Robotica (1999) volume 17, pp. 87–95. Printed in the United Kingdom © 1999 Cambridge University Press

A modular and sensor-oriented motion planner

J. Luthiger* and E. Burdet†

(Received in Final Form: March 2, 1998)

SUMMARY

The motion planner presented in this paper enables the user to design movements in an interactive way, and to integrate sensor signals efficiently. In addition to the usual movements bound to a path, which can now be combined like lego pieces, smooth and fast reactions to arbitrary changes of the target are also provided. This motion planner opens new possibilities for flexible and sensor-oriented robotics, as examples from its implementation on two robots show.

KEYWORDS: Motion planner; Sensor signals; Flexible robotics; Modularity.

1. INTRODUCTION

1.1. Motivation

In order to use robots for more complex tasks than pick-and-place, it is necessary to provide them with the ability to use sensor information efficiently. In addition, robots will be more widely used and better accepted when the user possesses simple ways for designing motion planning. This paper presents a *modular sensor-oriented motion planner*, which corresponds to these needs. This planner provides frame conditions for a high level of sensor integration and enables the user to design the motions step by step and interactively.

1.2. Sensor integration

The described motion planner is part of a modular and sensor-oriented robot control structure.[1] The motion planning is closely related to the flow of sensor information, which will be described in this subsection.

In our control structure, sensor signals can influence the control at different levels, using four interfaces, *a*, *b*, *c* and *d* (Figure 1). Interfaces *a*, *c* and *d* are available in most current robot systems, while interface *b* is a novelty of our control structure.[1] Interface *a* corresponds to instructions of the user program, for example “stop the movement” if the robot approaches a dangerous area. Interface *d* inserts sensor signals just before the controller, at the same rate. It is for example used by the teach pendant.

Reactions to changes of the target position or to deviations from the desired path are generally performed by adding a correction signal after the interpolation, using interface *c*. If this strategy can be applied successfully for small perturbations, it will lead to instability when the changes are too large. It has been proposed[2] to use filtering to solve this problem. However, in order to deal with important perturbations of the target position, like tracking a fast moving object observed by a vision system (which has a large dead time), it is necessary to replan the movement on-line.

Interface *b* offers functions not usually available, and by which the robot can react almost immediately to sudden changes in the environment. Examples of these functions are:

(i) Modify the motion speed during the movement
(ii) Stop and continue the movement
(iii) Stop and reverse the movement direction
(iv) Set a new target position.

Regarding function iv, several planners have been proposed,

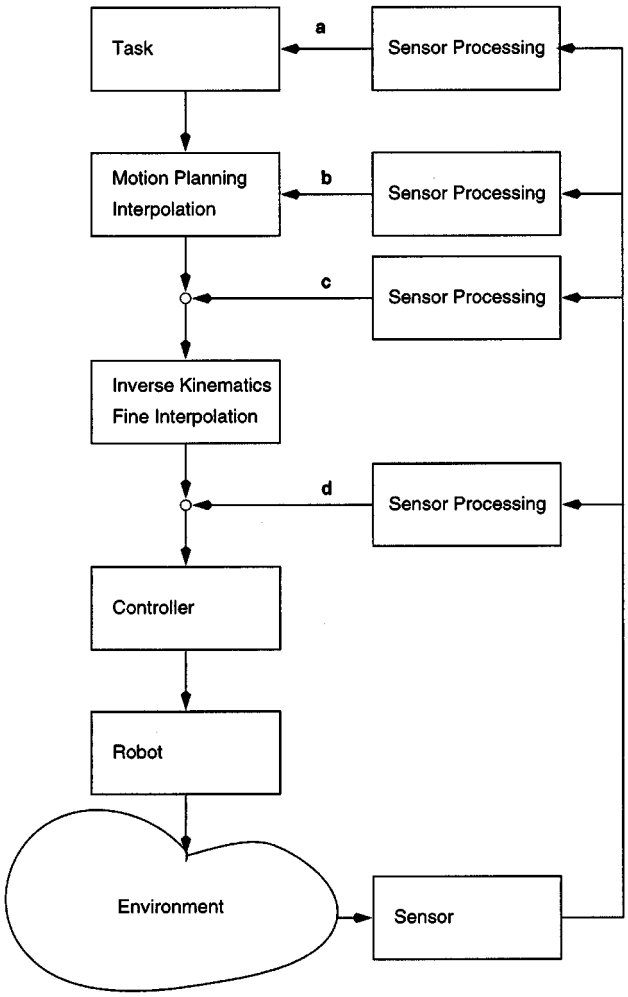


Fig. 1. Dataflow of a robot control.

* Institute of Robotics, ETH-Zurich (Switzerland)
† School of Kinesiology, Simon Fraser University, B.C. (Canada)
Email: luthiger@isoe.ch, e.burdet@ieee.org
WWW: www.isoe.ch, www.sfu.ca/~eburdet/

https://doi.org/10.1017/S0263574799000727 Published online by Cambridge University Press

in which the plan is parameterized by a polynomial whose parameters can be slightly modified to account for a new target position.[3–5] We implemented a more radical algorithm: with the *Flexible Motion Planner* (FMP), the movement is completely replanned when it is necessary, allowing the FMP to react smoothly to arbitrary large changes of the target position.

1.3. Modular motion design

If the mechanical concept of modular robots already exists,[6–10] modular software, i.e. robot independent software, has yet to be designed. The modularity of our motion planner, as well as the modularity of the whole control structure,[1] is an effort in this direction.

Modularity of the motion design means that the user can combine, add and remove motion blocks arbitrarily, that he can modify the velocity in part of the motion, etc. This allows him to create a library of simple movements which can be combined arbitrarily, and to use parts of existing movements for new movements. A graphical motion editor has been realized on the basis of our movement description.[11]

The modularity has been supported by realizing the motion planner in an object-oriented way. Object Oriented Programming (OOP) provides a type *class* combining data and functions in the same elements.[12] *Class* is an abstract data type, while *object* is an instance of *class*. The functions associated to an object are called *methods*. *Subclasses*, which inherit methods of their class, extend them by adding some data types and methods. *Extensibility* and *inheritance* are the main properties of OOP. In order to visualize the relations between the different classes we use in this paper a notation tool called *Collaboration Graph*,[13] whose definition is summarized in Figure 2.

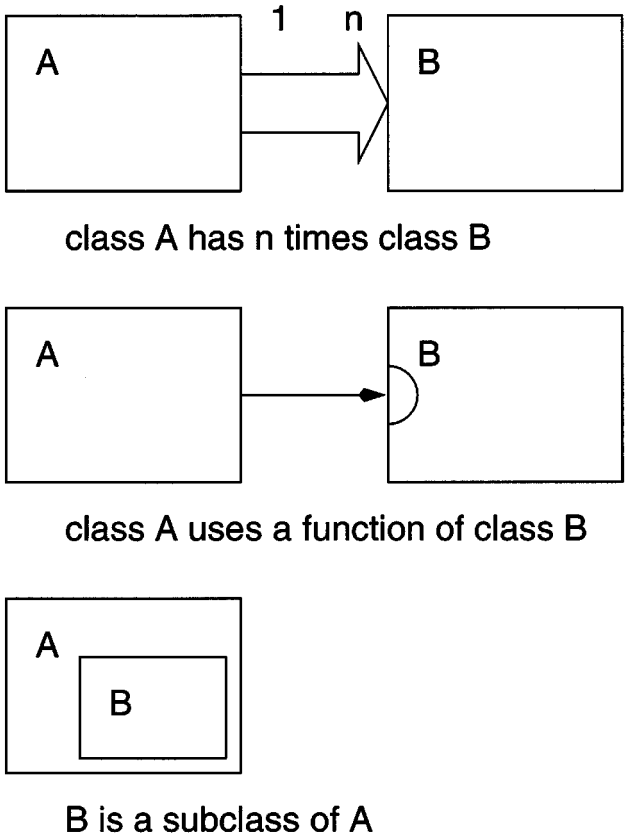


Fig. 2. Definition of a *Collaboration Graph*, a tool for visualizing OO structures.

1.4. Different kinds of motions

The motion planner provides two kinds of motions (Figure 3):

PathMotion/PTPMotion. These notions are bound to a geometric path described in configuration space/joint space. At least some of these movements are available in industrial robots. New features are our modular motion description, its extensibility made possible by the OOP, and the fact that functions 1, 2 and 3 of subsection 1.2 are available. These motions are described in section 2. The velocity profile is represented by a *kinematic module* described in section 3.

FlexPathMotion/FlexPTPMotion. These motions, described in section 4, are not bound to a given path, but realize a fast movement to a desired target or velocity condition. The desired target features can be modified on the way, i.e. in addition to the functions 1, 2 and 3 from subsection 1.2, function 4 is also available. *FlexPathMotion* is described in the configuration space and *FlexPTPMotion* in the joint space.

1.5. Practical realization

The motion planner is written in Oberon–2,[14] the object-oriented grandson of *Pascal*, also developed by the group of N. Wirth, at ETH-Zurich.* It has already been used for several different robots. It was, in particular, used for controlling a manipulator guided by a vision system[15] and a robot with various sensors used for feeding a bending machine with metallic plates.[16] Several practical aspects of these two implementations are discussed in section 5.

2. MOTIONS BOUND TO A PATH: THE *MOTION* CLASS

We start this section with the motion description of the movements bound to a geometric path.

Path. Class *Move* describes the geometry of a path segment. Subclasses of *Move* are implementations like *Line*, *Circle* or *Spline*. The analytical expressions for these implementations are standard and can be found in reference 17. Class *Location* corresponds to a pose.† A path is constituted of setpoint objects from *Location* and objects from *Move*. *Move* and *Location* are subclasses of *PathEl* which transforms *Path* into a simple, sequential list of *PathEl* objects (Figure 3).

Velocity Profile. The classes corresponding to the velocity profile are *BasicMotion* and *KinModule*. At each step,

* It also is the "father" of *Java*, in the sense that it already contains the main features of *Java*.

† A pose is the generalisation of a position. It is constituted of a position in the Cartesian space and an orientation. The orientation can be specified by a rotation matrix (SO3), by Euler- or Kardan-angles, or by quaternions or a rotation vector (SU2).

https://doi.org/10.1017/S0263574799000727 Published online by Cambridge University Press

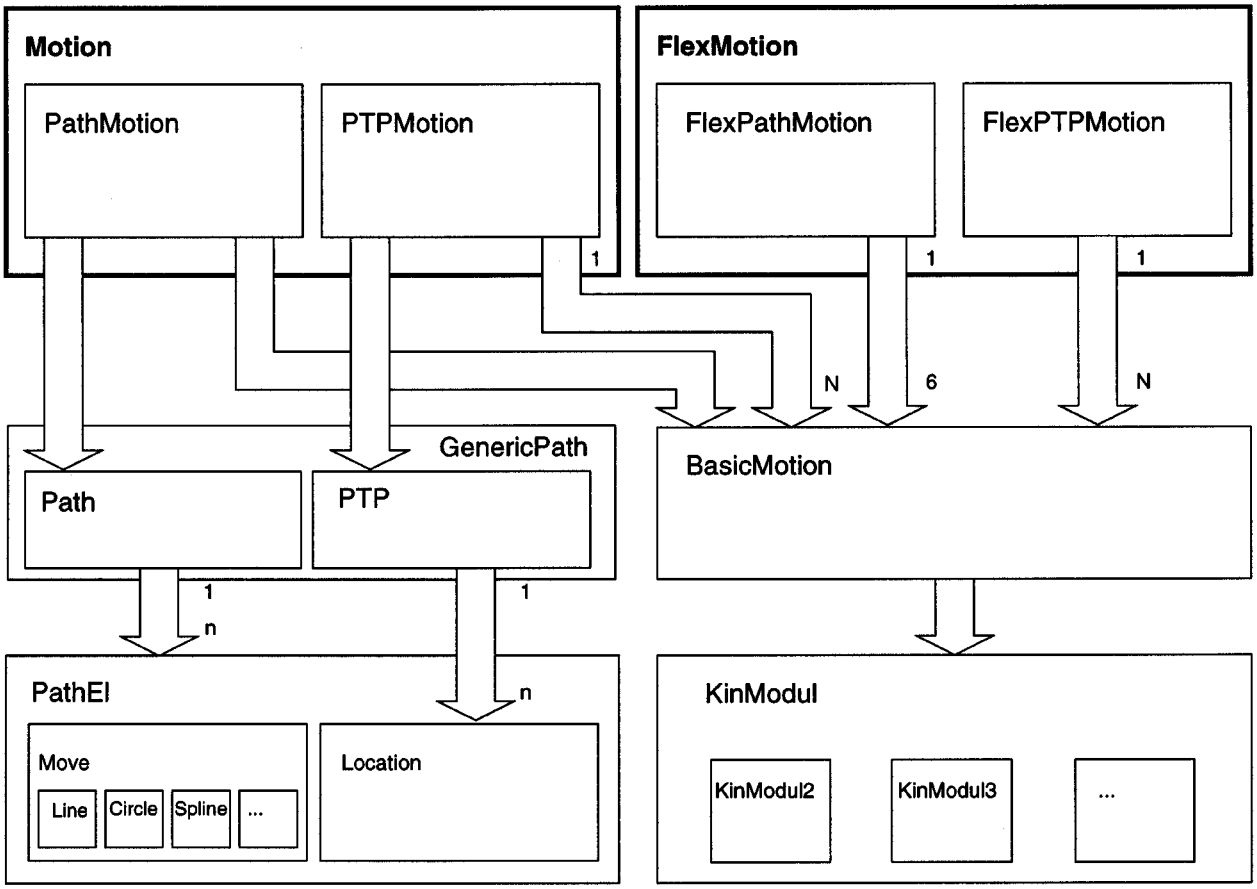


Fig. 3. Object-oriented description of the motion planning: *Motion* and *FlexMotion* classes.

KinModule computes the desired state for the next time increment. It considers boundary conditions for the pose and the velocity, and also adjusts the velocity to its nominal value. A braking phase is automatically started in order to reach the desired final pose.

BasicMotion adds to *KinModule* the control of different functions: overshooting of the goal, stop and continue, and the reversing of the motion direction. As *BasicMotion* is based on *KinModule*, it is a one-dimensional *incremental path planner*: Using the sensor inputs, it calculates at each step the desired values only for the next time increment.

Description of the Overall Movement. All the above elements are used for constructing *PathMotion* and *PTPMotion* (Figure 3). They are combined in the generic class *Motion*.

PTPMotion has a *BasicMotion* for each joint whereas *PathMotion* needs just one velocity profile along the path, i.e. just one *BasicMotion* object. Class *PTP* is similar to class *Path*, but contains only *Location* objects. Both are subclasses of the class *GenericPath*, which provides the interface to *Motion*.

Editing Movements. The editing of movements can be made by combining different objects from *Move*, *Location* and *KinModule* using the main methods of *Motion* like *Append*, *Split*, *Head* or *Tail* (Figure 4). Due to the extensibility provided by OOP, new objects from *Move* and *KinModule* can be added without requiring to modify the higher movement commands at all.

3. VELOCITY PROFILE: THE *BASICMOTION* CLASS

3.1. Different kinematic modules

We implemented two versions of kinematic modules.* The 2nd order kinematic module calculates the fastest velocity profile with bounded velocity and acceleration. The shape of this velocity profile generally corresponds to the trapezoidal velocity profile used by most industrial manipulators (Figure 5). Three phases have to be considered for programming this kinematic module: the *acceleration phase*, the phase of constant velocity or *slew phase*, and the

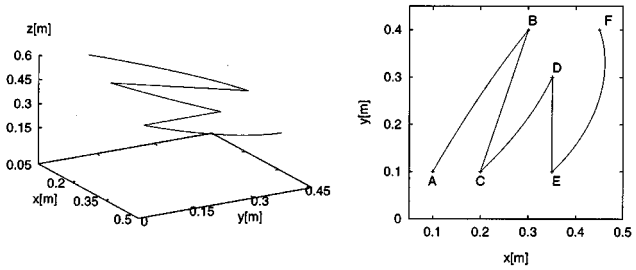


Fig. 4. Movement consisting in the combination of 3 *PTPMotion* and 2 *PathMotion*.

* Note that thanks to OOP, switching from one kinematic module to the other, or adding a new one, does not require modification of the main program.

https://doi.org/10.1017/S0263574799000727 Published online by Cambridge University Press

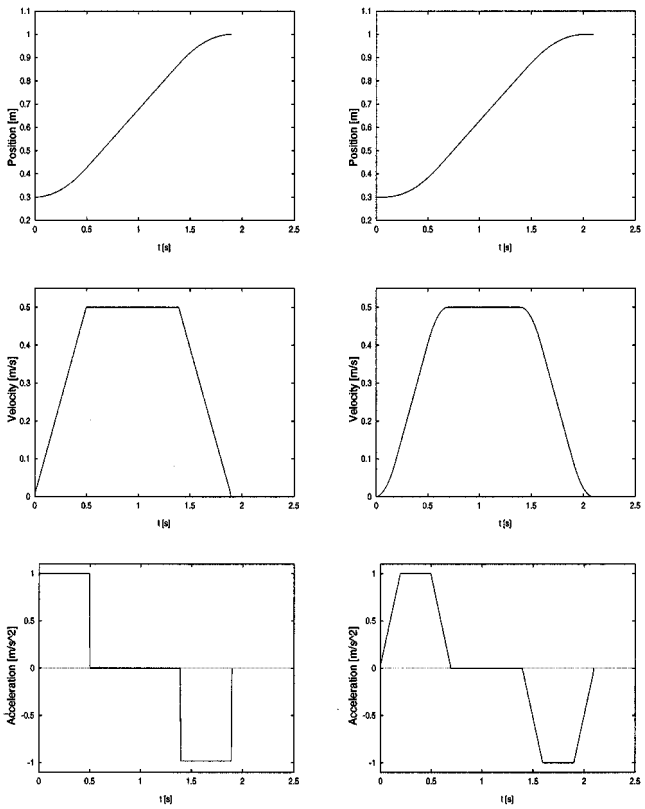


Fig. 5. Comparison between the velocity profile performed with the 2nd order (left) and with the 3rd order kinematic modules (right). Note the abrupt transitions of the acceleration for the 2nd order kinematic module, leading to infinite jerk.

deceleration phase. Our implementation is similar to the module described in reference 18.

In order to prevent sudden changes of the acceleration resulting in vibrations, in the 3rd order kinematic module not only the velocity and the acceleration, but also the jerk* are bounded (Figure 5). In contrast to the second order kinematic module, the profile computed by the third order module has seven different movement parts, which makes it complex.

3.1.1. Algorithm of *KinModule3*. For a one-dimensional movement with jerk j bounded by $j_m > 0$: $-j_m \leqslant j \leqslant j_m$, the time optimal trajectory corresponds to a bang-bang control of the jerk:[19]

$$j(k) \equiv \begin{matrix} j_m \\ 0 \\ -j_m \end{matrix}, \tag{1}$$

$$a(k+1)=a(k)+j(k)\,\Delta t, \tag{2}$$

$$v(k+1)=v(k)+a(k)+\frac{1}{2}(\Delta t)^2,$$

$$l(k+1)=l(k)+v(k)\,\Delta t+\frac{1}{2}a(k)(\Delta t)^2+\frac{1}{6}j(k)(\Delta t)^3.$$

* The *jerk* is defined as the time derivative of the acceleration.

In these equations, k corresponds to the discrete time $k\Delta t$, where Δt is the sampling interval. l is the length parameter, v its time derivative, i.e. corresponds to the norm of the velocity vector.[20] a is the second time derivative of the length parameter, i.e. corresponds to the norm of the tangential acceleration.[20]

At each time step $k\,\Delta t$, the velocity and acceleration bounds also have to be considered:

$$-j_m \leqslant j(k) \leqslant j_m \tag{3}$$

and

$$-a_m \leqslant a(k) \leqslant a_m. \tag{4}$$

Together with the required end conditions, this makes the planning a complex task (Figure 6). As for the second order kinematic module, the planning consists of three main phases: the acceleration phase, the slew phase, and the deceleration phase. A fourth *braking phase* is used in order to correct for execution errors in the final deceleration. As the acceleration is a trapezoid, all these phases except the phase of constant velocity contain 3 subphases corresponding to the three possible values of the jerk. The main problem consists in switching in each time step from one (sub)phase to the next one.

The condition for switching from the acceleration phase to the slew phase is

$$v(k) < v_m? \tag{5}$$

while the condition for switching from this slew phase to the deceleration phase is

$$v(k) > v_m? \tag{6}$$

In the acceleration phase, it must be switched from the subphase of positive jerk to the subphase of 0 jerk when

$$a < a_m, \tag{7}$$

while in the deceleration phase this switching occurs when

$$a > -a_m. \tag{8}$$

In the acceleration phase, it must be switched to negative jerk subphase in order to ensure that the maximal velocity will be smoothly reached, i.e. when

$$v(k) \geqslant v_m - \frac{a^2(k)}{2j_m}. \tag{9}$$

Similarly, in the deceleration phases, it must be switched to positive jerk when

$$v(k) \leqslant v_e + \frac{a^2(k)}{2j_m}, \tag{10}$$

where v_e is the desired final velocity.

3.1.2. Calculation of the deceleration phase. The calculation of the decelaration phase requires the examination of two different cases. In the case that

$$a^* > -a_m, \tag{11}$$

i.e. that the minima of the acceleration a^* is higher than the

https://doi.org/10.1017/S0263574799000727 Published online by Cambridge University Press

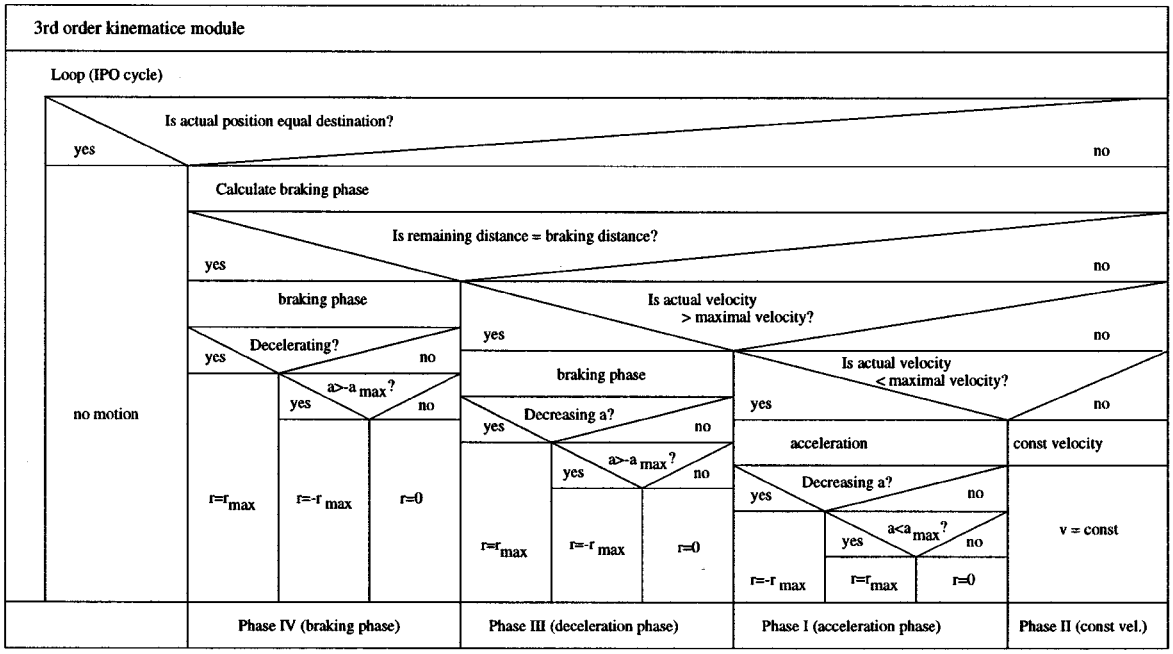


Fig. 6. Program structure of the 3rd order kinematic module.

lower bound for the acceleration (Figure 7), then the time necessary to realize the braking would be

$$t_{dec} \equiv t_e - t = \frac{a(k)}{j_m} + \left(2\frac{a(k)^2}{j_m^2} + \frac{4}{j_m}(v(k) - v_e) \right)^{1/2}. \quad (12)$$

In this case a^* and t^* are

$$a^* = a(k) - j_m(t^* - t), \quad (13)$$

$$t^* - t = \frac{a(k)}{2j_m} + \frac{t_{dec}}{2}. \quad (14)$$

Using a^* and t^*, the distance d_{dec} necessary to decelerate to the desired end conditions can then be calculated using equation (2).

In the case that the acceleration bound is reached:

$$a^* = -a_m, \quad (15)$$

then the times for the different subphases described in Figure 7 can be calculated as

$$t^* - t = \frac{a(k) + a_m}{j_m}, \quad (16)$$

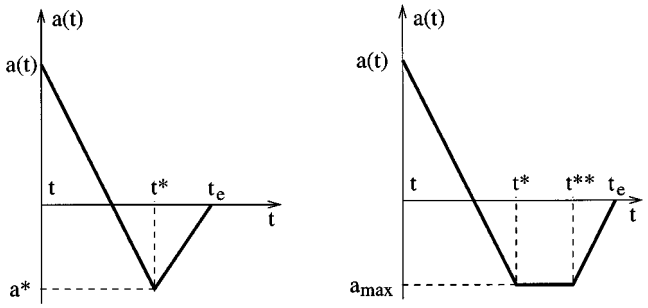


Fig. 7. Different cases for the calculation of the deceleration phase of *KinModul3*. Left: when $a^* > -a_m$, right: case $a^* = -a_m$.

$$t^{**} - t = t_e - t - \frac{a_m}{j_m},$$

$$t_e - t = \frac{v(k) - v_e}{a_m} + \frac{(a(k) + a_m)^2}{2j_m a_m} + \frac{a_m}{j_m}.$$

In order to reach the desired position l_e and velocity v_e exactly, the jerk is modified from j_m to

$$j^* = 2(v(k) - v_e)\left(\frac{v(k) + 2v_e}{3(l_e - l(k))} \right)^2 \quad (17)$$

and the acceleration to

$$a^* = -3j^*\left(\frac{l_e - l(k)}{v(k) + 2v_e} \right). \quad (18)$$

This corresponds to the *braking phase* of Figure 6.

3.2. From KinModule to BasicMotion

While *KinModule* is controlling only the relative length parameter l, *BasicMotion* contains an offset l_o and therefore controls the absolute length parameter l_a. It also contains a parameter $\alpha = \pm 1$ for the movement direction:

$$l_a(k) = l_o + \alpha l(k). \quad (19)$$

By this, *BasicMotion* handles the functions *stop*, *continue* and *reverse direction*.

If the remaining distance to the desired position is not sufficient to brake, i.e. when

$$l_b \geq l_e - l(k), \quad (20)$$

then an overshooting is realized by the operations

https://doi.org/10.1017/S0263574799000727 Published online by Cambridge University Press

$$l_a \equiv l(k), \quad (21)$$
$$l(k) \equiv 0,$$
$$l_{e,new} \equiv |l(k) - l_e|,$$
$$\alpha_{new} \equiv \alpha.$$

4. THE FLEXIBLE MOTION PLANNER (FMP)

The flexibility of *PTPMotion* and *PathMotion* is restricted to on-line velocity and direction changes. Sensor-oriented applications, e.g. vision based robot control, require one to reset a new target position fast and frequently during the motion. We designed in this aim the *Flexible Motion Planner* (*FMP*), which reaches the target smoothly and stably even when the target position or the velocity is changed at each cycle (Figure 8).

Actually we have seen in the previous sections that this property is already fulfilled by *Basic Motion* . . . in one dimension only. The idea of the FMP is to use a *BasicMotion* object in every dimension and to coordinate these objects. The structure of the FMP is represented in Figure 3. Built on the functionality of *BasicMotion*, the FMP inherits from it all four functions of section 1.2.

4.1. FlexPTPMotion

The FMP is principally implemented in joint space, as *FlexPTPMotion*. In this case one *Basic-Motion* is taken for each axis. When *KinModul*2 is used as kinematic module, the coordination of the different axes is performed by the following steps:

(i) Using the bounds $v_{m,i}$ and $a_{m,i}$ for the velocity and the acceleration, determine in each dimension i the times $T_{a,i}$, $T_{s,i}$ and $T_{d,i}$ for respectively the acceleration, slew and deceleration phases.

(ii) Select for each phase the maximal time over all the dimensions. Slow the other dimensions in order that the duration of each phase is the same for all dimensions. New upper bounds $v'_{m,i}$ are then calculated for all the phases, and then the corresponding $a'_{m,i}$ are determined.

(iii) At this moment we have a coordinated movement, but the overall velocity or acceleration may overshoot the bounds. The velocity is then lowered in each dimension in order to prevent this overshooting:

$$v''_{m,i} \equiv v'_{m,i} \frac{v_m}{(\Sigma_{i=1}^{n}(v'_{m,i})^2)^{1/2}}. \quad (22)$$

These $v''_{m,i}$ determine new acceleration bounds $a''_{m,i}$. These new acceleration bounds may still be too large. They are therefore scaled by

$$a'''_{m,i} \equiv a''_{m,i} \frac{a_m}{(\Sigma_{i=1}^{n}(a''_{m,i})^2)^{1/2}}. \quad (23)$$

(iv) All the dimensions of the movement are now coordinated, and the new bounds $v_{m,i} \equiv v''_{m,i}$ and $a_{m,i} \equiv a'''_{m,i}$ can be used in order to realize the movement.

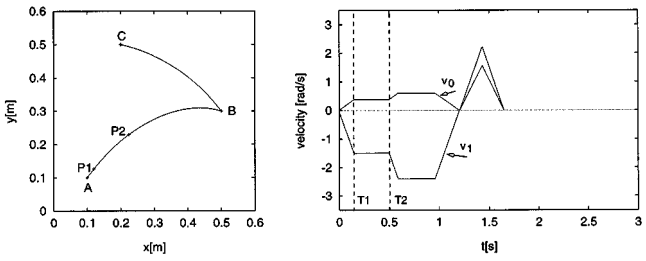


Fig. 8. Modification of the velocity during a *PTPMotion* made possible by the use of *Basic-Motion* as velocity profile. The movement starts at position A, and the velocity bound is 3 m/s. Two velocity changes are applied while driving towards B. At t=T1 the speed is reduced to the half, and at t=T2 3/4 of the velocity are allowed. During the movement from B to C this limit cannot be reached by any joint, because the movement is too short.

4.2. FlexPathMotion

Using some exemplaries of *FlexPTPMotion*, it is possible to design Cartesian versions of the FMP corresponding to particular architectures. The advantage of a Cartesian version is that obstacles present in the workspace can more easily be accounted for than with the version in joint space (Figure 9).

We have currently implemented a version of *FlexPathMotion* corresponding to a SCARA architecture. Two independent *FlexPTPMotions* are used: one for the position of the end-effector in the 3-Dimensional space, and one for the orientation. Making the orientation independent from the position is useful for certain applications where only the target position or orientation varies during the movement. In this case the changes in orientations (respectively in positions) do not affect the positioning (respectively the orienting).

5. APPLICATIONS

5.1. From theory to reality

Because of the modularity of our motion planner, most of the functionality is already contained in the class *KinModule*. Special care had therefore to be taken for the numerical implementation of *KinModule*. For very short movements (within two time periods of the kinematic module), upper bounds of the velocity and acceleration calculated as in subsection 4.1 diverged and had thus to be replaced by constants.

5.2. Sheet-bending robot

The sheet-bending manufacturing cell developed consists of a laser cutting machine, conveyor belts, a press brake and a robot arm (Figure 10).

Corresponding to the modular robot philosophy, a mechanical structure corresponding to the sheet-bending task was selected.[16] The resulting 6 DOF manipulator has 2 prismatic and 4 revolute joints. An intelligent gripper equipped with several sensors was used in order to grasp and manipulate the metal plates. The robot is controlled by a VME-bus with a 68040 processor and several I/O boards.

https://doi.org/10.1017/S0263574799000727 Published online by Cambridge University Press

Both the sensor and motion processes run at a sampling rate of 30 ms (33.3 Hz), and the controller at 5 ms (200 Hz). Bringing the plate to the machine was performed in three phases. The gross motion was controlled by a *PTPMotion*, a *PathMotion* was used for the positioning and the final approach phase was controlled by the gripper sensors.[16] Due to the low speed necessarily used during the fine tuning phase, this phase could be controlled by a sensor control loop using interface *d* of our control structure.

Programming the motion was performed "on-line" in an interactive way, and could therefore be realized relatively fast. The user did not need to specify the velocity in advance, but could adjust it during test movements. He could stop the movement, go back and forth, and modify the velocity whenever necessary. None of the application program needed to be rewritten during this operation. Designing the motion planning interactively also increased the safety, which was necessary, because the manipulator can produce a force up to 2250 N.

5.3. Visually guided robot

The robot we used consisted of a vision system and a parallel manipulator moving in a horizontal workspace. The manipulator has 3 degrees-of-freedom, the x and y positions of the end effector and its orientation in the $x-y$ plane. The robot is controlled by three VME boards: a 68040 was used for controlling the manipulator, and the vision system consisted of a frame grabber and a 68020 for vision processing. This robot and the experiments are described in reference 15.

The robot had to track a light bulb fixed on the wagon of a model train and drop small objects into it. With the primitive vision system used, the time interval between two images and the delay for finding the wagon in the image were both 80 ms. As the wagon was moving with a velocity up to 1 m/s, and its trajectory was unpredictable for the robot (because the rails were not perceived), the wagon position could be shifted by 10 cm between two consecutive images. Tracking the wagon therefore required replanning

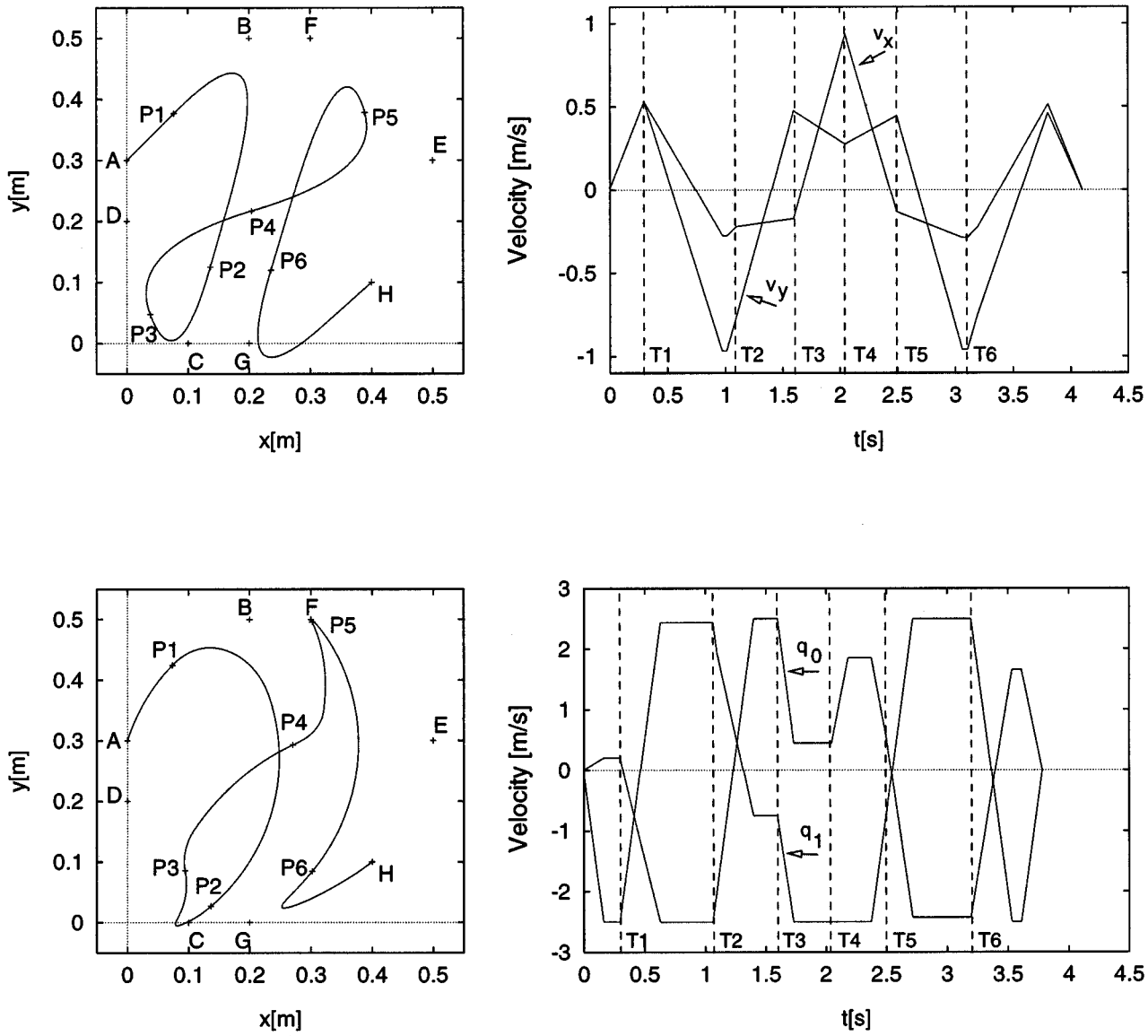


Fig. 9. Reaction to changes of the target position with a *FlexPTPMotion* (above) and a *FlexPathMotion* (below). These figures show a movement initially planned from *A* and *B*, but where the target was reset 6 times, at (t=t1, P=P1:B), . . . , (t=T6, P=P6:H). Note that in contrast to the joint space version *FlexPTPMotion*, the Cartesian version *FlexPathMotion* allows the user to control the workspace.

https://doi.org/10.1017/S0263574799000727 Published online by Cambridge University Press

of the movement after each image. The movement was planned as a *FlexPathMotion*. The sampling time of the controller was 3 ms and the replanning was performed about every 15 ms.[21]

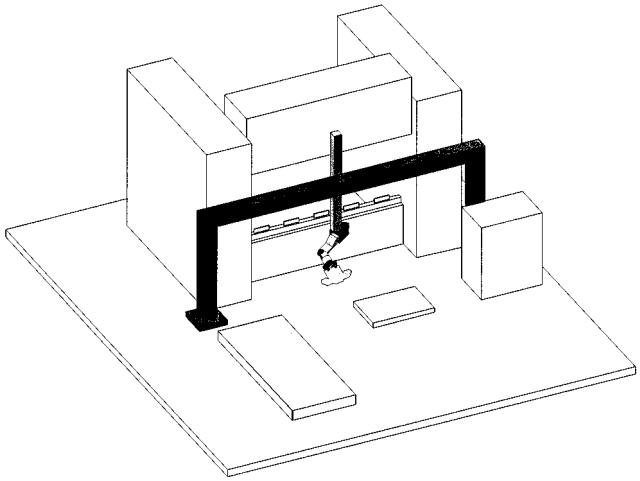


Fig. 10. Drawing of the robot feeding a sheet bending machine, which movements were controlled using the motion planner described in this paper.

Figure 11 shows trajectories performed with different upper bounds for the velocity. The video[22] shows how this velocity bound and other parameters were learned during specified movements. After the learning, the robot could drop metallic objects easily. Thus using simpler hardware than in references 3 or 4, similar or better results were obtained with our robot.* This success is probably due in a large part to the efficiency of the *FlexPathMotion*.

6. CONCLUSIONS

This paper presents a new motion planner, which has been tested in numerous simulations and implemented in two different robots controlled by various sensors. Examples from these implementations illustrate the advantages of the motion planner:

- it allows a fast and stable use of sensor information;

* Of course, such a comparison is not really objective, because different hardware were used and different tasks had to be executed. For example, our task may demand more precision than simply catching an object.[4] On the other hand, in reference 3, a more complex vision processing had to be used.

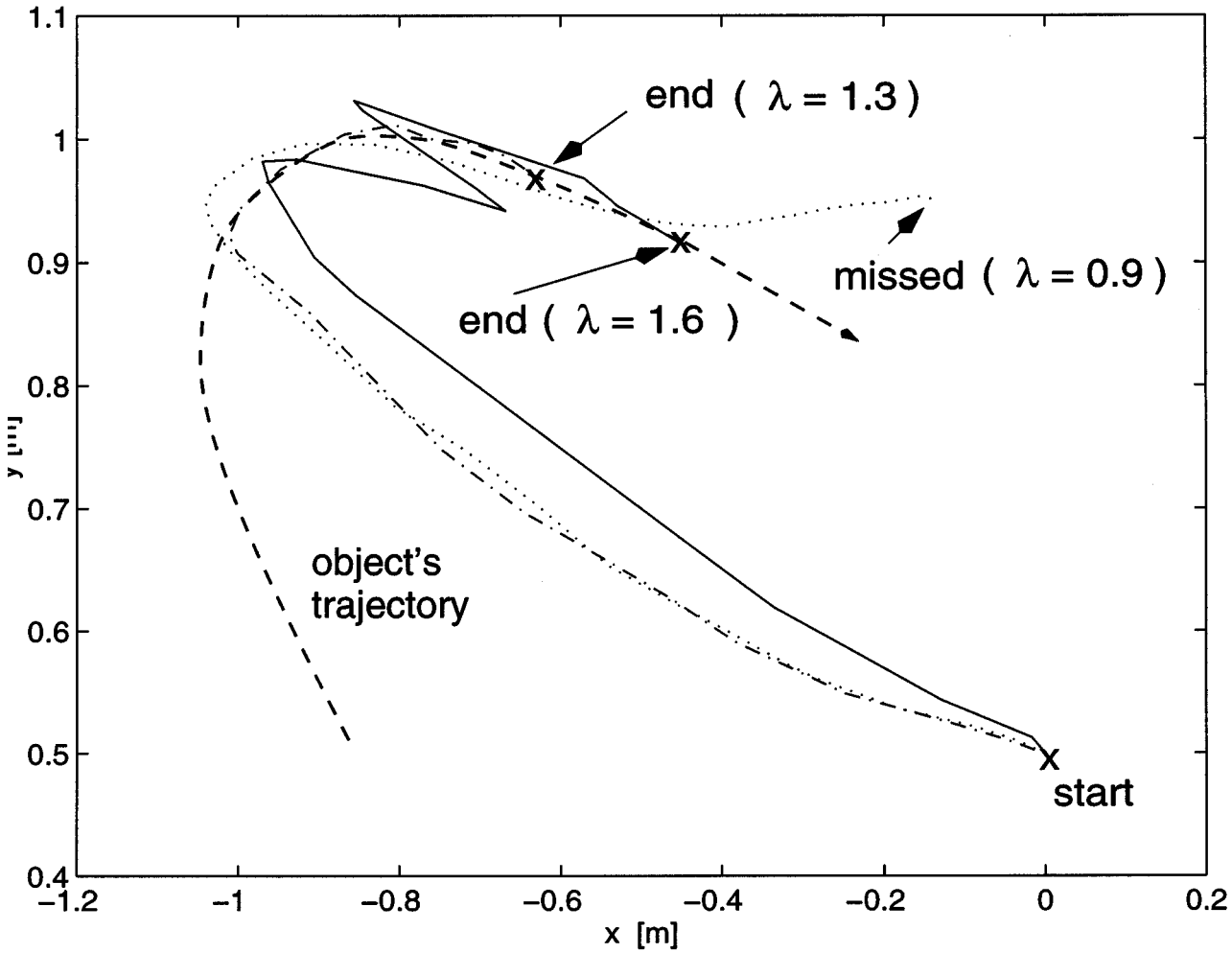


Fig. 11. Trajectories realized with a visually guided robot, which had to track a moving object. A *FlexPathMotion* was used as motion planner. This Figure shows motions performed with different velocities ($\lambda = 0.9$: too low velocity, $\lambda = 1.6$: too high velocity bound (produces vibrations), $\lambda = 1.3$: optimal value learned by unsupervised stochastic learning.

https://doi.org/10.1017/S0263574799000727 Published online by Cambridge University Press

- its modularity permits the user to design the motions by combining motion primitives and modifying the plan interactively;
- the modularity and good sensor integration made possible to introduce a Flexible Motion Planner (FMP), which is able to react fast to arbitrary changes of the target position.

The good motion behaviour of both of these robots[15,16] show the efficiency of our motion planner.

ACKNOWLEDGEMENTS

The experiments reported in this paper have been performed in the Institute of Robotics, ETH-Zurich. We would like to thank all the students and colleagues who contributed to this project.

References

1. J. Luthiger and E. Burdet "An object-oriented control structure for flexible robotics" (Submitted to Journal of Computer Integrated Manufacturing and Automation, 1998).
2. G. Pritschow and A. Horn "Dynamik derzeitiger Sensorregelkreise für Industrieroboter" *Robotersysteme*, **7**, 178–184 (1991).
3. A. Timcenko and P. K. Allen "Modeling dynamic uncertainty in robot motions" *IEEE International Conference on Robotics and Automation*, *pp*. 531–536 (1993), **Volume 3**.
4. G.C. Buttazo, B. Allotta and P. Fanizza "Mousebuster: A robot for real-time catching" *IEEE Robotics and Automation Magazine* 49–55 (February 1994).
5. W. Wong and J. J.-E. Slotine "Experiments in hand-eye coordination using active vision" 4*th International Symposium on Experimental Robotics* (1995), pp. 130–139.
6. T. Wurst "The conception and construction of a modular robot system" *16th International Symposium on Industrial Robotics* (1986), pp. 37–44.
7. T. Yabuta and T. Yamada "Possibility of neural networks controller for robot manipulators" *IEEE International Conference on Robotics and Automation* (1990), pp. 1686–1691.
8. E. Nielsen and R. Hueppi Modro "A modular concept for an application adapted robotic system" *23rd International Symposium on Industrial Robots* (1992), pp. 495–500.
9. I.-M. Chen "Theory and Applications of Modular Reconfigurable Robotic Systems" *PhD thesis* (California Institute of Technology, 1994).
10. C. J. J. Paredis and P. K. Khosla "Agent-based design of fault tolerant manipulators for satellite docking" *IEEE International Conference on Robotics and Automation* (1997), pp. 3473–3480.
11. R. Roshardt "Design und Implementation einer offenen, modularen Robotersteuerung in Oberon" *PhD thesis* (ETH-Zuerich, 1997, no. 12296).
12. H. Moessenboeck *Object-Oriented Programming in Oberon*-2 (Springer, Berlin, 1995).
13. R. Wirfs, R. Wilkerson and L. Wiener, *Designing Object-oriented Software* (Addison-Wesley, Reading, Mass., 1990).
14. N. Wirth and J. Gutknecht *Project Oberon*: *The Design of an Operating System and Compiler* (Addison-Wesley, Reading, Mass., 1992).
15. E. Burdet and J. Luthiger "Learning the coordination of robot movements with vision processes" (*subm*. *to IEEE Transactions on Robotics and Automation*, 1997).
16. Ch. Uhrhan "Ein interaktives Robotersystem zur flexiblen Biegeteilfertigung" *PhD thesis* (ETH-Zürich, 1997, No. 12086).
17. J. Luthiger "Sensorintegration und sensororientierte Bahnplanung: Elemente einer modernen, modularen Robotsteuerung" *PhD thesis* (ETH Zürich, 1996, No. 11659).
18. J. Olomski *Bahnplanung und Bahnfuehrung von Industrierobotern* (Vieweg Braunscheig, 1989).
19. H. P. Geering et al. "Time optimal motions of robots in assembly tasks" *IEEE Transactions on Automatic Control*, **AC-31** 512–518 (1986).
20. L. Sciavicco and B. Siciliano *Modeling and Control of Robot Manipulators* (McGraw-Hill, 1996).
21. J. Mueller "Optimierung der visuo-motorischen Koordination" *Technical Report* (Institut fuer Robotik, ETH-Zrich, 1995).
22. E. Burdet and J. Mueller, A robot learning reaching motions (video). *IEEE International Conference on Robotics and Automation* (1996).

https://doi.org/10.1017/S0263574799000727 Published online by Cambridge University Press