Intuitive Robot Programming of Spatial Control Loops with Linear Movements

Katharina Soller and Dominik Henrich

Abstract. Programming by demonstration (PbD) is a promising approach to facilitate the use of robots. The bandwidth of PbD systems extends from simple play back to artificial intelligence methods. Our approach is based on the former. This paper aims to introduce and examine intuitive programming of spatial loops, i.e. program loops whose body changes according to a spatial variable. The loop is programmed by executing e.g. the first, the second, and the last run. The system then determines all components of the loop so they can be used to generate a program. The focus of this paper lies in automatic determination of the variables for the start, the step and the end of spatial loops. This method of robot programming has interesting applications in the field of industrial robotics, especially with respect to small- and medium-sized enterprises, as well as for service robotics.

1 Introduction

One of the largest tasks involved in expanding the application range of robots (e.g. for use in household applications or frequently changing applications in small enterprises) are the typically difficult and time-consuming methods of programming them. The usual way to program an industrial robot is to write a program (either offline or online) and teach the positions used in the program directly on the robot. One major drawback of this procedure is that the programmer must have some competence in the field of robot programming. In addition, this type of programming does not support the typical work flow used in small enterprises or households, because it is time-consuming and unwieldy.

For these reasons numerous efforts have been made to make robot programming more intuitive. One approach is called “Programming by Demonstration” (PbD). The idea behind this approach is to program a robot by executing the task instead of programming it textually. The task demonstration is either performed in the real environment or in a virtual environment [2]. The latter approach has some disadvantages: it is not easy to navigate in a virtual environment and such programming does not support the work flow. Data gloves and head-mounted displays are not typically tools cooks or carpenters want to deal with while working. Alone the time needed to put on such devices restricts the applicability.

Katharina Soller · Dominik Henrich
Lehrstuhl für Angewandte Informatik III, Universität Bayreuth,
Universitätsstraße 30, 95447 Bayreuth, Germany,
e-mail: `{katharina.soller,dominik.henrich}@uni-bayreuth.de`

A further distinction may be made with regard to whether the demonstration is performed using the robot or not. In the former case, also called kinesthetic learning [5], the robot's internal sensors can be used for measuring trajectories, otherwise further sensors in the environment are required (e.g. cameras) or must be worn by the programmer while performing the demonstration (e.g. data gloves). The disadvantages of wearing sensors have already been mentioned above. Environmental disadvantages of sensors are possible occlusions and a restricted transportability. Our approach for PbD concentrates on the remaining alternative: measuring trajectories using the (internal) sensors of the robot while the user moves the robot via zero-force control. In general our approach is applicable to all of these approaches, as long as they make use of trajectories. Table 1 shows an overview of the different approaches and a small selection of references for each of them. A description of a system utilizing a robot model in a simulated environment for PbD could not be found in literature.

Table 1 Classification and evaluation of approaches to Programming by Demonstration (PbD).

	With Robot	*Without Robot*	
In real environment	[12] [10] [5] This paper	[7] [11]	– Need for highly structured environment and/or good object recognition techniques
In simulated environment	Just simulation, no PbD	[1] [2] [9]	– Need for integrating knowledge into simulated environment – Unwieldy VR-hardware
	– Need for high safety + Gives the user a better feeling for robot motions	– Need for adapting the human demonstration to the robot's kinematic + Independence of robot	

Work on PbD can also be classified with respect to the abstraction level [5]. High-level approaches regard complete actions and skills (see e.g. [7]), while low-level approaches concentrate on trajectories and basic robot actions. A high-level approach for loop structures can be found in [11].

Since the topic of our paper is the intuitive programming of spatial loops, composed of trajectories, it clearly involves a low-level approach. *Spatial loops* are repetitive movements of the robot that differ in one or more parameters (e.g. parameters for translation or rotation). For example, coating a surface consists of many similar movements (strokes of the brush) distributed equally over the surface. Such spatial loops may be programmed in a textual and ordinary manner by specifying three values for the loop parameter and a body that makes use of this loop parameter. Programming this task with the simple play back method, in which the robot directly repeats the demonstrated motion, requires one to perform each individual repetition. In addition, such an approach will not lead to a uniform loop due to demonstration insufficiency.

The system should be programmable via the following program flow structures, which are more than enough for writing general computer programs [3]: sequence, choice, loop, subroutine and break. The goal is to transfer the essential parts of general programming languages to intuitive robot programming, thus, inheriting the potential of computational completeness. In this paper, we concentrate on loops. With our approach it is possible to program a spatial loop in an intuitive way by demonstrating a portion of the desired task, for example the first, the second and the last cycle. The system then identifies the parameters of the loop, the values (start, step width and end) and the body, freeing the user from calculating or measuring these values. In addition such programming of loops makes sense even for applications which are required just once but contain many repetitions or for which the precision of the robot is needed to attain an exact distribution of the repetitive movements.

The rest of this paper is organized as follows: The next section gives an overview of the research related to PbD, especially low-level approaches. The third section briefly describes the proposed approach, where the component for loop programming should be embedded. The loops themselves and different ways to demonstrate them are covered in the fourth section. The fifth section formally defines the problem of deriving plausible spatial loops from a demonstration and presents a simple implementation. Based on this, we programmed translational spatial loops, and describe them in the sixth section.

2 Related Work

Our approach to PbD directly builds upon the playback (walk-through) method. This method is well known but its application was mainly restricted to early painting robots. Since these robots were heavy and dangerous, this kind of programming fell into misuse, but thanks to new technologies and standards these problems have been resolved [12].

Other problems associated with this method are demonstrations made insufficient by tremor and noise and high storage space requirements. [13] suggests smoothing of piecewise linear paths by removing unimportant points. Other approaches comprise B-Spline smoothing, Fourier transformation or Kalman-based smoothing (e.g. [6]). In [12] and [4] systems are described in which the user can modify the trajectory after demonstrating. Another approach to get rid of the insufficiency of a single demonstration is to interpolate several demonstrations. All these techniques just modify the trajectories and no interpretation of the user's intention can be drawn from them.

Some research addresses generalization of several demonstrations, allowing the robot to cope with uncertainties such as differing object positions. For example, the system described in [5] extracts the relevant features of trajectories (i.e. relative and absolute position constraints) from several varying demonstrations. The positions of the objects are obtained by a vision system. This way, the robot can apply the learned program to situations with new object positions. While this is a powerful method to generalize demonstrations, the number of needed demonstrations is dependent on the user's experience. In contrast to our work, this approach concentrates on the learning of task-specific constraints, while we concentrate on the kinesthetic definition of a loop structure.

To our knowledge, there is no approach that allows programming of looping movements by demonstrating just two or three movements. The idea is to give the user the possibility to implicitly define the variables and values needed for the execution of a loop structure.

3 Overview of System Concept

This section gives a brief overview of the system concept. The programmer uses the robot and a control panel to demonstrate the task he wants the robot to assume. For this purpose the demonstrator must be able to move the robot, e.g. via zero-force control, a joystick or a tracking device (for the latter, see e.g. [8]). Together with the signals from optional external sensors, this information forms the demonstration. The system then analyzes and interprets the demonstration and generates a corresponding robot program. The analysis and generation of the program are based on sets of control structures, movement primitives and variable types. The user then obtains a visualization of the spatial loop and can review the result.

4 Spatial Loops and Demonstration Types

The classical types of loops are condition-controlled loops (while loops), infinite loops and count-controlled loops (for loops)[2]. In the case of a *condition-controlled loop*, a Boolean variable or function is required that indicates e.g. if a button is pressed or a certain amount of time has passed. A loop is called *infinite loop* if there is no such condition or if this can never be met. In contrast to this kind of loops, *count-controlled loops* are stopped, when a variable – the counter – reaches a given end value. In all cases, the body of the loop either can be unchanged, i.e. the execution is a repetition of exactly the same behavior, or it can contain variations based on variables. In the case of condition-controlled and infinite loops, there is no general possibility to derive the break condition from the demonstration of two or three loop cycles, so for these loop types it is enough to determine the type and size of the step that is done between two consecutive cycles. In contrast, for count-controlled loops the number of cycles also has to be determined. For this reason we only regard count-controlled loops. The ideas are transferable to the other types.

In classical programming languages, a count-controlled loop consists of four components: three values for the loop parameter (initial value, boundary value and step size) and the body.

[2] Modern programming languages utilize further types of loops, e.g. for-each loops. We neglected these because they do not apply to the context of pure trajectories without further sensory input.

Table 2 Overview of loop types and their components.

	Condition-controlled loop	*Infinite loop*	*Count-controlled loop*			
			Demonstration Type 1	*Demonstration Type 2*	*Demonstration Type 3*	*Demonstration Type 4*
Condition	Boolean expression	–	–	–	–	–
Start value	For varying the body	For varying the body	First trajectory	First trajectory	First trajectory	(auto)
Step			Second trajectory	Second trajectory	(auto)	First trajectory
Number of steps	Break by condition	–	Numerical input	(auto)	Numerical input	Numerical input
End value			(auto)	Third trajectory	Third trajectory	Second trajectory

In the case of spatial loops, the three values consist of spatial variables (e.g. coordinates). The start value and end value of such a spatial loop can be defined by movements. Such a movement is called a sub-trajectory. The size of the step can be regarded as transformation on these sub trajectories. The body includes some robot commands that use the spatial variable and create the motion. There are basically four[3] alternatives for specifying a count-controlled spatial loop (Table 2). Depending on some circumstances, which we will examine next, it is possible to determine the last of the four values (start value s, step size δ, number of steps k and end value e) using the other three. The idea is outlined in 2D in Figure 1: The left side shows the different Demonstration Types 1 to 4. They all define the same spatial loop shown on the right. It is easy to see that in this case three of the four values are enough to determine the fourth with the following formula: $k \cdot \delta = e - s$.

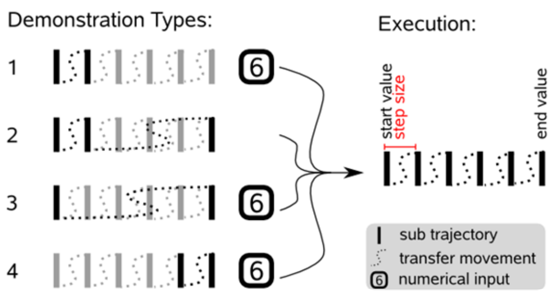

Fig. 1 Four alternatives for programming a translational loop consisting of six linear sub trajectories.

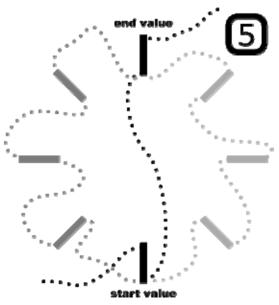

Fig. 2 Two possible spatial loops (dark grey and light grey) for Demonstration Type 3.

[3] Of course it would be possible to use all four components or even more sub-trajectories for the demonstration to increase robustness of the loop calculation, but we want to keep the demonstration as short as possible and focus on the minimum number of needed values.

We now look at the transformations that can be used for spatial loops. We do not consider the orientation of the tool center point here, i.e., we regard the orientation as fixed. In the context of this paper the transformation that describes how a trajectory for loop cycle $i + 1$ is derived from a trajectory for loop cycle i, can be translation and rotation, i.e. rigid body transformations. We do not regard reflection or scaling. Affine transformations that are more complex than the chosen transformations (e.g. shear) are also neglected because we regard them as less intuitive. We also restrict ourselves to linear interpolation for the generation of the loop, i.e. the steps between the cycles are all of the same size δ.

Considering the Demonstration Types 1, 3 and 4, we see that for translational transformations there is a unique mapping of the demonstrations to the spatial loops since a straight line is defined by two constraints. This does not hold for circular transformations, where three constraints are needed for a unique definition. Figure 2 shows an example with Demonstration Type 3. The input consists of two demonstrations (start value and end value) and the number of steps. Two spatial loops that both fit the demonstration are shown. Each rotation of a matching spatial loop around the axis defined by the start and end value leads to another matching spatial loop.

The situation is similar regarding the question whether a demonstration should be mapped to a translation or a rotation. Since a straight line can be considered as a circle with infinite radius, this distinction can not always be made with Demonstration Types 1, 3 and 4 without further knowledge.

On the one hand, since the use of always the same modality (e.g. position input) is more intuitive than multiple modalities (e.g. position and alphanumerical input), we suggest that Demonstration Type 2 is more intuitive than the other types. On the other hand, the other demonstration types have advantages as well. If the number of steps is essential for a certain application one of them might be the better choice.

5 From Demonstrations to Program Loops

In this section, the procedure of programming a spatial loop by Demonstration Type 2 and translational transformation is described. After instructing the system

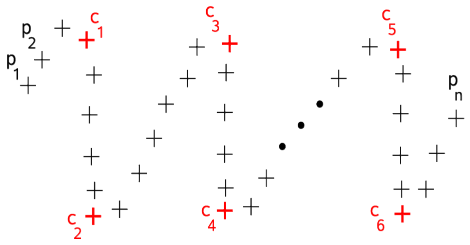

Fig. 3 Trajectory of Demonstration Type 2, divided into its sub-trajectories.

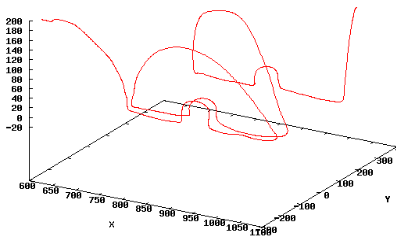

Fig. 4 The interpretation of the loop body depends on the set of motion primitives.

that a spatial loop is desired (e.g. via the control panel), the user demonstrates the first, the second and the last run of the loop and marks the beginning and the end of each cycle on the control panel, leading to a segmented trajectory that will then be analyzed.

At first the system has to decide whether the demonstration is a valid demonstration of Type 2. Since for Type 2 there have to be three demonstrated sub-trajectories, there must be six cuts, otherwise the system shows an error message.

5.1 Problem Description

A trajectory $r = (p_1, \ldots, p_n)$ is a sequence of n coordinates $p_l \in \mathbb{R}^d$, $l \in \{1, \ldots, n\}$, where d is the dimension of the coordinates. As mentioned above, we also have $m - 1$ cutting points c_j, $1 \leq j \leq m - 1$, on the trajectory, which divide the trajectory in sub-trajectories $r_1,\ldots r_m$. Let $m(i)$ be the number of cuts needed for demonstration type i.[4] Then there are $m(i) + 1$ sub-trajectories $r_1,\ldots,r_{m(i)+1}$ in a valid demonstration for Demonstration Type i. The trajectories we are interested in are those with an even index. All other trajectories are just transfer movements, which are neglected in this paper. Figure 3 shows an example with Demonstration Type 2.

Let R be the set of all trajectories in a given representation and T a set of transformations on these trajectories: $T = T = \{f \mid f: R \to R\}$. Let λ: $R \times R \to \mathbb{R}^{\geq 0}$ be a measure for the difference between two trajectories in R. Then the problem of finding a corresponding spatial loop to these trajectories can be stated as follows for Demonstration Type 2:[5]

$$\operatorname*{argmin}_{f \in T; k \in \mathrm{N}^{\geq 2}} \left(\alpha \lambda(f(r_2), r_4) + (1-\alpha)\lambda(f^k(r_2), r_6)\right), \text{ with } \alpha \in [0,1].$$

With the first term of the formula, the loop r_2 is fitted to the second sub-trajectory r_4 and with the other to the last one (r_6). With the parameter α, it is possible to change the weighting between these terms.

There are many degrees of freedom in this problem formulation: The representation of the trajectories, the set of transformations and the measures. The concrete transformation of a demonstration to a spatial loop is dependent on the choice of those. Since the purpose of this paper is to introduce the idea of spatial loop programming, we restrict ourselves to comparatively simple representations (Section 5.2), transformations and measures (Section 5.3).

5.2 Representation of Sub Trajectories

There are various options for representing trajectories. The most obvious possibility is to let them as they are – an ordered list of points. They also can be represented for example as parameterized primitives (e.g. straight lines, circles …), as Splines of different degrees or in Fourier space.

[4] $m(2) = 6$, $m(i) = 4$, $i \in \{1,3,4\}$;

[5] In the case of other demonstration types, this problem formulation differs slightly.

The body of a loop can consist of a single primitive motion or may be composed of different consecutive primitive motions. Of course, for a given sub-trajectory it depends on the set of primitive motions of the given system, whether it is atomic or composed.

The trajectory in Fig. 4 consists of three demonstrations for a loop (Demonstration Type 2). It is easy to see, that every sub-trajectory has a bulge in its center. Depending on the system's representation of sub-trajectories, this body can either be interpreted as one primitive (e.g. as a parameterized curve) or as composition (e.g. linear movements and a circle segment).

In this paper we start with the simplest loop bodies that consist of one single straight line represented by its two end points.

5.3 Transformations and Measures

The choice of the transformation is highly dependent on the representation of the sub-trajectories. If a sub-trajectory is defined via some points, the transformation can be any transformation that is applicable to points. Such a transformation might treat the different points of a sub-trajectory in a different manner or in the same way. The former allows the scaling of lines. In this paper we concentrate on translations and rotations.

For comparing two sub-trajectories, a measure is needed to describe the degree of similarity of the sub-trajectories. In the case of straight lines the most obvious measure is the sum of the Euclidean distances between the two pairs of corresponding end points. Like the transformations, the measure is also highly dependent on the representation of the sub-trajectories. Both have to be compatible with the representation.

5.4 Implementation

The concept was implemented on the industrial robot Stäubli RX130. The robot is equipped with a force/torque sensor (JR3 KMSi 90M31-63) on its wrist, allowing the user to demonstrate the trajectory. The cuts are set with a button on the teach pendant.

Given the trajectory and the cuts, the system has to decide what type of transformation fits best. Since we restrict ourselves to translation and rotation and do not consider combinations of them, we classify according to geometrical criteria. This section concentrates on transformations that work on the whole trajectory and not on the individual points.

Translation and rotation are rigid body transformations, so a common criterion for them is that the length of the strokes does no vary except for the user's imprecision.

For pure translations with Demonstration Types 1, 3 or 4 it must be checked, whether the two strokes have the same orientation.

For a Demonstration of Type 2 we regard the angle between the cutting points at the beginning and at the end of the sub-trajectories and the distance between them. Let γ_j be the point of a sub-trajectory that emerged from cut c_j and $\Theta_1, \Theta_2, \Theta_3 \in [0,1]$ be pre-set thresholds. To be classified as translation, a demonstration has to comply with the following conditions:6

$$\| \gamma_5 - \gamma_1 \| - \| \gamma_6 - \gamma_2 \| \leq \Theta_1 ; \quad \| \gamma_3 - \gamma_1 \| - \| \gamma_4 - \gamma_2 \| \leq \Theta_1 ;$$

$$< \frac{\gamma_5 - \gamma_1}{\| \gamma_5 - \gamma_1 \|}, \frac{\gamma_3 - \gamma_1}{\| \gamma_3 - \gamma_1 \|} > \geq \Theta_2 ; \quad < \frac{\gamma_6 - \gamma_2}{\| \gamma_6 - \gamma_2 \|}, \frac{\gamma_4 - \gamma_2}{\| \gamma_4 - \gamma_2 \|} > \geq \Theta_3 ;$$

The threshold Θ_1 should be near zero while the others should be near one. The first two formulas assure that both points are translated equally. If they were not, the transformation can not be a translation. The other two formulas measure the angles $\sphericalangle(\gamma_3, \gamma_1, \gamma_5)$ respectively $\sphericalangle(\gamma_4, \gamma_2, \gamma_6)$. If

$$\text{round}\left(\frac{\| \gamma_5 - \gamma_1 \|}{\| \gamma_3 - \gamma_1 \|} \right) = \text{round}\left(\frac{\| \gamma_6 - \gamma_2 \|}{\| \gamma_4 - \gamma_2 \|} \right) = k,$$

there are $k + 1$ loop passes altogether. The new points of the spatial loop can now be calculated easily from the direction of $\gamma_5 - \gamma_1$ and k. A similar condition can be established for rotations but it is skipped here for lack of space.

As soon as a demonstration is assigned to a transformation, the points for all loop cycles can be calculated and used for the generation of the robot movement. The transfer movements consist of an approach to the first point of a line and a depart movement from the second point.

6 Experiments and Results

We demonstrated translational spatial loops with this system with Demonstration Types 1 to 3. We skipped Demonstration Type 4, since it is symmetrical to Demonstration Type 1. The robot was programmed to draw six straight lines with a resilient ball pen. The demonstrations were carried out via the zero-force control and the cutting points were set via the teach pendant. Figure 5 schematically shows the strokes the robot should be programmed to draw. At first, the robot drew a master on a fixed piece of paper. This provided the basis for measuring the deviation of the intuitively programmed loop from the textually programmed loop used for the master. We took the Euclidean distance in x- and y-direction (i.e. the points were projected onto a plane parallel to the table and perpendicular to the axis of the pen) as measure. The deviation in the z-direction was not included because of the compliance of the pen. The user then programmed the loop by repeating the strokes as accurately as possible by grabbing the pen (see Figure 6). For measuring the deviation we calculated the mean of all deviations of corresponding points of the ideal loop and the programmed loop. For each experiment and demonstration type, six experiments were conducted.

6 The symbol ||•|| denotes the Euclidean norm and <•,•> is the inner product.

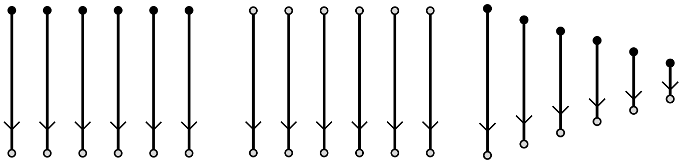

Fig. 5 Three experiments: pure point-based translation, pure translation working on the whole line and scaling of the lines by point-based translation.

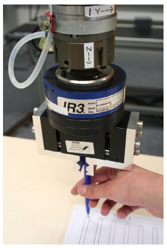

Fig. 6 The user grabs the pen and moves the robot.

In the first two experiments the strokes are simply translated. The first experiment was conducted with a point-based transformation, i.e. the translations for the start points and the end points are deduced separately. For the second experiment an implementation was chosen that calculated the translation vector from the mean of the two vectors derived with the implementation from the first experiment. The third experiment was conducted only for point-based transformation. With each cycle, the stroke becomes shorter, the points approach each other. This is a kind of scaling, but it is the result of the point-based transformation. This could not be achieved by a rigid body transformation that works on the whole line.

The diagrams in Figure 7 show the mean error of all points of each experiment. It is in evidence that Demonstration Type 1 is much more error-prone, since the error in the demonstration accumulates with the number of cycles while for Type 3 the error of the execution is limited by the error in the demonstrations. Demonstration Type 2 is dependent on the choice of parameter α. In our experiment, α = 0.5, since we take the mean of the two vectors. For α = 0, Demonstration Type 2 is like Demonstration Type 3 except that sub-trajectory 4 is used to derive the number of cycles instead of numerical input. For α = 1 the same holds with Demonstration Type 1 and sub-trajectory 6. This explains why Demonstration Type 2 lies in between the other two demonstration types in most cases.

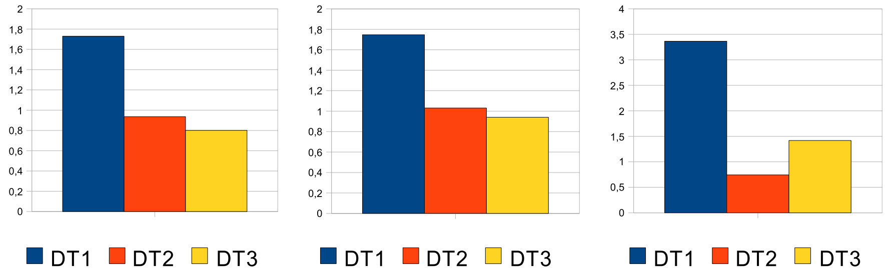

Fig. 7 The mean deviation in mm for the three experiments and for Demonstration Types 1, 2 and 3, measured in the x-y-plane.

7 Conclusions

We introduced the concept of intuitive spatial loop programming, where the parameters of a loop are derived automatically. We identified four demonstration types and compared them according to the uniqueness of the demonstration and the intuitiveness. Demonstration Type 2 turned out to be less ambiguous than the other types. After the comparison we analyzed the different components needed for spatial loop programming and presented possible candidates to be used. We implemented the concept and showed the applicability with an experiment.

Future work will focus on further kinds of transformations, particularly on combined transformations. Pushing a button to segment the sub-trajectories turned out to be annoying and error-prone, so an automatic segmentation will be considered. Since spatial loop programming was only tested on pure trajectories we intend to augment the concept with further sensor data such as forces.

References

1. Acker, J., Kahl, B., Henrich, D.: Environment Guided Handling of Deformable Linear Objects: From Task Demonstration to Task Execution. In: 37th International Symposium on Robotics (ISR, 4th German Conference on Robotics), Robotik, München, Germany (2006)
2. Aleotti, J., Caselli, S., Reggiani, M.: Toward Programming of Assembly Tasks by Demonstration in Virtual Environments. In: 12th IEEE Workshop Robot and Human Interactive Communication, San Francisco, CA, October 31, November 2 (2003)
3. Böhm, C., Jacopini, G.: Flow diagrams, turing machines and languages with only two formation rules. Communications of the ACM 9(5), 366–371 (1966)
4. Brageul, D., Vukanovic, S., MacDonald, B.A.: An Intuitive Interface for a Cognitive Programming By Demonstration System. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 3570–3575 (May 2008)
5. Calinon, S., Guenter, F., Billard, A.: On Learning, Representing, and Generalizing a Task in a Humanoid Robot. IEEE Trans. on Systems, Man and Cybernetics, Part B 37(2), 286–298 (2007)
6. Croitoru, A., Agouris, P., Stefanidis, A.: 3D Trajectory Matching By Pose Normalization. In: ACM international workshop on Geographic Information Systems, pp. 153–162 (November 2005)
7. Dillmann, R., Rogalla, O., Ehrenmann, M., Zöllner, R., Bordegoni, M.: Learning Robot Behaviour and Skills based on Human Demonstration and Advice: the Machine Learning Paradigm. In: 9th International Symposium of Robotics Research (ISSR 1999), Snowbird, UT, USA, October 1999, pp. 229–238 (1999)
8. Hein, B., Hensel, M., Wörn, H.: Intuitive and Model-based On-line Programming of Industrial Robots: A Modular On-line Programming Environment. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 3952–3957 (May 2008)
9. Kahl, B., Henrich, D.: Virtual Robot Programming for Deformable Linear Objects: System concept and Prototype Implementation. In: 12th International Symposium on Measurement and Control in Robotics (ISMCR 2002), Bourges/France (June 2002)

10. Mayer, H., Burschka, D., Knoll, A., Braun, E.U., Lange, R., Bauernschmitt, R.: Human-Machine Skill Transfer Extended by a Scaffolding Framework. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 2866–2871 (May 2008
11. Pardowitz, M., Glaser, B., Dillmann, R.: Learning Repetitive Robot Programs From Demonstrations Using Version Space Algebra. In: The 13th IASTED International Conference on Robotics and Applications (2007), `http://wwwiaim.ira.uka.de/data/File/Publications/RA07-VSA.pdf`
12. Schraft, R.D., Meyer, C.: The Need for an Intuitive Teaching Method for Small and Medium Enterprises. In: ISR 2006 - ROBOTIK 2006: Proceedings of the Joint Conference on Robotics, May 15-17, Munich (2006)
13. Waringo, M., Henrich, D.: Efficient Smoothing of Piecewise Linear Paths with Minimal Deviation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3867–3872 (August 2006)