Robot Programming - From Simple Moves to Complex Robot Tasks

F. M. Wahl and U. Thomas
Institute for Robotics and Process Control
Technical University of Braunschweig

1 Introduction

The development of robot programming concepts is almost as old as the development of robot manipulators itself. As the ultimate goal of industrial robotics has been (and still is!) the development of sophisticated production machines with the hope to reduce costs in manufacturing areas like material handling, welding, spray-painting and assembly, tremendous efforts have been undertaken by the international robotics community to design user-friendly and at the same time powerful programming methods. The evolution reaches from early control concepts on the hardware level via point-to-point and simple motion level languages to motion-oriented structured robot programming languages. Comprehensive surveys on the historical development of robot programming may be found in [1, 2]. A characteristic feature of robot programming is, that usually it is dealing with two different worlds, (ref. to Fig. 1): (1) The real physical world to be manipulated, and (2) abstract models representing this world in a functional or descriptive manner by programs and data. In the simplest case, these models are pure imagination of the programmers; in high level programming languages, e.g. it may consist of CAD data. In any case, commands based on some model are causing robots to change the state of the real world as well as the world model itself. During a sequence of actions both worlds have to be kept consistent to each other. This can be ensured by integrating internal robot sensors as well as external sensors like force/torque and vision sensors. Already in the early seventies research groups started to focus on so-called task-oriented robot programming languages. One of the first examples are IBM's AUTOPASS system [3], the RAPT language developed at the University of Edinburgh [4] and the LAMA system proposed by MIT [5].

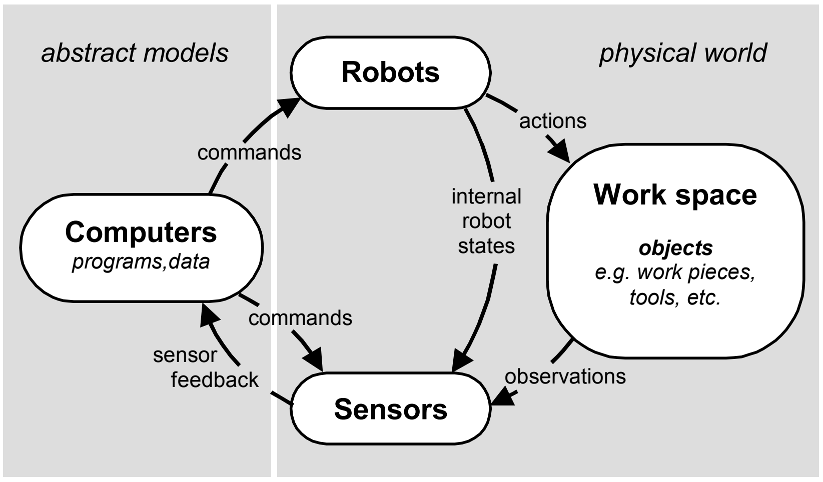


Figure 1: General robot programming paradigm.

The basic idea behind these approaches is, to relieve the programmer from knowing all specific machine details and free him from coding every tiny motion/action; rather, he is specifying his application on a high abstraction level, telling the machine in an intuitive way *what* has to be done and not *how* this has to be done. This implicit programming concept implies many complex modules leading to automated robot programming. E.g., there is a need for user-friendly human interfaces for specifying robot applications; this may range from graphical specifications/annotations within a CAD environment, till to spoken commands or gestures, interpreted by some speech understanding or vision system respectively. These commands have to be converted automatically into a sequence of actions/motions by a task planning system; at the Technical University of Braunschweig we developed ^{High}LAP [6, 7], which will be outlined below.

During the course of the years it turned out, that the most difficult part in automatic robot programming is the *execution and control* of automatically generated action/motion sequences including automated generation of collision-free paths and force controlled mating operations. Besides some first simple industrial applications, like in 4-DOF assembly of PCBs, this still is subject of international ongoing research. The necessary prerequisites from control theory have been developed during the last decades; they are ready to be used. Already in the early eighties Mason has shown, how to control robots, when they are in contact with their environment [8]. His concept is outlined below, when we are describing the concept of skill primitives as versatile interface between programming and control. However, before diving into advanced issues of implicit robot programming, we briefly will discuss the state of the art of explicit, i.e. motion oriented programming techniques in the following section.

2 Modern Explicit Programming Concepts

The development of modern motion-oriented robot programming languages started in the mid seventies. Languages like VAL [9] (the predecessor of Adept's V+) and AML [10] are examples of early structured robot programming languages, which already incorporate sophisticated data structures. As some robot vendor's proprietary languages shipped today are still far behind these early developments, many research laboratories developed their own languages. Most of these languages are extensions of wide-spread programming languages. E.g., at our laboratory we embedded robot functionality in C and C++ to build the object oriented robot programming language ZERO++ [11]. Motion-oriented robot programming languages nowadays are indispensable in industrial robot applications; in research they often constitute the basis of higher level robot programming concepts.

One of the essential ingredients of modern robot programming languages is the thorough usage of the frame concept. I.e., all robot poses and object locations as well as motions are expressed in accordance with human spatial intuition in terms of Cartesian coordinates. By using homogeneous coordinates, translations and rotations can be computed by multiplying points or coordinate systems in 3D Euclidian space with one single 4x4 transform matrix. As a matter of course, languages using the frame concept should supply programmers with a multitude of built-in functions to specify such transforms. ZERO++, e.g., provides many functions to convert x,y,z-coordinates and/or Euler angles or RPY angles into transform matrices and vice versa. Operator overloading and robotics specific math-functions allow a simple notation of transform matrix equations, etc. For the application programmer (who at least in the industrial world usually is not an expert of robotics) the details of the robot hardware have to be hidden behind well-defined easy-to-use software interfaces. Respecting this, from the application programmer's point of view, also the difference between programming serial or parallel robots should diminish!

Usually, advanced applications are heavily dependent on sensor integration of internal as well as external sensors, like force/torque and vision sensors. The robot programming implications of this led in the early eighties to the development of the so-called monitor concept [12]. Monitors are small pieces of concurrent user-defined or built-in programs, heavily communicating with the users' applications and the motion pipelines of robot control systems. I.e., monitors are reading a specific sensor or sets of sensors in specified time intervals; in dependency of the sensor values the monitors are modifying via the motion pipeline the robots' paths *'guided motion'* or are triggering some action (e.g., an immediate stop of motion, *'guarded motion'*). Usually, sensor integration requires from the robot programmer an in-depth understanding of the robots' functionality. Thus, it is very important to supply programmers with powerful programming language constructs to ease such difficult tasks. Fig. 2 shows a simple path following example keeping a constant distance between the robot's tool center point and the surface of the corrugated sheet of iron. As can be seen, this application can be coded with just a few lines of ZERO++ code. The main section simply defines 'start' and 'goal' positions. After moving the robot to the 'start' position in joint interpolation mode, it is moved in Cartesian interpolation mode to the 'goal' position while a 'Monitor' has been activated. The 'Monitor' function is reading ultrasonic sensor values, which are used to compute 'delta' frames used by the ZERO++ Kernel to continuously modify interpolated frame values between 'start' and 'goal'. In similar ways any functional dependencies of some path properties (speed, force, torque, distance, etc.) can be specified in a textual programming manner, which, however, is cumbersome and error-prone. Unfortunately, up to now there is a lack of off-line tools supporting robot programmers to specify robot path properties comfortably.

```
void GuidedMotion()
{
   RX60 robot();
   FRAME start=Trans(100.0, 20.0, 70.0);
   FRAME goal=start*Trans(0.0, 400.0, 0.0);

  robot.Move(start,ROBOT::JointInterp);
  robot.Move(goal,MonitorFunction,ROBOT::FrameInterp);
}

int MonitorFunction(FRAME &delta)
{
  static USSENSOR sensor;
          double        height;

   height=sensor.GetValue();
   if(height!=DIST)
      delta.TransZ(DIST – height);
   else
     return IGNORE;
  return CHANGE;
}
```

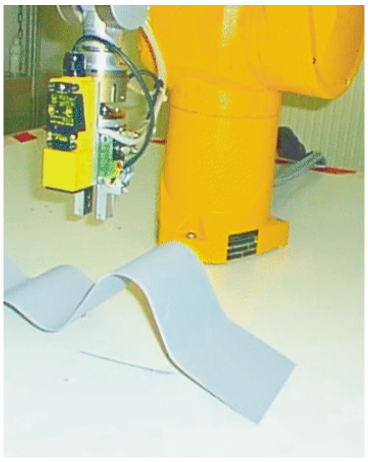
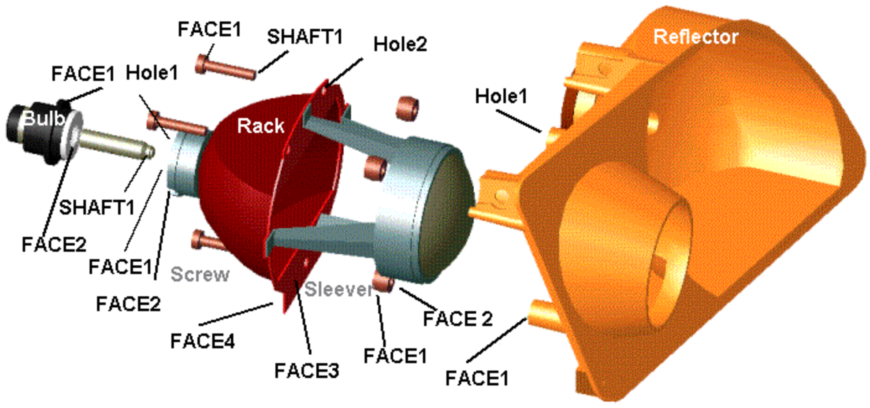


Figure 2: Usage of monitors for sensor guided motion.

The skill primitives discussed below may be considered as an extension and generalization of the monitor concept.

3 CAD Based Program Specification

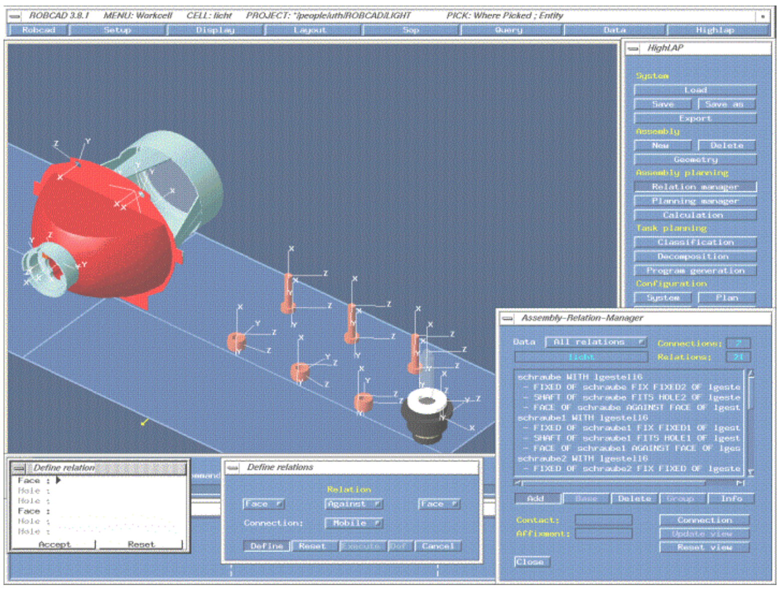
As mentioned above, for CAD based program specification sophisticated, intuitive, and easy to use software tools are necessary. As the design engineer knows about the geometry of the objects to be assembled, the appropriation of some kind of symbolic spatial relations (SSR) is obvious. Ambler and Popplestone [13] defined a set of SSRs, which consists of four different relations. E. g. to specify, that the shaft axis of the bulb shown in Fig. 3 should align with the hole axis of the rack, a design engineer may specify the symbolic spatial relation ‘SHAFT1 OF BULB FITS HOLE1 OF RACK’. This implies, that only two degrees of freedom between the two objects remain: a possible rotation around the hole axis and a translation along the same axis. To reduce further degrees of freedom, the user might add other symbolic spatial relations for instance a ‘FACE AGAINST FACE’ relation. In this case it means, that the lower ‘FACE2 OF BULB’ lies against ‘FACE2 OF RACK’, which is inside the socket and is perpendicular to the hole axis. Herewith, just one degree of freedom is left, namely a rotation around the hole axis. To specify the assembly group completely, every degree of freedom has to be eliminated in a similar way. Fig. 3 shows some parts of an automotive headlight assembly with corresponding SSRs. The easiest way to generate the spatial relations explicitly, is to interactively let the design engineer click on the surfaces, e. g. shafts, holes and faces in his CAD environment in order to specify appropriate features (i. e. coordinate systems) and subsequently allow him to select suitable relations between these features, e. g. fits, against, coplanar. Fig. 4 displays the user interface of the assembly planner ^{High}LAP, which has been embedded in the commercial robot simulation system Robcad. The system supports the user while he is specifying the relations in such a manner, that it signals contradictory specifications and shows the user remaining degrees of freedom between assembly objects.



FACE1 OF BULB	AGAINST	FACE1 OF RACK
FACE2 OF BULB	AGAINST	FACE2 OF RACK
SHAFT1 OF BULB	FITS	HOLE1 OF RACK
HOLE2 OF RACK	ALIGNED	HOLE1 OF REFLECTOR
FACE3 OF RACK	AGAINST	FACE1 OF SLEEVER
SHAFT1 OF SCREW	FITS	HOLE2 OF RACK
FACE1 OF SCREW	AGAINST	FACE4 OF RACK
FACE2 OF SLEEVER	AGAINST	FACE1 OF REFLECTOR

Figure 3: Parts of a headlight assembly with SSRs

The symbolic spatial relations specifying an assembly also can be used for the automatic calculation of possible assembly plans as well as for planning of appropriate sensors, which may guide the assembly process during execution. This kind of specification provides an easy to use interactive graphical tool to define any kind of assembly; the user has to deal only with a limited and manageable amount of spatial information in a very comfortable manner.



```
schraube WITH lgestell6
 - FIXED OF schraube FIX FIXED2 OF lgeste
 - SHAFT OF schraube FITS HOLE2 OF lgeste
 - FACE OF schraube AGAINST FACE OF lgest
schraube1 WITH lgestell6
 - FIXED OF schraube1 FIX FIXED1 OF lgest
 - SHAFT OF schraube1 FITS HOLE1 OF lgest
 - FACE OF schraube1 AGAINST FACE OF lges
schraube2 WITH lgestell6
 - FIXED OF schraube2 FIX FIXED OF lgeste
```

Figure 4: Specification interface of the assembly planner ^{High}LAP

4 Towards Automated Robot Programming

An ultimate long-term aim is an almost automated programming and execution of assembly processes. A system should decide which order is the most appropriate order for assembly considering certain criteria like geometric feasibility, the mechanical stability of subassemblies, the degree of parallel execution, the number of necessary reorientations of parts during execution, types of fixtures, tools etc. An assembly consisting of n-parts in general may have up to $(2^{n-1}-1)$ possible assembly sequences. Thus, automatic generation of assembly sequences has been shown to be a NP-complete problem [14]. In the assembly planning community, the *assembly by disassembly* philosophy became generally accepted [15]. Hence, starting from the complete assembly and removing one part after another leads to some sequences in reverse order. Evaluating these sequences considering the above mentioned criteria yields an optimal assembly sequence. For some parts of the indicator light plus headlight assembly the calculated result is depicted in Fig. 5.

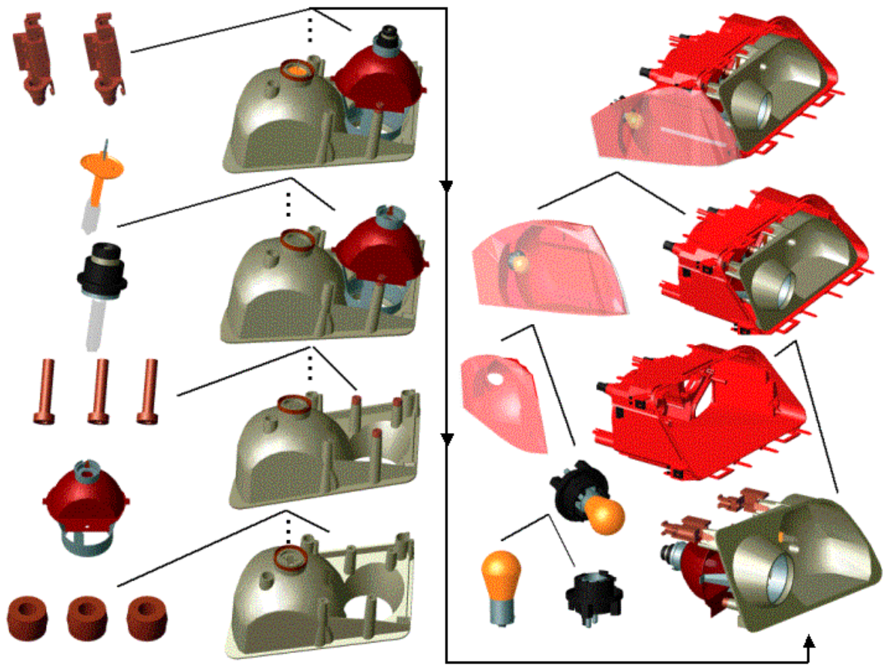


Figure 5: The assembly plan for parts of an automotive headlight and indicator light assembly

As mentioned above, while removing parts of the assembly, a sophisticated system has to evaluate the geometric feasibility of removal operations carefully. The symbolic spatial relations explained in the previous section could be used to compute possible mating directions. A superior method, which we are developing currently, applies the configuration space approach proposed by [16] to 3D. Fig. 6 right shows a bulb light and its corresponding bayonet socket of an automotive indicator light. Based on the configuration space representation (Fig. 6 left) a suitable mating direction for the bulb can be computed [17].

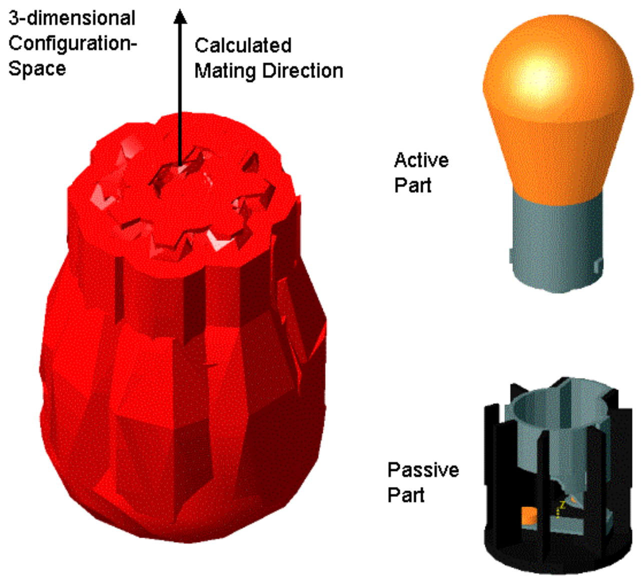

3-dimensional Configuration-Space
Calculated Mating Direction
Active Part
Passive Part

Figure 6: Calculation of mating direction for the bayonet securing task

Once a suitable assembly sequence has been generated each hyperarc in the graph shown in Fig. 5 represents an assembly cycle, which contains a grasp strategy, a transfer motion path and a mating/joining strategy. Focusing on the mating/joining strategy, each hyperarc has to be considered as a complex robot task. An automatic robot programming system has to recognize the correct robot task type and should map it to a sequence of elemental robot operations [18, 6]. Robot tasks may range from simple object placing tasks, screwing tasks, peg in hole tasks, to rather complex tasks like bayonet securing tasks. The applicable robot tasks are designed and programmed off-line and stored as functional modules. They automatically can be loaded and parameterized (goal positions, mating directions, etc.) by the assembly planner. While execution is in progress, the robot comes into contact with its environment. Thus, determination of contact state transitions is essential. Due to uncertainties in the work space, which may arise from modelling with limited precision or unpredictable displacements, all possible contact state transitions need to be represented in a so-called skill primitive net. During execution of the robot task one path through the skill primitive net is traversed. Each time a skill primitive (described in detail below) is executed by the robot control system, it changes the contact state of the moved object with its environment.

5 Skill Primitives – Interface Between Programming and Control

As mentioned above, skill primitives can be considered as a sophisticated interface between textual or automated robot programming and robot control. Skill primitives extend the known compliance frame concept introduced by Mason [8]. In order to uniquely determine robot motions, positions/orientations, velocities (linear and rotational) and forces/torques in 6 DOF have to be specified in terms of a 6x6 *compliance frame* matrix **C**. The reference coordinate system for the 6 DOFs is called *center of compliance* **CoC;** it is defining the task space. For a compliant motion, some of these DOFs are constrained, others may be free. E.g., for the classical peg in hole problem along the peg's rotational z-axis, the diagonal of **C** equals

$$\begin{aligned}\text{diag}\,\mathbf{C} &= (x_{trans}, y_{trans}, z_{trans}, x_{rot}, y_{rot}, z_{rot})^T \\ &= (constrained, constrained, free, constrained, constrained, free)^T\end{aligned}$$

For sake of simplicity we assume the **CoC** at the tip of the peg. A simple control of an insertion (neglecting the possibility of jamming) could be, to select the following desired values for robot control in the Cartesian task space

$$\begin{aligned}\text{diag}\,\mathbf{C} = (&x_{force} = 0\text{ N}, y_{force} = 0\text{ N}, z_{velocity} = 0.01\text{ m/s}, x_{torque} = 0\text{ Nm}, \\ &y_{torque} = 0\text{ Nm}, z_{orientation} = 0.1\text{ rad})^T\end{aligned}$$

As no time limit for the motion is specified, the robot holding the peg would collide with the object containing the hole at some point. I.e., for proper operation, in addition to the **CoC** and **C** some *stopping condition* **s** has to be specified, which may be defined as some Boolean expression. In our simplified example this could be

$$\mathbf{s} = (z_{position} > 340\text{ mm} \quad \text{OR} \quad z_{orientation} > 0.1\text{ rad} \quad \text{OR} \quad \text{time\_out} > 5\text{ s})$$

In our laboratory currently we are developing a Cartesian control architecture able to interpret commands in the above given form. The basis of our implementation is a flexible and modular middleware for robot control and automation applications. First skill primitives, e.g. for placing polyhedral objects on some unknown surface, have been developed by means of Cartesian so-called add-on force/torque controllers [19].

6 An Example of a Force Controlled Robot Task

In order to prove the suitability and the strength of skill primitive based robot task implementation, we have chosen a bayonet securing task of an automotive light bulb. In our experimental set up, we have used a Stäubli 6 DOF robot, equipped with an external JR3 force/torque sensor mounted on the robot's wrist. The robot's control unit is connected via TCP/IP to a PC equipped with the JR3 interface card; the PC is running the control process. The Stäubli robot control system receives and executes each 16 ms a move operation.

Fig. 7 shows the light bulb (4) to be secured by means of a bayonet socket (6). Assuming, that the rotational axis of the bulb is fairly aligned with the axis of the socket, the decomposition of the robot task into skill primitives leads to the following four states (Fig. 8): Moving from free space into contact state (a) until the first electrical contact (1) is pressed by base (4) of the bulb; (ref. to Fig. 7), while minimizing lateral forces arising from small displacements. The next skill primitive moves the bulb further down in z-direction until the central electrical contact (2) is pressed with 15N.

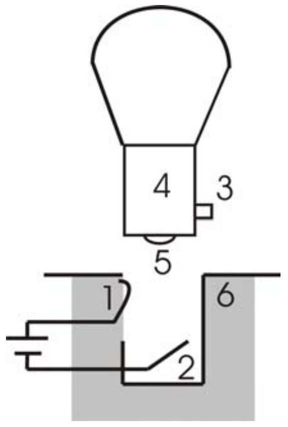


Fig 7: Light bulb with bayonet socket

Force oscillations with up to 5N are due to friction and due to the slow robot control loop available in our current experimental set up. During execution of the next skill primitive the spring remains pressed while the robot is changing the bulb's orientation around the z-axis (until a high torque is produced, because nipple (3) of the bulb has reached its stop position in the socket) yielding state (c). Aiming the final position corresponding to state (d), the robot pulls the bulb slightly backwards, so that the bulb is properly secured into the socket. Measurements of forces and torques during this sequence of skill primitives are shown in Fig. 9.

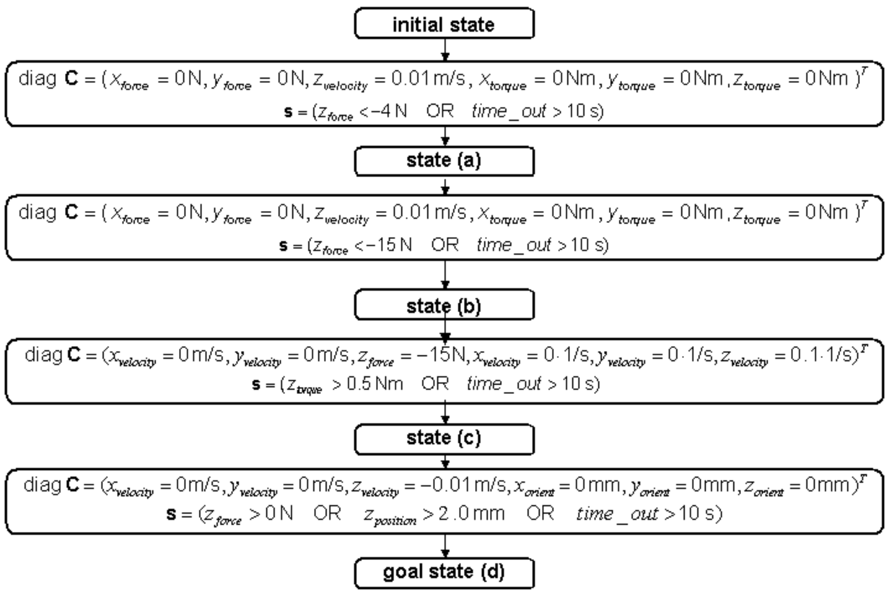
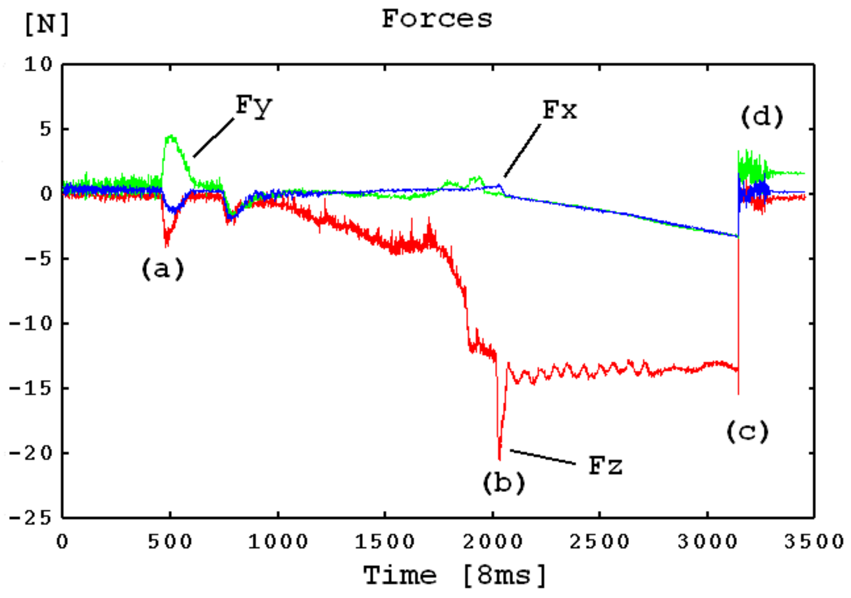

initial state
diag $\mathbf{C} = (x_{force} = 0\,\mathrm{N}, y_{force} = 0\,\mathrm{N}, z_{velocity} = 0.01\,\mathrm{m/s}, x_{torque} = 0\,\mathrm{Nm}, y_{torque} = 0\,\mathrm{Nm}, z_{torque} = 0\,\mathrm{Nm})^T$
$\mathbf{s} = (z_{force} < -4\,\mathrm{N}$ OR $time\_out > 10\,\mathrm{s})$
state (a)
diag $\mathbf{C} = (x_{force} = 0\,\mathrm{N}, y_{force} = 0\,\mathrm{N}, z_{velocity} = 0.01\,\mathrm{m/s}, x_{torque} = 0\,\mathrm{Nm}, y_{torque} = 0\,\mathrm{Nm}, z_{torque} = 0\,\mathrm{Nm})^T$
$\mathbf{s} = (z_{force} < -15\,\mathrm{N}$ OR $time\_out > 10\,\mathrm{s})$
state (b)
diag $\mathbf{C} = (x_{velocity} = 0\,\mathrm{m/s}, y_{velocity} = 0\,\mathrm{m/s}, z_{force} = -15\,\mathrm{N}, x_{velocity} = 0 \cdot 1/\mathrm{s}, y_{velocity} = 0 \cdot 1/\mathrm{s}, z_{velocity} = 0.1 \cdot 1/\mathrm{s})^T$
$\mathbf{s} = (z_{torque} > 0.5\,\mathrm{Nm}$ OR $time\_out > 10\,\mathrm{s})$
state (c)
diag $\mathbf{C} = (x_{velocity} = 0\,\mathrm{m/s}, y_{velocity} = 0\,\mathrm{m/s}, z_{velocity} = -0.01\,\mathrm{m/s}, x_{orient} = 0\,\mathrm{mm}, y_{orient} = 0\,\mathrm{mm}, z_{orient} = 0\,\mathrm{mm})^T$
$\mathbf{s} = (z_{force} > 0\,\mathrm{N}$ OR $z_{position} > 2.0\,\mathrm{mm}$ OR $time\_out > 10\,\mathrm{s})$
goal state (d)

Figure 8: The skill primitive sequence of the bayonet securing task



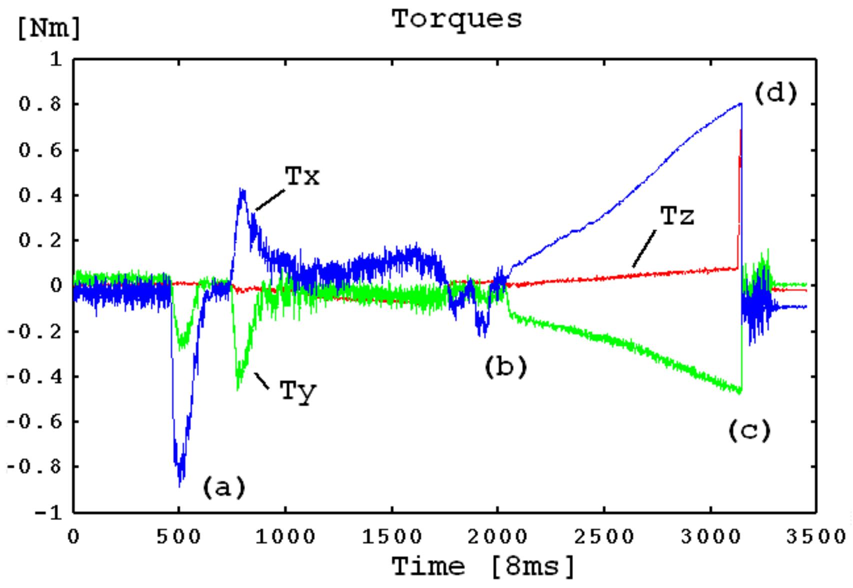


Figure 9: Measured forces and torques during the execution of bayonet securing task

7 Conclusion

The purpose of this paper subject to limited space has been two-fold: (1) Revisiting major developments and concepts in robot programming and discussion of its current state and (2) indicating insufficiencies and further developments. As has been pointed out, there is a tremendous gap between commercially available robot programming languages and methods developed in research laboratories all over the world. Although powerful motion oriented programming concepts – like the above mentioned monitor concept - are known since almost three decades, they rarely have found their way to products. Most commercial systems still offer simple set_value commands for specifying properties of robot paths, like set_speed(…), set_force(…), etc. specifying the properties of the next path segment(s) by some given (constant) parameters. To our knowledge, there is no single commercial robot programming language available, allowing flexible specification of functional interdependencies of path properties, e.g., applied forces/torques as function of position/orientation, etc. Since the early work of Mason we know, how such principles can be implemented in a task oriented manner. In the meanwhile, there also is a huge body of control concepts available, ready to support functional sensor guided motion applications. In our opinion, the skill primitive concept outlined above offers a versatile interface between programming and control. However, we recognize a big backlog for developing tools to support motion-oriented sensor-guided *robot programming by hand*.

As has been pointed out, skill primitives also can serve as a powerful interface between *automated robot programming* and robot control. Whereas the art of assembly planning already has reached a mature level, autonomous execution of automatically generated assembly plans still is in its infancy. In this paper and elsewhere we have shown some first examples, how to implement basic robot tasks (placing objects, securing light bulbs into bayonet sockets, etc.); they are ready to be used in automated assembly robot programming environments. Nevertheless, there are yet many open issues to be solved in order to achieve fully automated assembly *planning/programming* and sensor guided *execution* of assembly processes. Certainly, one big important research area in the future will concentrate on automated *sensor planning* simultaneously performed with *assembly planning*. We expect fully automated robot programming and execution for complex assemblies, robust enough to be applied in industrial applications, will not become true before the end of this decade.

References

[1] C. Blume, W. Jakob: Programmiersprachen für Industrieroboter. Vogel-Verlag , 1983.

[2] P. G. Ránky, C. Y. Ho: Robot Modelling – Control and Applications with Software. IFS Ltd./Springer, 1985.

[3] L. Lieberman, M. Wesley: AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly. IBM J. Res. Dev. Vol. 21, No. 4 , 1977.

[4] R. Popplestone, A. Ambler, I. Bellos: An Interpreter for a Language Describing Assemblies. Artificial Intelligence, Vol. 14, No. 1, 1980.

[5] T. Lozano-Pérez, P. H. Winston: LAMA: A Language For Automatic Mechanical Assembly. International Joint Conference. Artificial Intelligence, 1987.

[6] H. Mosemann, F. M. Wahl: Automatic Decomposition of Planned Assembly Sequences Into Skill Primitives. IEEE Transactions on Robotics and Automation, Vol. 17, No. 5, 2001.

[7] U. Thomas, F. M. Wahl: A System for Automatic Planning, Evaluating and Execution of Assembly Sequences for Industrial Robots. IEEE/JR International Conference on Intelligent Robots and Systems , 2001.

[8] M. T. Mason: Compliance and Force Control for Computer Controlled Manipulators. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 11, No. 6, 1981.

[9] B. E. Shimano: VAL: A Versatile Robot Programming and Control System. COMPSAC 79, 1979.

[10] R. H. Taylor, P. D. Summers, J. M. Meyer: AML: A Manufacturing Language. International Journal Robotics Research Vol. 1, No. 3, 1982.

[11] C. Pelich, F. M. Wahl: ZERO++: An OOP Environment for Multiprocessor Robot Control. International Journal Robotics and Automation, Vol. 12, No. 2, 1997.

[12] M. A. Lavin, L. I. Lieberman: AML/V: An Industrial Machine Vision Programming System. International. Journal Robotics Research Vol. 1, No. 3, 1982.

[13] A. P. Ambler, R. J. Popplestone: Inferring the Positions of Bodies from Specified Spatial Relationships. Artificial Intelligence, Vol. 6, 1975.

[14] L. Kavraki, J.-C. Latombe, R. H. Wilson: On the Complexity of Assembly Partitioning. Information Processing Letters, Vol. 48, 1993

[15] L. S. Homen de Mello and S. Lee: Computer-Aided Mechanical Assembly Planning. Kluwer Academic Publisher, 1991

[16] T. Lozano-Pérez: Spatial Planning: A Configuration Space Approach. IEEE Transactions On Computers, Vol. C-32, No. 2, 1983.

[17] U. Thomas, M. Barrenscheen, F. M. Wahl: Efficient Calculation of Mating Directions Based on Configuration Spaces. To be published elsewhere.

[18] T. Hasegawa, T. Suehiro, K. Takase: A Model-Based Manipulation System with Skill-Based Execution. IEEE Transactions on Robotics and Automation, Vol. 8, No. 5, 1992

[19] B. Finkemeyer, T. Kröger, F. M. Wahl: Accurate Placing of Polyhedral Objects in Unknown Environments. To be published elsewhere.