On the Computer Control of Movement*

R. W. Brockett
Division of Applied Sciences
Harvard University
Cambridge, MA 02138

Abstract

In this paper we describe the theoretical background for an ongoing experiment in motion control. The work has as its goal the discovery of efficient ways to distribute the computation and organize the communication needed to manage systems involving many degrees of freedom and unpredictable environments. Formalizing motion control to the extent necessary to be able to investigate correctness of motion control programs is an important subgoal. A conceptual framework is presented together with some remarks on an existing implementation which uses these ideas.

Introduction

A common way to modularize the development of computer controlled devices is to focus on three problems:

a) The definition of an interpretive language capable of describing the tasks to be done.

b) The construction of an interpreter/mechanism which accepts such descriptions and executes the task.

c) The creation of suitable applications programs for facilitating the generation of the description of specific tasks in terms of the interpretive language.

Frequently the interpretive language is designed with a degree of device independence so that the work invested in its development can be amortized over many products. For example, various commercially available pen plotters and laser printers fit this pattern and a number of motion control products are also of this type.

The main problem in motion control is to simultaneously achieve adequate speed and expressiveness. In this paper we discuss a solution to these problems based on the type of modularization scheme just outlined. More specifically, we define the essential parts of a general purpose motion description language and show that with a modest investment in computational hardware for the interpreter, a language based on these primitives can give adequate performance. The main scientific issue to be addressed is that of how to choose the operators which are to be included in the language and, hence, interpreted by the interpreter/mechanism. We define just three, although two of these accept a number of parameters. They are: *i)* an operator for specifying a feedback dependent, vector-valued velocity for the mechanism together with a time interval over which the velocity rule is to be used; *ii)* an operator which specifies a change of coordinates thus altering the meaning of subsequent input data; and *iii)* a reporting function. From the point of view of robotics, perhaps the most innovative features are the introduction, through *i)*, of programmable compliance at the lowest level and the use of a combination of table lookup and matrix multiplication, using the idea of exponentially separable functions, to implement the sweeping generalizations of the inverse kinematics problem embodied in *ii)*. Analysis is given supporting the expressiveness and computational tractability of a solution based on these choices.

In addition to being an abstraction of certain parts of engineering/computer science practice, our approach has been influenced by the psychology literature. In particular Bernstein's work on the degrees of freedom problem [1] and Hinton's work [2] on the computational tractability of the inverse kinematics problem are provocative as are the more specialized theories of Marr [3] and Albus [4]. In this context, one of the attractive features of the framework discussed here is the clear separation between low level feedback rules (part b) and higher level planning (part c).

Portions of the setup described here have been implemented at the Harvard Robotics Laboratory and some remarks about this are given in later sections. Alex Bangs, Victor Eng and Josip Lončarić, as well as others in the lab, have contributed to this implementation and made suggestions about the theory.

Open- and Closed-Loop Control

We wish to achieve a degree of device independence in our treatment of motion control. Nonetheless, because the amplifiers, actuators, and mechanisms which will ultimately realize the motions are governed by the laws of physics, adopting a differential equation model for them is reasonable. Our notation for the description of this "lowest level" hardware is

$$\dot{x}(t) = f(x(t)) + G(x(t))v(t) \quad ; \quad y(t) = h(x(t))$$

*This work was supported in part by the U.S. Army Research Office grant number DAAL03-86-K-0171 and the National Science Foundation grant number CDR-85-00108.

with x being an n-dimensional state vector, v being a m-dimensional vector of controls and y being a p-dimensional vector consisting of the sensor outputs.

In the process of arriving at such a model various approximations will be necessary. Even so, typically x will include components for the position and velocity for each mechanical degree of freedom, components representing dynamical effects in the actuators and amplifiers, etc. The vector y will include components for the outputs of position sensors for all the mechanical degrees of freedom, force sensors for at least some mechanical degrees of freedom, and possibly components for the readings of amplifier currents, vision sensor outputs, etc. The components of v will be the inputs to the various amplifiers associated with the actuators.

The function of a motion control system is to use the sensory data represented by y, together with instructions supplied exogenously, to compute v. Engineers commonly divide control actions into two categories, *open-loop control* and *closed-loop control.* The open-loop situation corresponds to setting v equal to some fixed function of time, ignoring y altogether. A prototype for open-loop control is launching a projectile with no post-launch corrective mechanisms. On the other hand if there is some sensing mechanism whose output can be used to adjust the input to the system based on its evolving performance, then one speaks of closed-loop control. The reason that closed-loop control is absolutely essential in many settings is that it enables the system to adjust its response to a partially unknown or changing environment. In the present context the equations of motion under the application of an open-loop control $u(\,)$ are just

$$\dot{x}(t) = f(x(t)) + G(x(t))u(t)$$

Closed-loop control, using the feedback rule $v(t) = k(y(t))$ results in the differential equation

$$\dot{x}(t) = f(x(t)) + G(x(t))k(h(x(t)))$$

Although this terminology is well established, its usage invites two types of difficulties. The first is that it does not seem to correspond to everyday experience with human motor control. One can easily appreciate that many human motions appear to be a combination of a pre-positioning phase achieved with little or no feedback, followed by an environmental adjustment phase characterized by a significant use of feedback. Walking, grasping and shaking hands are examples of what we mean. Thus it seems that in dealing with complex systems it is more insightful to think of the mode of control as alternating between these two possibilities. The second point has to do with basic mathematics. As defined above, open-loop control is not an alternative to closed-loop control but simply the special case for which $k(y)$ is independent of y. Commonly, one appeals to additional conditions so as to be able to make a clear distinction between open- and closed-loop control. One possibility is to designate a neutral or "home" value for y. If this value is y_0, then it is possible to split any $k(y(t))$ up as the sum of two terms, one of which is independent of y and the other of which vanishes when y is in the home position. That is,

$$k(y(t)) = k(y_0) + (k(y(t)) - k(y_0))$$

In this equation the two terms on the right can, without ambiguity, be called the open- and closed-loop parts, respectively.

For a motion control situation in which the components of y represent positional data and force data sensed from the environment, y_0 would correspond to the home position of the mechanism and a zero value for the sensed environmental forces.

MDL Devices/Modal Segments

Eventually we will arrive at a more or less complete description of a family of computer controlled mechanisms which we will call *MDL devices.* (Here MDL refers to "motion description language".) The full description of these mechanisms will come in stages, only the first part of which is given in this section.

Consider a mechanism with inputs and outputs as above. Suppose, furthermore, that it is embedded in a computing environment which includes

i) Registers, buffers and random access memory.

ii) Analog to digital converters operating on a fixed cycle of period τ, producing a stream of sampled and quantized values of y.

iii) A microprocessor which can operate on the stream of sampled values of y, together with suitable descriptions of u and $k(\,)$ from memory, so as to produce digital representation of a stream of values of $u(t) + k(y(t))$ delayed relative to the y stream by less than τ units of time.

iv) A digital to analog computer for converting the digital representation of $u(t) + k(y(t))$ to analog form.

We do not specify either the sampling period τ or the precision of the digital representations. Our point of view is that u and k will be downloaded to the MDL device with more accuracy than any practical implementation could use. We suppose that τ is small compared to the mechanical/electrical time constants of the system and that the implementation is such that making τ smaller will only improve the approximation which the actual path makes to the path predicted by the differential equation model. Thus all deviations from the ideal are the result of τ being positive rather than being infinitesimal and the word length used to describe y and v being finite rather than infinite.

So far then a MDL device is a computer controlled mechanism which accepts u and k and forces x along a path which closely approximates the solution of

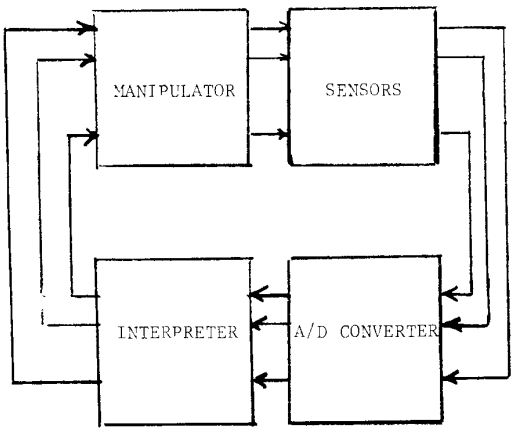

Figure 1: Partial Description of a MDL-Device.

$$\dot{x}(t) = f(x(t)) + G(x(t))(u(t) + k(y(t)))$$

with quality of the approximation improving as τ tends to zero and the word length tends to infinity. However, because different parts of a task may call for different control policies, instead of just transmitting (u, k) to the MDL device we will transmit a triple (u, k, T) with the understanding that T defines the epoch over which the (u, k) pair is to be used. We will refer to triples (u, k, T) as being *modal segments* because they specify a mode of control over a segment of time. To be more explicit, what we have in mind is that on receipt of an input string (u_1, k_1, T_1) (u_2, k_2, T_2) $\ldots (u_r, k_r, T_r)$ the MDL device executes a motion which closely approximates the $x(\ )$ defined by

$$\begin{aligned}
&\dot{x} = f(x) + G(x)(u_1 + k_1(y)) \quad ; \quad 0 \le t < T_1 \\
&\dot{x} = f(x) + G(x)(u_2 + k_2(y)) \quad ; \quad T_1 \le t < T_1 + T_2 \\
&\cdots \\
&\dot{x} = f(x) + G(x)(u_r + k_r(y)) \quad ; \quad \sum_{i=1}^{r-1} T_i \le t < \sum_{i=1}^{r} T_i
\end{aligned}$$

Both the sampling rate $1/\tau$ and the rate at which (u, k, T) instructions can be processed are important measures of the performance of an MDL device.

Expressiveness of Affine Modal Segments

In order to be able to implement a system which interprets a family of modal segments it is necessary to index the possible modes in a finite way. This necessitates restricting the set of all possible u's and k's in some way. A natural choice is to take what is, in effect, the first terms in a Taylor series expansion for $u + k(y)$. By an *affine modal segment* we understand a (u, k, T) of the form

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} + \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mp} \end{bmatrix} \begin{bmatrix} y_1 - b_1 \\ y_2 - b_2 \\ \vdots \\ y_p - b_p \end{bmatrix}$$

with the u_i, a_{ij}, b_i, and T all being floating point numbers. For a system with m inputs and p outputs such modal segments are described by a total of $(m + 1)(p + 1)$ numbers. We now establish two slightly technical results supporting the idea that restricting the implementation to affine modal segments does not, in a practical sense, limit the expressiveness of the interpreter/mechanism.

More precisely, we make the following claims:

a) If f and G are continuous functions of x whose components satisfy a Lipshitz continuity condition, then affine modal segments can, in the limit as τ goes to zero, be used to generate an arbitrarily good approximation to any curve which the mechanism is capable of generating.

b) If f and G are differentiable functions of x and if there is any u, k pair which makes x_0 an equilibrium point while making the solution $x(t) = x_0$ an exponentially stable solution of $\dot{x} = f(x) + G(x)(u + k(x))$, then there is an affine control law $u + k(y - b)$ such that the affine control law makes x_0 an equilibrium point such that the solution $x(t) = x_0$ is exponentially stable.

The importance of the first of these statements is obvious; the importance of the second may be explained as follows. In constructing motion control programs one is, in the first instance, concerned with kinematics; at a more refined level dynamics become important. In order to minimize the importance of dynamics, that is to minimize the importance of the short term transient effects, one may try to keep the motion close to a series of equilibrium states. If these equilibria are stable, this procedure will be robust. It may also be noted that stable equilibrium points are to mechanisms what wait states are to microprocessors, necessary evils useful for solving timing problems.

We can easily prove both of the above assertions. With respect to the first one, a proof goes as follows. Let $\bar{x}(t)$ be any solution of

$$\dot{x}(t) = f(x(t)) + G(x(t))v(t)$$

According to a well known result in differential equation theory, (see, e.g. [5]), the standard Euler approximation to this solution, obtained by solving the difference equation

$$x(nh + h) = x(nh) + hf(x(nh)) + hG(x(nh))u(nh)$$

and using straight line interpolation, converges, as the step size h goes to zero, to the true solution. Letting $T = \tau$,

letting $k = 0$, and letting $u_i = v(hi)$ we see that we get a system which, if approximated by Euler's method, would have the same approximation as the original system. Thus, by the result cited, as h goes to zero, the solution of the system driven by the (u, k, T)'s approaches the original solution. We note in passing that this proof describes a method of achieving expressiveness at the expense of dealing with a high rate of (u, k, T) communication. As far as the second result is concerned. Suppose that

$$\dot{x}(t) = f(x(t)) + G(x(t))(u(t) + k(h(x(t))))$$

has x_0 as an equilibrium point. Then u must be constant and

$$0 = f(x_0) + G(x_0)(u + k(h(x_0)))$$

Expand $u + k(h(x))$ near x_0 as

$$\begin{aligned} u + k(h(x)) &= k(h(x_0)) + \frac{\partial k}{\partial y} \cdot \frac{\partial h}{\partial x}(x - x_0) \\ &= \tilde{u} + PQ(x - x_0) + \text{nonlinear terms} \end{aligned}$$

If we use an affine modal segment of the form $(\tilde{u}, Py, T)$, then the linearization of the modal controlled system will be the same as that of the original system. However, as is known (see [6]), exponential stability depends only on the linearization.

Coordinate Changes/Inverse Kinematics

Because the sampled value of $y(t)$ will change every τ seconds (e.g. perhaps as often as 10^3 times per second) it is necessary to be able to evaluate $k(y(t))$ rapidly. This is to be carried out, using special purpose hardware if necessary, by the interpreter. Even a modest system might call for $(m + 1)(p + 1) \approx 100$ and even if just 4 bytes are allocated for a floating point number, then about 400 bytes will be required for the description of a single affine modal segment. It will be advantageous to arrange matters so that the interpreter/mechanism can be adequately expressive with (u, k, T)'s which, on average, have lifetimes of $10^2 \cdot \tau$ to $10^3 \cdot \tau$, since shorter lifetimes would result in unreasonably high communication rates between the (u, k, T) generator and the interpreter/mechanism. To enhance the expressiveness of single modes, as opposed to relying on frequent mode changes in order to achieve expressiveness, we introduce an additional operation. Its availability will, at the same time, make it easier to write succinct motion control programs. The operation is a type of coordinate change whereby u is replaced by Φu for Φ some invertible p by p matrix whose entries depend on y. The idea is that when the interpreter receives a Φ, subsequent modal segments will enforce the rule

$$\dot{x} = f(x) + G(x)\Phi(y)(u + k(y))$$

We need to explain why this particular type of coordinate change is the most crucial one to implement. One of the basic problems in the control of robotic mechanisms is the so-called inverse kinematics problem. In its simplest form, it deals with the problem of inverting the vector valued relationship $z = g(\theta)$ which maps equilibrium configurations corresponding to the motor positions $(\theta_1, \theta_2, \ldots, \theta_q)$ to end effector positions $(z_1, z_2, \ldots, z_r)$. If we differentiate g, we get a relationship expressing the velocities of the z's in terms of the velocities of the θ's

$$\dot{z} = G(\theta)\dot{\theta}$$

Assuming that the θ's can be controlled, i.e. that they are, in effect u's, we can "undo" the system kinematics by replacing $u = \dot{\theta}$ by $G^{-1}(\theta)u$. Above, in the definition of the change of coordinates, we asked that Φ depend only on the sensor outputs y. We have, however, insisted that all kinematic variables be sensed so that G^{-1} is actually expressible in terms of y.

Because of dynamical effects, the above algorithm for inversion of the system based on kinematics alone will only provide an approximation to the true dynamical inverse. The quality of the approximation improves as the velocity and acceleration terms approach zero. True dynamical effects will not be discussed here since they are, in terms of systems of the complexity envisioned here, not as important as the kinematics.

Even within these constraints not all the types of coordinate changes which can be envisioned can be implemented; we discuss only one particular type which, however, includes the usual "inverse kinematics" of open chain robots and a number of other cases as well. In our earlier paper [7] we showed that the homogeneous coordinate matrix X associated with an open chain manipulator can be expressed as

$$X(\theta_1, \theta_2, \ldots, \theta_q) = e^{A_1\theta_1} e^{A_2\theta_2} \ldots e^{A_q\theta_q} X_0$$

with $\theta_1, \theta_2, \ldots, \theta_q$ being the motor shaft angles and $A_1, A_2, \ldots, A_q$ reflecting the link geometry. It is shown in the paper referred to that the derivative with respect to time can be written as

$$\begin{aligned} \frac{d}{dt}X(\theta_1, \theta_2, \ldots, \theta_q) &= (A_1\dot{\theta}_1 + e^{A_1\theta_1}A_2\dot{\theta}_2 e^{-A_1\theta_1} + \ldots \\ &\quad + e^{A_1\theta_1}e^{A_2\theta_2} \\ &\quad \ldots e^{A_{q-1}\theta_{q-1}}A_q\dot{\theta}_q e^{-A_{q-1}\theta_{q-1}} \\ &\quad \ldots e^{-A_2\theta_2}e^{-A\theta_1}) \\ &\quad X_0(\theta_1, \theta_2, \ldots, \theta_q) \end{aligned}$$

The merit of this expression is that it can be used to display the jacobian of the map from θ to X as a relatively simple combination of functions of one variable rather than as an arbitrary function of n variables. This will allow us to use table lookup in an advantageous way.

We want to establish a result here on the evaluation of a function of several variables and its differential. In view of the results of [7] it applies directly to the change of coordinates problem.

Definition: We will say that a function ϕ of the variables $(\theta_1, \theta_2, \ldots, \theta_q)$ is *exponentially separable* if, after a possible reordering of the θ's, there exists a positive integer n, a set of n by n matrices $A_1, A_2, \ldots, A_q$ and a pair of n-vectors c and b such that

$$\phi(\theta_1, \theta_2, \ldots, \theta_q) = ce^{A_1\theta_1}e^{A_2\theta_2}\ldots e^{A_q\theta_q}b$$

Theorem: Suppose that ϕ is an exponentially separable function of $\theta_1, \theta_2, \ldots, \theta_q$. Denote by L a grid of l^q points $\{\theta_1, \theta_2, \ldots, \theta_q | \theta_i = \text{ multiple of } \delta, -l/2 \leq \theta_i < l/2\}$, then with the storage of $qln^2 + 2n$ real numbers, ϕ can be evaluated at any point in L by reading qn^2+2n real numbers from memory and performing $qn^3 + 2n$ scalar multiplications. Moreover, the partial derivatives $\partial\phi/\partial\theta_i$, $i = 1, 2, \ldots, q$ can be evaluated at any point in L by reading $qn^2 + 2n$ real numbers from memory and performing $2qn^4 + 2nq$ scalar multiplications.

A homogeneous coordinate matrix in three dimensions contains 12 nonconstant entries. If we divide 2π into v equal segments, then with $12v$ numbers or, say $48v$ bytes, we can store an approximation to $e^{A_i\theta}$ for $0 \leq \theta < 2\pi$. In order to form the set of matrix valued coefficients for $\dot{\theta}_1, \dot{\theta}_2, \ldots, \dot{\theta}_q$ in the above expression we need to carry out $2q$ matrix multiplications. (At first sight it may seem that it takes about q^2 matrix multiplications but the products are nested in a certain way so that computing $e^{A_1\theta_1}, e^{A_1\theta_1}e^{A_2\theta_2}$, ..., $e^{A_1\theta_1}e^{A_2\theta_2}\ldots e^{A_q\theta_q}$ takes only $q-1$ matrix multiplications.) Thus to evaluate G in the kinematic equation

$$\dot{x} = G(x)v$$

takes only $12 \cdot v \cdot 2 \cdot (n-1) \cdot 16 = 384 \cdot v \cdot (n-1)$ multiplications. (The leading constant in this bound can be improved by looking in more detail at the block upper triangular structure of the matrix multiplication involved.)

Having $G(y)$ consider the matrix differential equation

$$\dot{H} = H(GH - I)$$

If G^{-1} exists and $H(0)$ is sufficiently close to G^{-1}, the solution of this equation tends to $H_0 = G^{-1}$. To evaluate the right-hand side of the differential equation takes 2 n by n matrix multiplications. Thus we can take one step in an Euler method at that cost in computational complexity.

As in the case of the (u, k, T)'s, it is necessary to restrict the choice of Φ's to some finite set and to index this set. Our solution to this will be simply to designate a finite subset of the set of exponentially separable Φ's. The justification of this is that most robotic mechanisms need just a few such changes of coordinates, e.g. changes that correspond to rectilinear or cylindrical movement in end-effector space.

Sorting all this out then, we see that we can keep a running value of H available to form $H(y)(u+ky)$ as needed to implement the change of coordinates. In this sense, then, we can assert that the above change of coordinate map lets us program in end effector coordinates simply by selecting a Φ which inverts G. For systems with moderately complex dynamics this can be done at an acceptable speed using, say, a five megaflop digital signal processing chip for the matrix multiplication and a megabyte of ROM to store a table of values for the matrices. What we have in mind is that a given MDL device will have a built-in finite list of coordinate changes which are supported by the combination table look up/computation scheme described here and that the user can select from this list of coordinate changes. This situation is not unlike what one finds with respect to printers and fonts; a given printer supports a number of fonts, usually defined in ROM, and the user can select from this list.

Internal States

It usually happens in the design of feedback compensation for control systems that it is desirable to introduce dynamic compensation. By this one means feedback control policies which are not just $v = k(y)$ but rather something like

$$v(t) = ky(t) - \int_0^t (y - y_0)d\sigma$$

Without the availability of such dynamic compensation terms the control system designer is severely hampered.

In order to achieve this kind of flexibility we supplement y with certain components which are associated with "simulated dynamics". That is, in addition to an MDL device having the properties ascribed above, we ask that it should have the capacity to run vector valued iterations, as specified by the instructions. More specifically we postulate that in addition to properties *i)* through *iv)* listed above MDL devices have the following capability.

v) Internal to the computing environment is the possibility for simulating α-dimensional vector equations

$$\dot{x}_e = v_e \quad ; \quad y_e = x_e$$

That is to say, having specified nonnegative integer α, the α-dimensional vector x_e is appended to the vector of quantized values of y and the vector v_e is appended to the quantized values of v. This means that the u and the k in a modal segment (u, k, T) is then "enlarged" in the sense that k depends on y and y_e and maps to x and x_e whereas u is enlarged to include an x_e part.

Once these internal states are added, the v and y of the mechanism are supplemented with v_e and y_e of the internal states and the total system is then a quantized, discretized version of

$$\begin{array}{l} \dot{x}(t) = f(x(t)) + G(x(t))v_{\text{old}} \\ \dot{x}_e(t) = v_e(t) \end{array} \quad ; \quad y = \begin{bmatrix} y_{\text{old}} \\ x_e \end{bmatrix}$$

with input

$$v = \begin{bmatrix} v_{\text{old}} \\ v_e \end{bmatrix}$$

In particular the size of the vectors u and b and the matrix k in the description of an affine modal segment increase as the result of adding simulation states.

In addition to being useful for feedback compensation, simulated states can be useful for generating certain types of nonstraight-line motions. For example, if one wants to generate a circle it is possible to use a two dimensional

internal vector with an affine modal segment selected so that

$$\dot{x}_e = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} x_e$$

since the solution of this equation is

$$x_e(t) = \begin{bmatrix} \cos t & \sin t \\ -\sin t & \cos t \end{bmatrix} x_e(0)$$

Recording

As we have argued before, at the interface between the interpreter and the mechanism interpreter the data rates are quite high but within the interpreter the computation is highly structured. At the level of the modal segment generator the data rates are lower but the computation has less structure. Ordinarily past values of the measured quantities, represented by y, are not needed or used. However there arise situations involving adaptation and learning for which past values of y are required. How can we accommodate an occasional request for past data without saturating the capability of the system?

Clearly one cannot keep the entire past of y in memory. On the other hand, a moderate size ring buffer storing the most recent 10^5 samples is perfectly reasonable and represents almost no computational burden. At 200 samples per second, and with y being 20 dimensional, this represents a 25 second epic. In order to get this information out of the modal segment interpreter we must enlarge our free monoid L so as to have it include atoms representing requests for recent data.

On receipt of the atom (d, α) the interpreter will return the present time (a time stamp) followed by a string of α real numbers consisting of the most recent samples of each of the variables $y_1, y_2, \ldots, y_\alpha$.

Example for Illustration

In order to give the reader a specific example of how this works, without getting into too much detail, we consider an idealized situation. Consider a mechanism with a differential equation description

$$\dot{x}_1 = v_1 \ ; \ y_1 = x_1 \ ; \ y_3 = f x_1$$

$$\dot{x}_2 = v_2 \ ; \ y_2 = x_2 \ ; \ y_4 = f x_2$$

That is, we consider a point which moves in the plane. There are four scalar variables which are sensed, these are the positions x_1 and x_2 and the x_1 and x_2 components of the forces felt by this point. If we assume that the forces are generated by the point (x_1, x_2) approaching objects fixed in the x_1, x_2 plane, then we can suppose that $f_{x_1} = \phi_1(x_1, x_2)$ and $f_{x_2} = \phi_2(x_1, x_2)$. We undertake to control this system using affine modal segments. Thus we can force the system to follow any piecewise smooth segmented path generated by differential equations of the form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \phi_1(x_1, x_2) \\ \phi_2(x_1, x_2) \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

A sample programming problem would be, say, to find a string of modal segments which drives the system from its initial value to a point where the force is $f_{x_1} = 1$, $f_{x_2} = 0$ and leave it there for a certain period.

Experimentation

For the most part, the ideas advanced here are the outgrowth of efforts in the Harvard Robotics Laboratory directed toward the development of an effective control environment for a two finger, four degree of freedom planar manipulator. This system consists of

a) The manipulator hardware, sensors, motors and amplifiers

b) A card cage with A/D and D/A converters, motion control cards and a single board computer

c) A Sun 3/110 with a Bus extender,

The software consists of device drivers and a partial implementation of the $(u, k, T) - (\Phi) - (d, \alpha)$ setup has been written by Alex Bangs, Josip Lončarić and, especially, Victor Eng.

Figure 2 shows the system in schematic form. Typical tasks of interest include inserting pegs in holes, manipulating objects, etc.

There are ten sensor outputs, four positions associated with motor shaft angles, two pressure sensors from the fingertip force sensors and four current sensors from the motors. These are sampled at approximately 300 Hz and made available on the VME Bus.

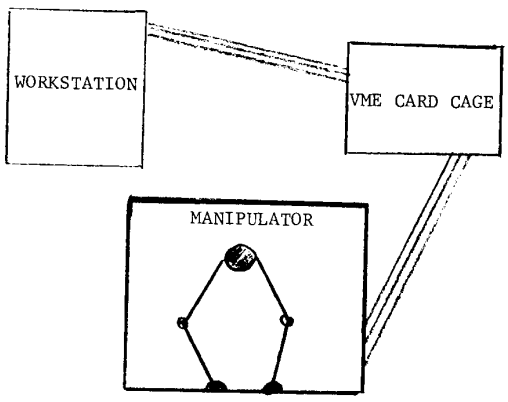

Figure 2: The hardware schematic for a planar manipulator.

The motion control cards operate on input strings and optical shaft encoder (position) data to produce pulse width modulated signals for the motor amplifiers. The commands to these boards must be sent in ASCII which results in a significant communication bottleneck. Each card can handle two motors and provides for some coordination between the motors controlled by the same card. The commands which these boards accept include feedback gains and setpoints. Unfortunately, the input strings which they accept are not unconstrained (i.e. do not form a free monoid) and the communication rates available are too low for some purposes.

Towards a Planning Medium

In order to be effective, the three types of primitives introduced here, the (u, k, T) primitive, the Φ primitive and the (d, α) primitive need to be packaged by an applications program with a number of editing and file handling features. We envision that human planning of the motion sequence will take place with the aid of such a program. In effect it will be the premotor cortex of the system. More specifically, this "choreography" program needs to incorporate in a smoothly integrated way

i) file handling to facilitate storage and recall of motion sequences,

ii) editing capabilities for splicing, time warping, scaling and translation of motions,

iii) a facility for previewing motions,

iv) algorithms for the optimization of motions via smoothing, interpolation, time warping, etc.

The printer analogy is again suggestive. *Postscript* [8] can be thought of as being a type of motion description language; to make full use of it, one needs an applications program facilitating its use. Generating motion is at least as complex as generating descriptions of printed pages and deserves to have much more elaborate support than is presently available.

Conclusions

We have described certain features of a three-part solution to the problem of general motion control. It involves a motion description language together with an interpreter mechanism and an applications program. The language is, to a large extent, device independent and thus applicable to a wide variety of systems. Its basic programming mode is that of specifying a position/force dependent velocity for the mechanism and a period over which the velocity specification is to be enforced. An analysis is given to show that a suitable type of change of coordinate maps with this mode of programming can be supported and is capable of generating a wide variety of motions. An existing experimental facility implementing ideas of this type is briefly described.

References

[1] N. Bernstein, *The Coordination and Regulation of Movements*, Pergamon Press, Oxford, England, 1967.

[2] G. Hinton, "Some Computational Solutions to Bernstein's Problems", in *Human Motor Actions - Bernstein Reassessed*, (H. T. A. Whiting, Ed.), North Holland, Amsterdam, 1984.

[3] David Marr, "A Theory of the Cerebellar Cortex," *J. of Physiology*, Vol. 202, 1969, pp. 437-470.

[4] James Albus, "A Theory of Cerebellar Function," *Math. Biosci.*, Vol. 10, 1971, pp. 25-61.

[5] E. Coddington and N. Levinson, *Theory of Ordinary Differential Equations*, McGraw-Hill, New York, 1955.

[6] W. Hahn, *Theory and Application of Liapunov's Direct Method*, Prentice Hall, Englewood Cliffs, NJ, 1963.

[7] R. W. Brockett, "Robotic Manipulation and the Product of Exponentials Formula," *Mathematical Theory of Networks and Systems*, (P. A. Fuhrman, Ed.) Springer-Verlag, Berlin, 1984.

[8] *Postscript Language Reference Manual*, Addison-Wesley, Reading, MA, 1985.