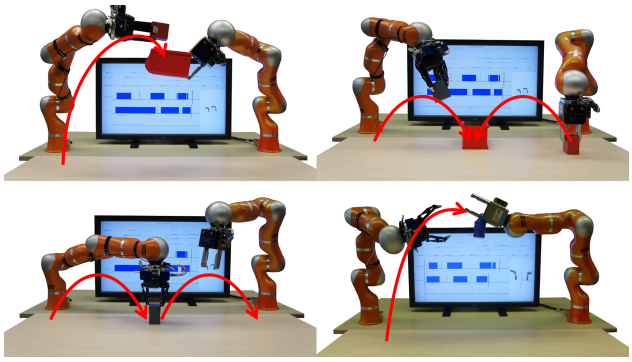

Figure 5 Further scenarios to use the programming paradigm in. Top left: Place a work piece with a robot manipulator in a box that is held by a second robot manipulator. Top right: Lay work pieces with two robot manipulators sequentially into a box in the middle. Bottom left: Place a work piece with the first robot manipulator in the middle, grab it with the second one and put it aside. Bottom right: Hang a work piece with one robot manipulator on a hook that is held with the second robot manipulator (In our case, the hook is the gripper of the second robot manipulator).

When choosing to implement the second option, which contains a synchronous movement with both robots grasping the work piece, the option to implement all robots at the same time, as it is shown in the top picture of Figure 2, is the best to choose. In our case, this option requires two operators to program the system, because both robot manipulators need to be programmed simultaneously and therefore one person is needed for each robot manipulator. The previously described movement is now programmed in a parallel way instead of the sequential programming in the paragraph before. After the recording is finished, the programming system displays a visual representation of the trajectories of both robot manipulators in each of their timelines. One of the programmers now just has to insert a synchronization interval that starts right after the second robot manipulator grabs the work piece and ends right before the first robot manipulator releases the work piece. By inserting this, it is ensured, that the synchronous motion always stays the same, even if parts of the remaining trajectories are edited with copy, paste or delete.

But there are also other possible scenarios where to use the new programming system in. Four of them are shown in Figure 5. The top left picture shows a motion, where the left robot manipulator grabs a work piece and places it inside a box that is held by the right one. In this example, a synchronization point has to be used to ensure that the left robot manipulator releases the work piece only when the second one is already at the correct position with the box. The top right picture displays a movement, where both robot manipulators place work pieces in a box that stands in between them. Here it is important, that a syn-

chronization point is used to ensure, that only one robot robot manipulator is at the box at any time, otherwise they will collide. A motion where the left robot manipulator places a work piece between the two manipulators and the right robot manipulator picks this work piece up and places it at a designated deposition spot is shown in the bottom left picture. Here, it is also important to use a synchronization point to assure, that not both robot manipulators are at the position between them at the same time, since they are going to collide otherwise. The last example given in Figure 5 is a motion, where one robot manipulator hangs up a work piece on a hook that is held by the other one. In this picture, we use one finger of the gripper as the hook to demonstrate it. These are just a few examples to show how flexible our newly developed programming system is.

4.3 Limits

The current implementation of the programming system has its limits when the tolerance of the task, that should be executed, is not big enough. This is due to the impedance mode that we use to play back the recorded trajectories. Therefore, tasks like putting a peg into a hole with small tolerance are not feasible with the current implementation, because it cannot be assured, that the played back movement is perfectly identical with the recorded one. The movements of two executions can slightly differ from each other, which would lead to failure if the tolerance of the task is smaller than the difference between the recorded and the played back movement.

5 Conclusion

To summarize, one can state, that we extended the multi-robot playback programming approach by an editing mechanism, that has a special visual representation similar to a Gantt diagram, a graphical editing function and a synchronization function. This is an important step towards bringing robot systems to small and medium sized enterprises, because no expert is needed anymore to program these systems. The whole paradigm is implemented in a prototypical way and was submitted as patent application at the German Patent and Trademark Office [18].

6 Literature

[1] D. Ardayfio. *Fundamentals of Robotics*. Mechanical Engineering. Marcel Dekker, Inc., 1987. 978-0824774400.

[2] Martin Hägele, Klas Nilsson, and J. Norberto Pires. Industrial Robots. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics*. Springer, 2008. 978-3-540-23957-4.

[3] Zengxi Pan et al. Recent progress on programming methods for industrial robots. *Robotics and Computer Integrated Manufacturing*, 28(2):87–94, 2012.

[4] Geoffrey Biggs and Bruce Macdonald. A survey of robot programming systems. In *Proceedings of the Australasian Conference on Robotics and Automation, CSIRO*, page 27, 2003.

[5] Thomas Lozano-Perez. Robot Programming. *Proceedings of the IEEE*, 71(7):821–841, 1983.

[6] SMErobot. Intuitive Programming by Demonstration: Welding. 2009. `http://www.smerobot.org/15\_final\_workshop/download/full%20resolution/D3_Programming_by_demonstration_1024x576_2000kBit.mov`, last visit: 16.03.2016.

[7] Marcelo H. Ang Jr., Wei Lin, Ser-Yong Lim. A walk-through programmed robot for welding in shipyards. *Industrial Robot: An International Journal*, 26(5):377–388, 1999.

[8] Christian Meyer, Rebecca Hollmann, Christopher Parlitz, and Martin Hägele. Programmieren durch Vormachen für Assistenzsysteme - Schweiß- und Klebearbeiten intuitiv programmieren. *it – Information Technology*, 49(4):238–246, 2007.

[9] ChanHun Park, KyoungTaik Park, Dong Il Park, and Jin-Ho Kyung. Dual arm robot manipulator and its easy teaching system. In *Assembly and Manufacturing, 2009. ISAM 2009. IEEE International Symposium on*, pages 242–247, 2009.

[10] Christine Connoly. Motoman markets co-operative and humanoid industrial robots. *Industrial Robot: An International Journal*, 36(5):417–420, 2009.

[11] Y. Gan, X. Dai and D. Li. Off-Line Programming Techniques for Multirobot Cooperation Systems. *International Journal of Advanced Robotic Systems*, 10, 2013.

[12] Sotiris Makris, Panagiota Tsarouchi, Dragoljub Surdilovic, and Jörg Krüger. Intuitive dual arm robot programming for assembly operations. *CIRP Annals - Manufacturing Technology*, 63(1):13 – 16, 2014.

[13] Javelin Technologies Inc. SolidWorks Tutorial: How to Animate a 6 DOF (degrees of freedom) Robot. `https://www.youtube.com/watch?v=QY4T51KkzJI`, last visit: 16.03.2016.

[14] Dassault Systèmes. SolidWorks. `http://www.solidworks.com/`, last visit: 16.03.2016.

[15] KUKA AG. KUKA LWR 4 data sheet. `http://www.kukaconnect.com/wp-content/uploads/2012/07/KUKA_LBR4plus_ENLISCH.pdf`, last visit: 16.03.2016.

[16] Robotiq Inc. Robotiq 3-Finger Adaptive Gripper data sheet. `http://robotiq.com/wp-content/uploads/2014/08/Robotiq-3-Finger-Adaptive-Gripper-Specifications-ES.pdf`, last visit: 16.03.2016.

[17] Schunk GmbH & Co. KG. Schunk 2-Finger Parallel Gripper data sheet. `http://www.schunk.com/schunk_files/attachments/PG_70_EN.pdf`, last visit: 16.03.2016.

[18] Universität Bayreuth. Robotersteuerung, September 2015. Deutsche Patentanmeldung Nr. DE 10 2015 116 086.2.