

Towards Skills-based Easy Programming of Dual-arm Robot Applications

Fan Dai

ABB Corporate Research, Ladenburg, Germany

Keywords: Dual-arm Manipulation, Robot Programming, Robot Skills.

Abstract: Programming dual-arm robotic applications requires good understanding of the tasks and the coordination between both arms must be well specified. This article analyses the synchronization modes required in dual-arm robot applications and describes a mechanism of programming these applications based on synchronizing the execution phases of robot skill functions for the two arms. Combined with a graphical user interface, it contributes to the ease of use of dual-arm robot systems.

1 INTRODUCTION

Dual-arm, or two-handed manipulation has been an interesting topic in robotics since the very beginning of robotics research related to tele-operation, and it is again of high interest with new developments in service and industrial robotics (Smith et al., 2012). Many efforts and advances are known in control of coordinated motion, but the programming of dual-arm manipulation is still one of the main bottlenecks of application development, because programming dual-arm robot applications requires very good understanding and specification of the coordination between the two arms.

Zöllner et al., (2004) worked on programming by demonstration for dual-arm manipulation tasks, where task executions are modelled with Petri nets. Having the arm states active or ready as conditions, dual-arm task planning and execution can be learned by mapping observed bimanual human demonstrations. While this method is theoretically quite promising, many problems still must be solved, especially the interpretation of human intentions in complex situations.

The concept of robot skills (e.g. Kröger et al., 2010, Thomas et al., 2013, Dai et al., 2016) has been introduced to allow task-level programming, which is more intuitive for application engineers. Theoretical concepts cover the coordination of parallel tasks as well, but most approaches are still focused on single arm application tasks, or tasks where two robot manipulators must avoid collisions with each other.

An appropriate mechanism for two-arm robot application tasks is needed.

Szynkiewicz (2012) worked on skill-based bimanual manipulation planning, uses Rubik's Cube as an example to implement a two-handed manipulation skill involving vision and force control. However, the mechanism of specification and realization of coordinated dual-arm tasks is not clearly described.

Stenmark et al., (2017) worked on improving intuitive dual-arm programming of collaborative industrial robots, utilizing the concept of re-usable robot skill functions, where the focus was on synchronizing primitive dual-arm motion constructs. They also combined it with an iconic graphical user interface, allowing adding and modifying synchronizations, but limited to primitive motion synchronization.

Our approach goes further to a mechanism of creating dual-arm robot skills by analysing and specifying the synchronization of single arm robot skills. It allows an easier way of programming bimanual application tasks.

In this paper, we analyse two-handed manipulation taking assembly applications as an example (section 2). Based on this, we derive in section 3 our concept of synchronizing two-arm tasks by introducing phase-based synchronization. We propose to implement robot skill functions with clearly defined phases and synchronization points, which can be parameterized, optionally via a graphical user interface.

2 TWO-HANDED ASSEMBLY TASKS

In this section, we take an application-oriented view on dual-arm manipulation, focusing on assembly tasks. Applications in which two-handed operations come into play can be grouped as follows:

- Coordinated parallel tasks
- In-hand manipulation using one hand as part holder (fixture) for the other one
- In-hand manipulation with simultaneous actions of both hands

In the following, we analyse these three groups of applications with some examples.

2.1 Coordinated Parallel Tasks

These types of applications may be further split into two sub-groups. For most cases in the first sub-group the overall task could be done by a single hand. Doing it with two hands simply increases the efficiency or shortens the cycle time.

For example, when piling up objects onto each other, a single robot arm could take the parts one by one and accomplish the task alone. But doing it with two robot arms is much quicker. The same holds when multiple different parts are to be mounted. Similarly, in many applications, one arm is used to take the raw part and place it at the location for processing. The other arm then retrieves the finished part.

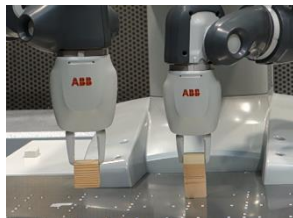


Figure 1: Piling-up cubes with coordinated parallel tasks.

The second sub-group includes applications in which two hands are needed to accomplish the task, but they do not come into direct mechanical or spatial interaction with each other, for example: One hand opens the cover; the other one puts the part into the container.

In these two-handed applications, compared to single hand operation, the actions of both hands must be synchronized with each other, but not the motion details. Important are: 1) Temporal dependencies of starting and ending the actions; 2) Spatial collision avoidance.

The goal is to achieve the shortest cycle time. If the constraints on cycle time allow, one can also try to optimize energy consumption, minimize mechanical stress etc.

2.2 One Hand as Part Holder for the Other One

At manual workplaces, we can very often observe two-handed operations in which one hand acts as a part holder for the other hand. Here we can have the following situations:

2.2.1 The One Hand Holds a Part, and the Other Hand Mounts a Second Part onto It

As shown in Figure 2, one of the robot hands holds the base part, while the other hand mounts the other part onto it. This is the most common case. In this case, the first hand must maintain a fixed position for the duration of the part assembly by the other hand.



Figure 2: Examples of two-handed assembly with one hand as part holder.

2.2.2 The One Hand Holds a Part, and the Other Hand Mounts This Part onto Other Part

The pictures below show a connector on a flexible flat cable. One hand must place the part on top of the socket, so that the other hand can then push on the connector to mount it firmly.



Figure 3: One hand supports the other to fix the socket plug.

In this kind of applications, the first hand must be compliant, following slightly the movement of the part when the part is moved by the other hand while mounting it.

2.2.3 The One Hand Hands over a Part to the Other Hand

When a part cannot be reached by the one hand that should operate with the part, or it can be reached, but cannot be picked at the right position, a commonly used option is to take the part by another hand first, and hand it over to the operating hand.



Figure 4: Handing over the cube.

In this case, there is a hand-over phase for a short while, requiring synchronization as the second hand takes the part.

2.2.4 Simultaneous Actions of Both Hands

The screw-mounting example can also be done by simultaneously turning the two parts counter-wise. This shortens the time for this task. In addition, if the rotation of the last robot joint is used to screw the part, like in this example, simultaneously turning both parts avoids re-grasping, which may become necessary due to joint limits in case of turning one part only.

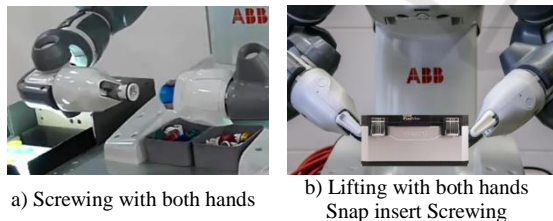


Figure 5: Examples of simultaneous actions.

Another example is using two hands to take a relatively large part and perform placing or mounting. Compared to a single-handed grasp, this can avoid undesirable torques on the gripper tool.

In such applications, the motions of both hands must be synchronized, or one of the hands must follow the other hand for defined phases of motion.

3 SYNCHRONIZATION OF BOTH HANDS



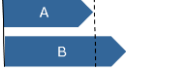


A two-handed robot task is composed of at least two sub-tasks, each performed by one of the robot hands. If the robot hands are programmed so that each of the hands executes their own skills, these skill functions must be coordinated with each other. The most important criteria are: 1) Ensure that the main phase of the desired task succeeds; 2) Achieve the shortest cycle time. Since the duration of individual tasks differ, it is usually not possible to schedule bimanual manipulation without waiting times. When the success of a two-handed manipulation task requires one hand to wait on the other, optimal scheduling can avoid excessive waiting times.

In the following, we consider the general modes of synchronization between parallel processes and the execution of skill functions as well as the corresponding (sub-) tasks of an application, and then discuss how the different types of two-handed operations can be synchronized.

3.1 Synchronization Modes

Theoretically, five modes of task synchronization can apply (Table 1). However, not all of them are essential for two-handed assembly. The most common mode is of type a) End-start synchronization. Special applications require synchronous motion (type e). Other modes can be beneficial in some cases but are not necessarily required.

Table 1: Task synchronization modes.

a) End-start: Start of task A (e.g. left arm) waits until end of task B (e.g. right arm)	
b) Start-start & end-end: Starting and ending at the same time.	
c) Start-start: synchronous start, free ends	
d) End-end: free starts, synchronous ending	
e) Synchronous motion: The complete motions of both hands are synchronized	

To know which mode is required in which situation, we analyse in more detail below the phases of assembly skills as they are executed by one hand, and how they can be used in the two-handed

application examples.

3.2 Phases, States, and Events

If we look at manipulation tasks with one robot arm, typically, they consist of the following major execution phases:

- **Approach** – go to the starting position for the main phase
- **Main phase** – execute the intended action, such as grasping, pushing, insertion, snapping, screwing, etc.
- (Optional) **Release** – release the work piece e.g. by opening the gripper, if it holds the part
- **Depart** – move away from the current work area

The details of the main phase depend on the type of actions. Taking snap insertion as example, we can have the following sub-phases:

- **Push** towards the goal position with compliance until measures of completion are met
- **Check** for success, if necessary, by trying to pull the part back, or move around, or by any other means of sensing
- **Error handling**, e.g. retry, dispatch, or report error.

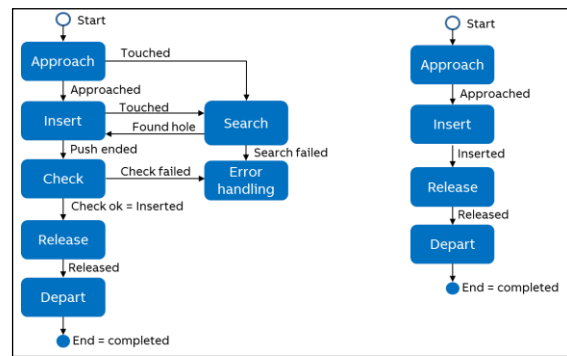
While moving towards the goal position (approach or push), it can happen that the part misses the correct starting position for insertion and touches the base object. In this case, an additional phase is necessary:

- **Search** for the correct starting position.

The above described phases and their conclusions are actually the significant states of the snap insert execution. Each phase change (state transition) can be exposed to another task for synchronization. While the phases are natural states, the phase end states would automatically be the start states for the next phase, if no waiting for external events is introduced.

Figure 6 shows a flow chart with these phases and phase changes. The Search phase can be entered from the Approach phase, when the part contacts the base object while moving towards an incorrectly defined start position, or from the Insert phase, when the Insert phase starts from an incorrect position. In both cases, the part would touch the surface of the base object in an unintended way.

From a task-oriented point of view, Search, Check and Error-handling are internal phases that do not reflect the intended states of task accomplishment. A synchronization of such phases with external actions is not required in general, except for certain error states, if strategies for error handling involving both hands are introduced. But error handling is a very



a) State model with internal phases b) Simplified model

Figure 6: Phase of snap insertion.

complex and special topic, which we do not further discuss in this article.

For simplicity, we use in the following the simplified state model shown in Figure 6 b), in which insertion can be replaced by other skills such as screw-driving, etc.

3.3 Two-handed Operations with Synchronization

When looking into the application examples, we can group them into: 1) Sharing work space; 2) One hand as part holder; 3) Hand-over; 4) Two-handed symmetric assembly; 5) Two-handed synchronous motion. In the following, we discuss how these groups of applications can be implemented with corresponding phase synchronizations.

3.3.1 Sharing Work Space

In case both hands must accomplish actions within the same work space, they cannot execute these simultaneously. Sometimes, the order of the actions is defined by the application itself; sometimes serialization is needed for collision avoidance.

For example, the piling-up example could be done by two-hands with one waiting for the other to finish its placing action (Figure 7).

But this may cause unnecessary waiting time. In this example, the most feasible way is in fact, that one hand waits at an intermediate approaching position until the other hand has left the departing position. Thus, introducing intermediate synchronization points can reduce the overall cycle time. Here, depart (of one hand) and approach phase (of the other hand) each can be split into two sub phases for better synchronization (Figure 8), but the synchronization mode is still “end-start” sequencing of these two phases.

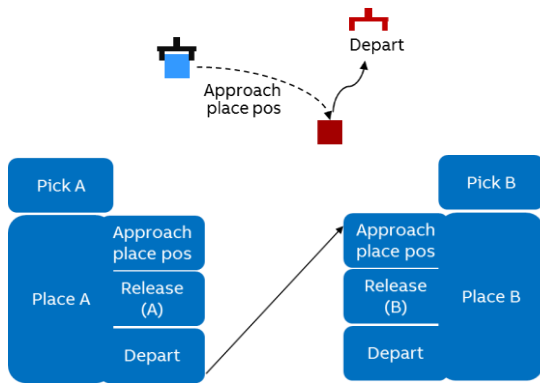


Figure 7: Piling up with two hands.

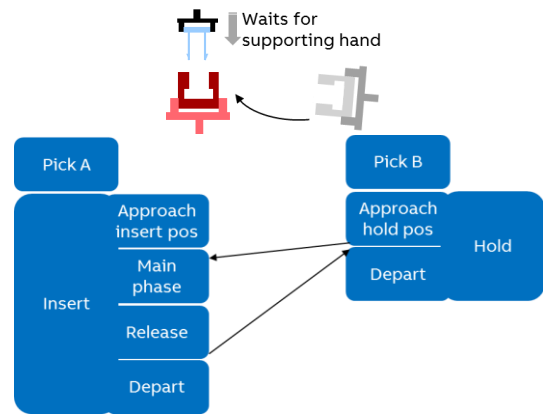


Figure 9: Using “end-start” for synchronization of two-handed insertion.

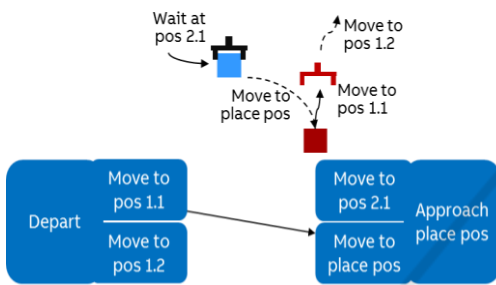


Figure 8: Use sub phases to reduce cycle time.

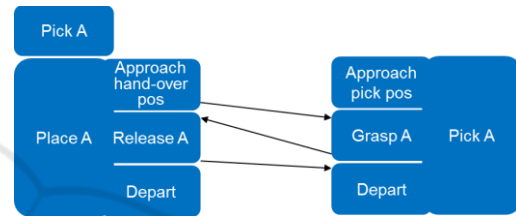


Figure 10: Synchronizations for hand-over.

This intermediate approach position is normally different from the approach position of single hand operations, because it must ensure that no collision with the other hand can occur.

3.3.2 One Hand as Part Holder

When one hand acts as part holder like in the two-handed snap-insertion example, it requires that the supporting hand is in position before the inserting hand starts the main phase of insertion, but the latter can already move to the approaching position. This can be simply implemented using “end-start” synchronization as depicted in Figure 9.

Certainly, it could also run with synchronized ends of the approaching phases. However, this does not lead to any benefits.

3.3.3 Hand-over

For hand-over operations, the one hand releases the part after the other hand has picked it. It is again a simple “end-start” relationship between the corresponding phases as depicted in Figure 10.

3.3.4 Two-handed Symmetric Assembly

As shown in Figure 11, two-handed symmetric assembly (taking the example of screwing) always can be implemented using “end-start” synchronization as well.

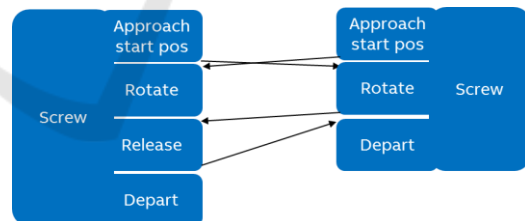


Figure 11: Using “end-start” synchronization in two-handed symmetric screwing.

This is equivalent to “start-start” synchronization of the rotate phases, if no additional waiting time before the starts of these phases is introduced by other events that are in principle possible, but not relevant here. Of course, symmetric assembly could also be implemented with synchronized start and end (“start-start & end-end”) of corresponding phases of both hands (e.g. Figure 12.), though this is not really required. In the practice, “end-start” synchronization is easier to implement.

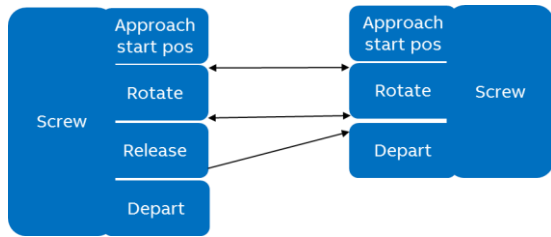


Figure 12: Using “start-start & end-end” for two-handed symmetric screwing.

3.3.5 Two-handed Moving

Different from the above use cases, synchronous motion is required in such special cases as lifting or moving a large or heavy part with two hands.

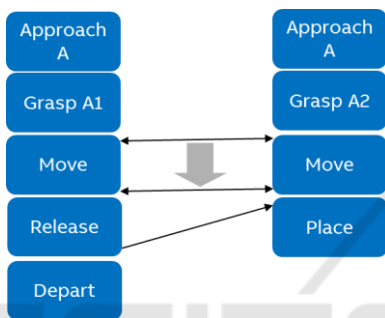


Figure 13: Synchronous motion (may also require simultaneous release).

3.4 Usage Summary of Synchronization Modes

Based on the above observations, we can conclude that “end-start” is the most commonly used mode, where “synchronous motion” is required for special cases. Other modes are not necessary for implementing two-handed assembly applications.

Table 2: Usage of synchronization modes.

Synchronization mode	Usage
End-Start	Most applications can be implemented this way
Start-start	Can be achieved with “End-start” mode (see e.g. Figure 12)
Start-start & end-end	Not required
End-end	Not required
Synchronous motion:	Required in special cases e.g. two-handed lifting

4 ROBOT SKILLS AND VISUAL PROGRAMMING

From a programming point of view, robot skill functions are nothing else than higher-level parameterizable functions. These can be used by application programmers in any programming environment with any programming tools including simple text editors. We also proposed to use visual programming for guiding and assisting the application programmer to use skill functions in an intuitive way (Dai et al., 2016).

In a concept demo implementation, the skill description is stored in XML format including: function name, parameters, locations of other related data, e.g. source code of the skill function, support functions used by the skill function or for applying it, UI elements that can be optionally used, description and help text, including multimedia presentations. For the UI and multimedia presentations, XAML was chosen, which can be dynamically loaded to the App to provide individually customized user interface pages.

According to our phase-based synchronization concept, parameters of the skill functions also include skill phases and the skill execution state.

Taking a snap insertion skill programmed in the robot programming language RAPID as an example, the robot skill functions have the following constructs allowing the synchronization concept described in the sections above:

```

PROC SnapIn(pose StartPose,
  \pers state myState,
  \pers state waitToStart,
  \state waitToStartValue,
  \pers state waitToInsert,
  \state waitToInsertValue,
  \pers state waitToRelease,
  \state waitToReleaseValue,
  \pers state waitToDepart,
  \state waitToDepartValue)

myState := stateWaiting;
IF (Present(waitToStart) AND
  Present(waitToStartValue))
THEN
  WaitUntil
    (waitToStart>=waitToStartValue)
    \pollrate:=0.004;
ENDIF
myState := stateApproaching;
!start approaching
...
!end approaching
    
```

```

myState := stateApproached;
IF (Present(waitToInsert) AND
    Present(waitToInsertValue))
THEN
    WaitUntil

        (waitToInsert>=waitToInsertValue)
        \pollrate:=0.004;
ENDIF
myState := stateInserting;
! start inserting
...
!do something
myState := stateEnd;
ENDPROC
    
```

The user can compose the action sequences for each robot hand (i.e. the corresponding robot control task) by selecting and concatenating these skill functions and program elements.

To compose two-handed applications, one usually also must specify the synchronization points for the two hands. This can be done based on the concept as discussed in the previous chapter. The UI page is shown below in Figure 14.

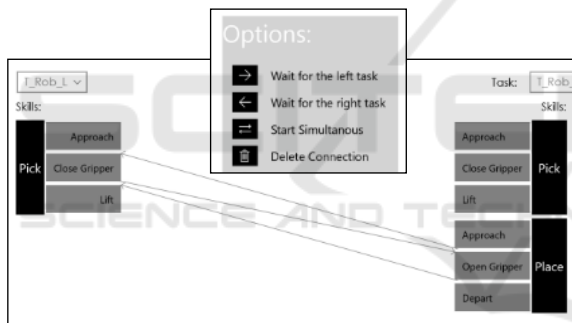


Figure 14: Skill phase synchronization page (arrow indicates the order).

For a pre-defined two-handed skill, there is no need for manual editing of the synchronization points. Therefore, we have chosen a different type of visualization (two lines without arrows) as shown below in Figure 15. It indicates that both skills are considered as a unit, so that user doesn't need to see the internal synchronization.

Typical for two-handed skills, some motion positions can be taught for both hands together, e.g. via lead-through as shown in Figure 16.

Finally, the app will create the main modules for the robot control tasks, which call the corresponding skill functions with these parameters (including synchronization points). The finished application is then ready to be started.

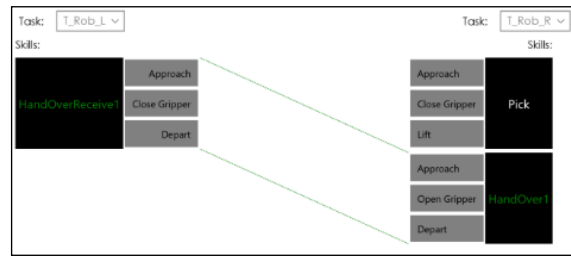


Figure 15: Hand-over as two-handed skill function.

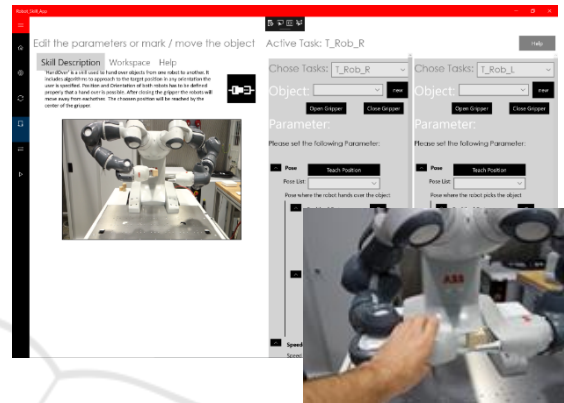


Figure 16: Teaching motion positions for both hands.

5 CONCLUSIONS AND FUTURE WORK

The robot skills concept can be applied to support more intuitive and easy programming of two-handed robot applications by synchronization of skill execution phases. Our analysis has shown that the two basic synchronization modes: end-start synchronization and synchronous motion are sufficient for all these applications. With pre-implemented synchronization points and optional parameters, the skill functions can be flexibly used by application programmers for different situations.

Visual programming can also contribute to ease of use. The skill execution phases can be shown beside the skill function. Known graphical interaction methods can be applied to define the synchronization between skill functions of the two robot hands, e.g. by connecting anchor points of the graphical elements.

Still, programming two-handed applications requires good knowledge of the skill functions, the synchronization modes, and logical reasoning. To further increase ease of use of robot systems like YuMi, two-handed skills or skill templates may be provided. Such skills may consist of pairs of skill functions for both hands with well-defined

synchronizations, considering the following two-handed operations:

- Sharing work space
- One hand as part holder
- Hand-over
- Two-handed symmetric assembly
- Two-handed moving of parts

In this context, we did experimental implementations of “hand-over” and “sync move” (two-handed moving of parts) with corresponding user interface elements.

Certainly, the mechanism of synchronization enables two-handed applications, but the efficiency of such applications depends also on other factors that influence the behaviour of the robot hands, e.g. the “intelligence” of the underlying skill functions, the execution parameters of the primary functions used by the skills. These are also coupled with the skill phase design and synchronization type. Therefore, we will further work on studying the correlations between both, and how machine learning methods can help to increase efficiency of two-handed skills and two-handed applications.

ACKNOWLEDGEMENTS

Research partially supported by European Union as part of the Productive 4.0 project (<https://productive40.eu>).

REFERENCES

- Smith, Ch., Karayiannidis, Y., Nalpantidis, L., et al. (2012) “Dual arm manipulation—A survey”. *Robotics and Autonomous Systems* 60(10), October 2012. pp. 1340–1353.
- Zöllner, R., Asfour, T., Dillmann, R. (2004) “Programming by demonstration: dual-arm manipulation tasks for humanoid robots”. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept. 2004.
- Kröger, T., Finkemeyer, B., and Wahl, F. M. (2010) “Manipulation primitives—A universal interface between sensor-based motion control and robot programming,” in *Robot Systems for Handling and Assembly*, 1st ed., ser. Springer Tracts in Advanced Robotics, D. Schütz and F. M. Wahl, Eds. Berlin, Heidelberg, Germany: Springer, 2010, vol. 67.
- Thomas, U., Hirzinger, G., Rumpel, B., Schulze, Ch. and Wortmann, A. (2013) A New Skill Based Robot Programming Language Using UML/P Statecharts, 2013 IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, May 6-10, 2013

Dai, F., Wahrburg, A., Matthias, B., Ding, H. (2016) “Robot Assembly Skills Based on Compliant Motion”, 47th International Symposium on Robotics (ISR 2016), June 2016

Szynkiewicz, W. (2012) “Skill-Based Bimanual Manipulation Planning”, *Journal of Telecommunications and Information Technology*, 2012(4): December 2012, pp. 54-62

Stenmark, M., Topp, B.A., Haage, M., Malec, J. (2017) “Knowledge for Synchronized Dual-Arm Robot Programming”, *AAAI Fall Symposium Series* 2017.