Human Motor Actions – Bernstein Reassessed
H.T.A. Whiting (editor)

CHAPTER IVb

SOME COMPUTATIONAL SOLUTIONS TO BERNSTEIN'S PROBLEMS

G. Hinton

INTRODUCTION

Bernstein's great strength was his appreciation of the immense complexity of the apparently simple task of motor control. He lived at a time when movements were seen as "responses", and their structure was generally regarded as a minor matter, of much less interest than their variable relationship to the "stimuli" that elicited them. Against this background, he insisted on analyzing the difficulties inherent in real motor acts. He understood the profound problems caused by reactive forces, surplus degrees of freedom, the unpredictability of external forces, and the need to use sensory feedback to modulate the effects of central commands. One may quibble with some of his solutions, but his understanding of the major problems was far ahead of his time.

In this commentary, I shall focus on three issues that were of major concern to Bernstein. First, what can we infer about the code that the brain uses to communicate with the periphery, and what does that tell us about how the computation is organized? Second, if the brain knew just what movements it wanted the body to make, could it figure out what to tell the muscles in order to make it happen? Third, how is it possible to coordinate a system with so many degrees of freedom that interact in such complex ways? How does the brain make sensible choices among the myriad possibilities for movement that the body offers?

Experiments on people, animals, and robots, and a better understanding of the computational problems have led to interesting theoretical advances in the last few years. I shall concentrate on explaining some of these developments and showing how they provide solutions to some of Bernstein's problems. To simplify matters, I assume a particular task - rapid reaching movements to a visually specified point without visual feedback from the movement. This task is complex enough to contain many of the problems that interested Bernstein but simple enough to be tractable. I shall have little further to say about the many important problems that the task avoids. These include:

1. *Impedance control*: By setting the stiffness of the muscles appropriately, it is possible to control the mechanical response of the body to externally applied forces (Benati et. al., 1980b; Hogan, 1980).

2. *Manipulation planning*: Skilled manipulation involves figuring out what forces, impulses, or movements to apply to objects and how to use the arms and fingers to do it (Mason, 1981; Lozano-Perez, 1979).

3. *Dynamic refinement of motor plans*: Planning is just one component of manipulation. Another crucial aspect is the use of tactile feedback to dynamically refine under-specified aspects of the plan as it is being performed. Bernstein emphasises that this type of close integration of movements and feedback is a central feature of motor control.

4. *Locomotion*: The maintenance of dynamic balance while walking or running is an area where rapid progress is currently being made in the computational theory (Raibert & Sutherland, 1983).

1. THE EQUIVOCAL EFFECTS OF CENTRAL COMMANDS

Suppose you want your finger to trace out a geometrical form in space. To do this you must activate the muscles in your arm so that they generate the necessary torques at the joints (though this may not be the best way to think about it). As Bernstein points out, generating the right torques is harder than it seems for several reasons:

1. The force exerted by a muscle depends not only on its level of activation, but also on its current length and the rate at which it is changing. So neural impulses to the muscles cannot unequivocally specify the forces to be exerted.

2. The magnitude and axis of the torque created by a given force in a muscle depends on the geometrical configuration of the muscle about the joint, and this varies as the joint angle changes.

3. It is hard to know what torques *should* be exerted to achieve a desired effect on a joint angle. The effect of a torque depends on the angular inertia about the joint, and in a moving chain of linkages like the arm, the angular inertia about each joint depends on the angles and *angular velocities* of all the joints. Also torques applied at one joint produce angular accelerations at *other* joints.

4. Even if the torques could be computed centrally and applied unequivocally by the muscles, the problem would still not be solved because small variations in the static or dynamic configuration of the body cause the same torques to produce different effects, and the central apparatus cannot know what the exact state of the body will be when its torque signals arrive there.

Between them, these arguments are certainly compelling evidence against the idea that the central apparatus simply sends out the right force signals to the muscles. They suggest that some more subtle code is used and that some peripheral computation is involved. This is an important discovery. Bernstein, however, seems to draw a further conclusion which is not justified and which hinges on a false assumption and a mathematical error. He appears to believe that the central apparatus is not concerned with torques and that it somehow deals with a different kind of representation that only gets turned into torques at the periphery. I think he would have been firmly opposed to the idea that the required torques are computed centrally and are then encoded in some other form for transmission to the muscles. I also think that this may be what actually happens.

Bernstein's false assumption is never explicitly stated, but is clearly evident in his discussion of the six properties required of every self-regulating system. He assumes that the detection of error and the corrective feedback signal are both neural events. He does not seem to have realized that the complex properties of muscles, which make them respond equivocally to neural impulses, are not a *problem* that makes motor control more difficult but an evolutionary adaptation that makes motor control easier. These special properties allow muscles themselves to act as very fast, negative feedback devices without any neural detection of the error (as will be explained in the next section).

Naturally, neural feedback is very important in real motor acts and the existence of other types of feedback does not mean that we can ignore the complex problem of how neural feedback is integrated into the action. However, physical feedback effects must be properly understood before it is possible to analyze the role played by neural feedback loops in real motor acts, even if these neural loops are very fast (Adams, 1976). I therefore describe in some detail how the physical properties of muscles can be used to simplify motor control.

I shall show that muscles can provide almost instantaneous feedback effects *provided* that the signals sent to the muscles use an indirect encoding of the required forces. Thus the fact that the signals do not directly encode torques (or rather the muscular forces needed to generate them) may not be because the forces are too hard to compute or because evolution could not produce muscles that responded unequivocally to force signals, but because the centrally computed forces need to be encoded in a special way to cash in on a neat trick for getting instant feedback.

When this trick was first discovered it was initially interpreted as a way of avoiding force computations altogether. The feedback provided by the muscles was seen as a way of moving the limbs around without any explicit force computations, but there is now convincing evidence against this interpretation (Hogan, 1982; Bizzi, Accornero, Chapple and Hogan, 1982), and it seems probable that the properties of muscles allow them to produce a very rapid feedback effect, but the forces still need to be precomputed. In general, it is always better to combine a precomputed feedforward component with a feedback loop than to rely on feedback alone. Feedback can only produce forces when there are errors, so big forces require big errors (Horn, 1978). By using feedforward to generate some of the required force, the errors can be reduced to those needed to produce the difference between the precomputed force and the force that is actually needed.

Bernstein's mathematical error is his claim that reactive interactions make the computation of torques exponentially hard. This claim underpins his view that the central apparatus does not explicitly compute torques. He is just plain wrong about the difficulty of the computation. It is not trivial, but it is now known to be quite easy. It has linear rather than exponential complexity. (This is discussed in more detail in section 2). Bernstein can hardly be faulted for his mistake. The easy way to do the computation has only been discovered in the last few years, and even this algorithm is quite complex when compared with the kinds of computation people were willing to attribute to the brain before we had modern computers.

Bernstein's overestimation of the difficulties created by reactive interactions, and his failure to realize *why* the signals to muscles did not directly represent forces appear to have led him to reject the idea of a full internal model of the dynamics of the body, and I think this was a serious mistake.

WHY LENGTH-TENSION FUNCTIONS MAKE GOOD CONTROL SIGNALS

Although it is intellectually easier to imagine that central signals specify forces or torques, it is actually much easier to control a system by sending out signals that specify *functions*. The α-motoneuron inputs to a muscle specify not a force, but a function that relates the length and the rate of change of length of the muscle to its tension, and hence to the force and torque that it generates (Asratyan and Feldman, 1965). From a design standpoint, the use of "length-tension" functions seems like an unnecessarily roundabout way of getting the muscles to generate the right forces. Why didn't we just evolve muscles that unequivocally generate a required force when told to do so?

To understand why it is better to specify length-tension functions than forces, consider the following simple problem: Suppose we have a system with just one joint, and we wish to change the joint angle according to some internal schedule which specifies what the joint angle should be at each instant. If there is an internal model of the dynamics of the physical system, it is possible to compute what torque is required at each moment in order to follow the required trajectory. If there is a torque motor at the joint, all we need to do is to tell it the computed torque. This open-loop scheme is simple but it is not robust. If the internal model is imperfect in any way, the computed torques will not be quite right, and the joint angle will diverge from its desired value. We could add a mechanism that sensed these deviations and modified the torque signals appropriately, but this mechanism would be subject to delays due to the slowness of neural transmission.

Now consider the alternative scheme using "angle-torque" functions. The internal model is used to compute the required torque, as before, but then a further computation takes place. The required torque and the desired joint angle are used to select an angle-torque function which will yield that torque at that angle. Then this function is sent down to an angle-torque motor which combines the angle-torque function with the current joint angle to generate the actual torque. Clearly, if the internal model is correct, this scheme will generate the internally computed torque. But why bother to turn the torque into a function and then convert it back again in the motor? The reason is that this scheme makes the system robust against errors in the internal model.

Suppose the internal model underestimates the physical mass that is being moved. In the simple open-loop scheme, the torque motor will always generate torques that are too small. In the angle-torque scheme, the internally computed torques will also be too small, but when these are combined with the current angle to select an appropriate angle-torque function, the *desired* value of the angle will be used. When the function is turned back into a torque by the angle-torque motor, the *actual* angle is used. So the torque actually generated differs from the computed torque, and it differs in an interesting way. If the physical system lags behind the internal model, the angle-torque function will generate a larger actual torque than was intended. So by using angle-torque functions we get an automatic feedback effect. The discrepancy between the internally desired joint angle and the actual joint angle gets converted into an increment in the actual torque. If the angle-torque function is linear, the mathematics is easy. The torque, τ, can be expressed as:

$$\tau = s(\alpha - \alpha_0) \tag{1}$$

where α_0 is the angle at which this function would yield zero torque, α is the joint angle, and s is the stiffness. (The equation assumes that angles are measured in the opposite direction to torques). The two parameters which can be varied to generate the different functions within this class are α_0 and s. At any particular moment, they must be chosen to yield a particular function which gives the appropriate torque at the appropriate angle. In other words, the function must be selected so that:

$$\tau_c = s(\alpha_c - \alpha_0)$$

where τ_c is the internally computed torque and α_c is the internal specification of the required angle. The *actual* torque generated is given by:

$$\begin{aligned} \tau_a &= s(\alpha_a - \alpha_0) && \text{(2a)} \\ &= s(\alpha_c - \alpha_0) - s(\alpha_c - \alpha_a) \\ &= \tau_c - s.e && \text{(2b)} \end{aligned}$$

where α_a is the actual angle and e is the difference between the actual and intended angles. Thus, by encoding the required torques into angle-torque functions we get an automatic negative feedback term in addition to the computed torque. Moreover, this feedback term does not require any neural sensing of the positional error. It is just a property of the way the angle-torque motor behaves. If the functions are non-linear, the mathematics is more complex, but the actual torque can still be characterized as the computed torque plus a negative feedback term that depends on the positional error.

In equation 1 there are two quite different parameters, α_0 and s which determine an angle-torque function. At first sight this seems redundant, because a single free parameter, α_0, would allow a function to be chosen that would satisfy equation 1 (i.e. would generate the desired torque at the desired angle). However, equation 2 shows why it is advantageous to be able to vary s as well as α_0 when choosing an angle-torque function. The value of stiffness, s, determines the size of the feedback term for any given positional error. If the internal model is fairly accurate, the discrepancy between τ_a and τ_c will be small and so the positional error, e, will not need to grow very big before the feedback term s.e is sufficient to compensate for the difference between τ_a and τ_c. If the internal model is inaccurate a larger value of s.e will be needed. If s was a constant, a bigger feedback term would require a bigger error to generate it. If, however, the internal model is *known* to be inaccurate a larger value of s can be used so that the necessary feedback is generated without requiring large positional errors. This use of high stiffness to overcome the inadequacies of a poor internal model fits in well with Bernstein's observation that when first performing a new task people's movements are very stiff but that they later become more relaxed as they master the task.

Even if there is no reliable information about the inaccuracy of the internal model it is still possible to dynamically optimize the stiffness parameter so that the errors are kept within reasonable bounds with as little stiffness as possible. Houk (1979) has suggested that neural detection of the error in position may be used to control the stiffness. This rather indirect type of feedback has a major advantage over the more conventional idea of using the neurally detected error to directly generate a corrective force. Neural detection and transmission are slow, and by the time an error has been detected and fed back as a correction, the system may have changed enough so that the original error is no longer what needs to be corrected. Indeed, the current error may be in the opposite direction so that the correction only makes things worse (this is the classic cause of oscillations in systems with delayed feedback). By modulating the stiffness of the angle-torque function and leaving the actual feedback to the muscles themselves, these problems are avoided because the feedback is all in the physics so there is very little delay.

There are no angle-torque motors in the body, but by using two or more opposing muscles about each joint, it is easy to implement angle-torque functions provided the muscles exhibit the linear equivalent - length-tension functions. In the simplest case, the relationship between the length-tension functions of two opposing muscles and the angle-torque function they implement is easy to state. Simple length-tension functions have the form:

$$t = s(l - r)$$

where t is the tension, s is the stiffness, l is the length and r is the resting length. If the leverage of the muscle about the joint remains constant, this can be rewritten in angular terms as:

$$\tau = s(\alpha - \rho)$$

where α is the joint angle for which the muscle has lengt l and ρ is the angle for which it has its resting length. The net torque generated by two opposing muscles is therefore:

$$\begin{aligned}\tau_{net} &= \tau_1 - \tau_2 \\ &= s_1(\alpha - \rho_1) + s_2(\alpha - \rho_2) \\ &= (s_1 + s_2)(\alpha - (s_1\rho_1 + s_2\rho_2)/(s_1 + s_2))\end{aligned}$$

This has the same form as equation 2a with s equal to the sum of s_1 and s_2 and α_0 equal to the weighted average of ρ_1 and ρ_2, using s_1 and s_2 as weighting coefficients.

More complex cases in which there are many muscles acting about a joint, non-linear length-tension functions, and varying leverages of a muscle about a joint are trickier, but they retain an important modularity. The required length-tension functions can be computed from the required angle-torque function, and so this function is all that needs to be communicated. Decisions about how to implement it can be made locally.

A further complication is created by the fact that many muscles act about two different joints. This means that the length of the muscle and hence the torque it exerts at any given level of innervation depends on both joint angles. This allows the torque exerted at one joint to be contingent upon the current angle of a neighboring joint, and Hogan (1980) has shown how this property can be very useful for giving the body the right motor impedances.

VISCOSITY PROVIDES A VELOCITY FEEDBACK TERM

The idea that the physics can be used to provide feedback effects is not restricted to positional feedback terms. In many circumstances, it is also helpful to have negative feedback that is determined by the difference between the desired and actual *velocities*.

So far, we have ignored the fact that the force generated by a muscle depends on its rate of contraction. Again, this initially seems to be a complicating factor but actually makes control easier by providing a velocity feedback term. The simple one-joint model illustrates this effect if there is viscosity at the joint (or in the motor). If the internal model of the dynamics of the system includes an accurate representation of this viscosity, it is possible to allow for it in selecting the appropriate angle-torque function. The computed torque then has two components, one to accelerate the system and one to overcome viscous drag. The angle-torque function must be selected to provide the sum of these two torques. Now, suppose the physical system is actually going more slowly than the internal model. There will be less viscous drag than predicted, and so some of the torque that was destined to overcome viscous drag will actually be left over and will go into accelerating the system. This gives a velocity feedback effect. The discrepancy between the actual angular velocity and the intended one gets turned into a torque that reduces this discrepancy. Moreover, no neural comparison of the intended and actual velocity is required. The feedback is generated by the physics.

In the body, there is little viscosity at the joints themselves, but the viscosity in the muscles gives an equivalent effect. From a design standpoint, the viscosity of muscles initially looks like a mistake and one is tempted to see it as a regretable imperfection of the biological hardware. It is true that it consumes energy and it is true that it slightly complicates internal models of the dynamics, but the almost instantaneous velocity feedback term that it provides may make it worth while.

2. COMPUTING TORQUES

An important ingredient of Bernstein's argument for the necessity of co-ordinative structures is his claim that computing torques is extremely difficult because of the reactive forces. This helps support his view that higher levels of the motor system are not directly concerned with torques. The idea is that higher levels use some other code, and the reactive forces are dealt with lower down in the system. This might be feasible if reactive interactions are handled by local feedback mechanisms, but if feedforward components are involved then they are much harder to compute locally because they necessarily involve non-local reactive interactions. Bernstein himself points out the importance of feedforward control towards the end of chapter 4, but he gives no hint as to how it is to be achieved.

Bernstein claims that reactive interactions between different joints in a chain of segments have exponential complexity. In other words, for each extra mechanical degree of freedom, the magnitude of the computation required to figure out the reactive forces is *multiplied* by a constant factor. If this factor is large the problem rapidly becomes intractable. Actually, far from being exponential, the problem is linear. For each new degree of mechanical freedom, a constant amount is *added* to the computation. Bernstein can hardly be blamed for not realizing this, since it was only recently discovered. Over the last few years the complexity of the best known algorithm has fallen from order(N^4)(Uicker, 1965) to order(N)(Luh, Walker and Paul, 1980). This makes it far more plausible that the brain could actually compute the torques required to follow a desired trajectory.

The mathematics used in computing the torques is fairly complicated (Luh et. al., 1980), but the physics underlying it is relatively straightforward. We first convert all the information about the desired trajectory into a single global non-accelerating frame of reference. Then we simply solve the equations of motion for each segment in turn starting with the most distal one. The desired linear and angular accelerations of this segment relative to the global frame are known and they allow us to compute the forces and torques acting at the joint between this segment and the penultimate one. Once these forces and torques are known, we can solve for the unknown forces and torques at the proximal end of the penultimate segment, and so on.

Luh et. al. work with the general case in which sliding joints are allowed as well as rotating ones. If there are only rotational joints, we can further simplify their method by only solving for the torques. Even though the joints are moving through space, at any instant a joint is at a point in space and the angular momentum of the whole of the system distal to the joint about that point in space can only be affected by the torque applied at the joint and by externally applied or gravitational torques. If these latter are known, the torque applied at the joint can be determined from the rate of change of angular momentum of the distal system about the point where the joint now is. Linear forces acting through the joint cannot

affect the angular momentum about it, and neither can torques acting at other joints. The rate of change of angular momentum is simply the sum of the rates of change for each of the segments that is distal to the joint. This means that it is possible to solve for the torque at one joint without first solving for the torques at all more distal joints.

CAN THE BRAIN DO ARITHMETIC

The discovery of an algorithm of linear complexity for computing torques is an important advance for robotics, but it is not necessarily relevant to the issue of whether the brain can do the computation. Computers do arithmetic very well. They represent numbers precisely enough so that the progressive accumulation of rounding errors during a long computation does not swamp the answer, and they perform each operation so fast that they can perform long sequences in a fraction of a second. Brains seems to work quite differently (Von Neumann, 1958). They have neither the speed nor the accuracy for the kind of computation discussed above. But they do have billions of processors each connected to thousands of others. If the computation can be decomposed into many pieces that can be performed in parallel, and if each piece has little sequential depth, both the speed and the accuracy problems can be solved. It may be possible to decompose the torque computation into separate pieces that can be performed in parallel for each joint, at the cost of a certain amount of duplication. It would be necessary, for example, to perform separate additions, at each joint, of the rates of change of angular momentum of the distal segments. This inevitably involves adding together the same numbers several different times, and increases the computational complexity from order(N) to order(N^2). However, the increased number of operations is not large, and may well be worth the time saved by the parallelism. The brain has lots of processors, but not much time. Benati et. al. (1980a) present a way of decomposing a similar computation into pieces each of which can be done in parallel by a hardware module that might correspond to a group of neurons.

It is important to remember that the internal computation of the torques does not need to be nearly as accurate as performance measures might suggest. If angle-torque functions are used, the "instant feedback" effect will take care of minor errors. Even rather inaccurate feedforward computations of the torques yield much better performance than feedback alone, so a sloppy internal model of the dynamics is much better than none at all.

A RECONSIDERATION OF THE EQUIVOCAL EFFECTS OF CENTRAL COMMANDS

Given the preceding discussion of the merits of angle-torque functions and the feasibility of computing torques ahead of time, it is easy to see the flaws in the arguments that suggested that torques are not computed centrally. The equivocal response of muscles to neural signals is just right for implementing angle-torque functions, and the optimal choice of these functions requires precomputation of the required torques. The need for feedback is satisfied by the physics of the system, but this does not mean that the torque computations are left entirely to the periphery, because the advantages of feedforward control over pure feedback require precomputation of torques. Finally, the idea that reactive interactions lead to exponential complexity in the computation of the torques is a plausible conclusion, but it is false.

The use of angle-torque functions is a particular example of a much more general principle. If a central controller does not know the precise state of the system it is controlling, or the precise forces that will be encountered, it can either wait for sensory feedback before it sends down commands for action, or it can send down contingency plans and leave the periphery to decide which contingency applies. Angle-torque functions are just contingency plans that have a very compact form. Even though he does not seem to have realized that muscles themselves can decode contingency plans, Bernstein was well aware of the advantages of the contingency plan approach as a way of organizing the interactions between different levels of control in the motor system.

3. THE DEGREES OF FREEDOM PROBLEM

Bernstein, was committed to achieving a mechanistic understanding of human motor control and he realized that the kinds of mechanism available at the time were totally inadequate. He was therefore forced (in the true spirit of articial intelligence) to speculate about possible mechanisms that might be adequate for the task. Unfortunately, he did not have computers to simulate these mechanisms and so they remained vague in many respects.

Bernstein decided that the existence of surplus degrees of freedom in the motor system posed a major problem for any theory of motor control. There are more joints in the human body than seem to be necessary for any one task, and each joint is typically affected by more muscles than seem to be necessary. This creates two kinds of problem. First, it is hard to decide exactly what to do because there are more degrees of freedom in the way the body moves than there are constraints in a typical "motor problem" so the motor problem cannot uniquely specify its solution. Second, even if we could decide exactly what to do with each joint and each muscle, there are still so many of them to be coordinated that the task is liable to swamp any "central executive", especially if modifications are required to cope with unpredictable external forces or the effects of errors in the internal model.

Bernstein proposed that the motor system handles these complexities by using hierarchical coordinative structures. He does not give a very clear statement of the idea anywhere in this book, but it appears to be an example of a way of handling complexity that should be familiar to all computer programmers. Instead of trying to bridge the gap between the "motor problem" and the neural impulses to the muscles in a single span, the gap is progressively narrowed by using a hierarchy of schemas. At the highest level there are schemas that translate the motor problem into terms that are more suitable for the next level down, and so on all the way down to the muscles. At each level, the schemas mediate between the task requirements passed down by the higher level schemas, and the possibilities that are made available by the lower level schemas, given the current state of the motor apparatus. The main advantage of the hierarchical approach is that the higher levels do not need to be concerned with low-level details. The myriad degrees of freedom provided by the individual muscles do not need to be explicitly considered, much as a general does not need to explicitly consider each soldier's actions in planning a battle.

The idea of hierarchical coordinative structures is very attractive, but it is under-specified and there are several different ways to make it more precise. One interpretation is in terms of a qualitative hierarchy in which each level deals with a different type of entity. Examples of possible types are hand-positions, joint-angles, torques, and muscle activations. A different interpretation of coordinative structures is discussed and criticized later.

A FOUR-LEVEL HIERARCHY

Hierarchical decomposition is an excellent way to deal with complexity when the task can be naturally divided into a number of levels, and decisions can be made at one level with little or no consideration of the level below. Saltzman (1979) has discussed possible levels at length, and makes finer distinctions than the ones which follow. Four good candidates for natural levels in motor control are:

1. *The level of the motor problem*. The planning activities that govern deliberate behavior may give rise to a sequence of relatively well-specified and self-contained motor problems such as "reach out to object A" or "grasp object A". This would allow the planning routines to ignore the details of the movements (though they would have to be sure that they were feasible).

2. *The level of movements of the body*. It may be possible, in solving a given motor problem, to decide how the body should move without explicitly considering the torques required to implement the movements. Similarly, decisions can be made about how the end-effectors should respond to external forces (Mason 1981) without considering the angle-torque functions at the joints that are needed to implement the desired motor impedance of the end effectors.

3. *The level of torques*. The torques required to cause a desired movement can be computed without explicitly considering how the muscles are going to implement these torques. Similarly, the angle-torque functions required to implement the desired impedances of the end-effectors can be computed without considering how these functions are to be implemented.

4. *The level of muscle innervation*. This is the bottom level of the hierarchy, and motor control is considerably simplified if the required muscle properties are only computed after the required torques or angle-torque functions have been decided.

In this modular decomposition, as in most modular schemes, there are various intrusions of lower level constraints on higher level modules. Certain movements, for example, are impossible because the muscles are not strong enough. More importantly, the efficiency or accuracy of certain movements depends on the magnitude or rate of change of magnitude of the muscle forces required for the movement. So the choice of a good movement appears to depend on considerations that are two levels down in the hierarchy. However, the modularity of the levels can probably be saved by incorporating, at each level, heuristics for making choices that will be easy of efficient to implement at the next level down. The heuristics contain *implicit* knowledge about the level below, but they are phrased in the terms of the level they are at, and are thus easy and fast to apply at this level. A concrete example of such heuristics is given below.

In chapter 4, Bernstein argues against the idea that a motor problem like reaching to a particular point in space is solved by computing a spatial trajectory. If he is right, the separation between levels 2 and 3 above may well be wrong, but this argument appears to be flawed. It rests on the observation that there is considerable variation in the trajectory used from trial to trial, but very little variation in the endpoint. This observation certainly rules out any model in which the knowledge of where to reach is encoded as a spatial trajectory, but is does not rule out models in which the spatial trajectory is computed afresh on each trial. Minor variations in the rules used for forming the trajectory could easily give rise to a family of slightly different trajectories that all had the same endpoint. (Slight variations in the coefficients in the model of reaching described below would have just this effect).

Minor variations in the spatial trajectory from trial to trial, would be very helpful in discovering a trajectory that not only solved the problem, but did so with the least effort or greatest accuracy. They would allow changes in the trajectory formation rules to be correlated with measures of performance like energy expenditure, jerk, or accuracy. In this way it would be possible to progressively improve the trajectory formation rules so that they gave rise to trajectories that had good dynamic properties, even though the formation rules did not mention torques or forces explicitly. This is an example of the idea of using heuristics at one level that *implicitly* contain knowledge about lower-level considerations such as torques and forces.

The idea of qualitatively different levels is only one of the ways in which hierarchical structure can be applied to motor control, and although Bernstein clearly distinguished some of these qualitative levels, his notion of coordinative structures seems to have been far richer. Several people, influenced by Bernstein, have proposed a view of coordinative structures which emphasizes the role of higher level schemas in constraining the possibilities at the next level down (Greene, 1972; Turvey, Shaw, and Mace, 1978).

Underlying this view is the belief that surplus degrees of freedom make motor control *harder*, and that they therefore need to be removed by imposing extra constraints. This may be true if control is performed by a sequential central executive with limited resources, but there are parallel, distributed forms of computation which are not hampered by surplus degrees of freedom. Indeed, if the right kind of computation is used, extra degrees of freedom make motor control *easier*. The next section briefly described an iterative distributed computation that coordinates many degrees of freedom in satisfying two goals simultaneously. A simplified task has been used to illustrate the style of computation, but the same style would work for more complex, three-dimensional tasks with more degrees of freedom.

4. A DISTRIBUTED PROGRAM THAT REACHES AND BALANCES

Consider a two-dimensional puppet composed of six segments as shown in Figure 1. The foot always remains fixed and is regarded as the proximal end of the puppet. The joints can all move between limits that are roughly appropriate for a person, and each segment has a roughly appropriate mass. Given a starting configuration and a desired position for the distal end of the lower arm (the tip), the task is to compute a set of joint angles that puts the tip at the goal position, puts the center of gravity vertically above the foot, and is as similar as possible to the initial configuration.

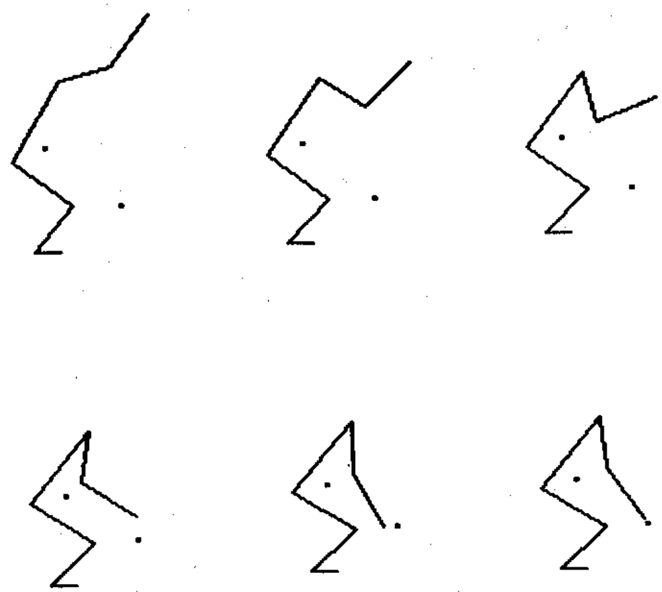


Fig. 1. This shows a sequence of configurations generated by the iterative algorithm. One of the black dots represents the center of gravity of the whole stick-figure, and the other represents the goal to be reached. The configuration is shown on every second iteration. The reason for the overshoot is that in addition to the computed joint increment, half of the previous increment is also added. This smoothes out oscillations and thus allows bigger co-efficients to be used without causing divergent oscillations. Extra processors which control several joints at once were used in this example. These extra processors are described later in the text.

REACHING ALONE

To begin with, let us ignore balance and just consider the problem of getting the tip to the goal. A simple physical analogy suggests a way of organizing the computation. Suppose we took a real pin-jointed puppet and connected the tip to the goal with a rubber band. The tip would move towards the goal and the surplus degrees of freedom would not cause any difficulties. Moreover, the method would work perfectly well if some of the joint angles were temporarily frozen.

It is relatively easy to *approximately* simulate the physics using a distributed computation in which there is one processor for each joint. The rubber band (the "desire vector") generates a torque about each joint that is given by:

$$T_j = d.r_j$$

where d is the magnitude of the desire vector and r_j is the perpendicular distance from the j'th joint to the line along which the desire vector acts. The joint angle is then incremented by a small amount that is proportional to the torque. To ensure that light segments move more easily than heavy ones, the increment, ΔA_j, is also proportional to the moment of inertia, I_j, of the distal portion of the system about the j'th joint:

$$\Delta A_j = k_r.T_j.I_j \tag{3}$$

where k_r is a constant that determines the size of the increments used in the reaching computation.

On each iteration, three types of computation occur in sequence:

1. The processor for each joint computes a new joint angle. To do this it must have access to the global desire vector, the line along which the vector acts, the position of the joint in space, and the moment of inertia of the distal portion of the system. The new joint angles can all be computed in parallel.

2. Given the new joint angles, the new positions of the joints in space can be computed by starting at the foot and following the chain of segment-lengths and joint-angles. Once the position of the tip is known the new desire vector can be computed.

3. Given the new joint positions, the new centers of gravity and hence the new angular inertias of the distal portions about each joint can be computed by starting at the tip and adding in the segments one at a time.

Provided k_r is sufficiently small, the combined effect on the desire vector of all the separate joint increments will be approximately the same as the sum of the effects of each increment by itself. All of the interactions (i.e. cases in which changes in one joint angle alter the way in which the desire vector is affected by changes in another joint angle) are mediated by the process of updating the positions of the joints in space, because the angle at one joint determines the positions of more distal joints and hence it determines the magnitude of the torque exerted by the desire vector about those joints. (The torque is a measure of the amount the desire vector would be changed by incrementing the joint angle).

This simulation glosses over all the complexities of the dynamics of a real physical system. It is actually a faithful simulation of a bizarre system in which inertial forces are negligible compared with viscous ones, but the viscosity at a joint is always set equal to the angular inertia about that joint.

BALANCING ALONE

If we ignore the reaching problem and just try to find a balanced configuration in which the center of gravity of the whole system is above the foot, a very similar computational scheme can be used. Instead of a global desire vector, there is a global measure of the extent to which the system is out of balance. This is simply the horizontal distance of the center of gravity from a vertical line through the center of the foot. Each joint angle can be incremented, in parallel, so as to reduce this quantity and then the new joint positions and new centers of gravity can be computed just as in the reaching algorithm.

Each joint controls the position of the center of gravity of the segments that are distal to it. The extent to which an angular increment to the j'th joint restores balance is determined by the mass, M_j, of the distal portion and also by the horizontal distance that its center of gravity moves. The horizontal distance moved is proportional to the vertical distance, V_j, between the joint and the center of gravity. So to help restore balance, the angular increment at the j'th joint is given by:

$$\Delta A_j = k_b . V_j . M_j / I_j \tag{4}$$

where k_b is a constant and I_j is its moment of inertia of the distal portion about the j'th joint.

COMBINING REACHING WITH BALANCING

To find a configuration that satisfies both the reaching and balancing goals simultaneously, the increments required for reaching and the increments required for balancing can simply be added together on each iteration. The relative values of k_r and k_b determine the relative importance of satisfying the two goals. If these values are set appropriately, joints near the foot are primarily influenced by the balancing goal because they control a large distal mass (M_j in equation 4), whereas joints near the tip are mainly affected by the reaching goal. Figure 1 shows a typical sequence of configurations generated by this simple parallel algorithm.

INADEQUACIES OF THE MODEL

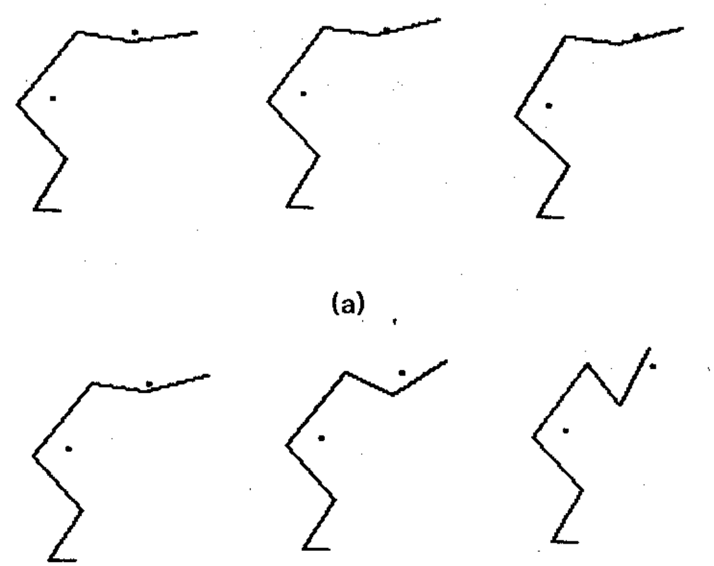
There are many criticisms of this simple computational model:

1. It gets stuck at local optimum, and so it can fail to find a suitable final configuration even though one exists. This is an important limitation of this kind of iterative parallel computation and it suggests that any computation of this type would need to be combined with more qualitative, schematic knowledge which could specify, very approximately, which region should be searched in the space of possible configurations. The use of schematic knowledge resembles table look-up, but it differs because the iterative computation used to compute the precise details of the final configuration is more powerful than simple linear interpolation between table entries.

2. It does not take obstacles into consideration. Obstacle avoidance requires an understanding of the space occupied by the body, and it also requires a more global view of the trajectory. The kind of myopic computation in which the current configuration is gradually changed into the desired one cannot deal with obstacles properly because it cannot see far enough into the future to avoid cul de sacs. An altogether different style of parallel computation may be required to avoid obstacles.

3. It is not parallel enough. Steps 2 and 3 in the computation seem to be inherently sequential. Actually, at the cost of some extra computation, they can be made parallel. If, for example, the orientations of the segments relative to the global frame of reference are stored and updated at the same time as the joint angles, it is possible to compute the new position of a joint in space simply by adding together the vectors for all the more proximal segments. This addition can be performed in parallel. The extra cost is that the same vectors get added several different times in computing the positions of different joints, and when a joint processor decides to update its joint angle it must also increment the global orientations of all the segments distal to it. This destroys the purely local communication structure in which each joint-processor only communicates with the processors for adjacent joints.

4. It takes a large number of iterations in certain situations. A simple example is shown in Figure 2a. The problem is that the line of action of the desire vector passes almost exactly through the shoulder and elbow joint. This means that the torques about these joints are very small so the joint angles are only changed very slightly on each iteration. It is obvious to us that the way to reach the goal is to shorten the arm by flexing it, but there is no explicit representation of the length of the arm in the parallel computation and so this insight cannot be incorporated. The next section shows how the introduction of higher order variables that coordinate the activities of several neighboring joints can overcome this problem and can dramatically reduce the number of iterations required.



(a)

(b)

Fig. 2. (a) shows how the simple iterative algorithm fails when the line of action of the "desire" vector passes close to the joints that need to be changed. The configuration is shown on every third iteration. (b) shows how the behaviour is improved by adding a synergy that controls the shoulder and elbow angles together and is invoked when the desire vector aligns with the direction from the tip to the shoulder. The configuration is shown after every iteration, so a single iteration with the synergy produces more progress than six iterations without it.

SYNERGIES

Changes in the angle of the elbow change the length of the arm ,but they also have side effects. They make the hand move along a circular arc centered on the elbow, and they change the hand's orientation in space. To make the hand move in a straight line away from the shoulder that cancel out the side-effects of the changes at the elbow. If the upper and lower arm segments are the same length, this can be done by increasing the elbow angle by 2α and decreasing the shoulder and wrist angles by α.

In cases like the one shown in Figure 2a, the iterative computation can discover the need for compensating changes at the shoulder. It first changes the elbow angle, and this changes the line of action of the desire vector so that it no longer passes directly through the shoulder. The desire vector then generates a torque about the shoulder which causes a change that compensates for the side-effect of the change in the elbow. However, this method of generating side-effects and then figuring out how to get rid of them is not nearly as efficient as avoiding unwanted side-effects by using "synergies" - combinations of changes whose side effects cancel.

Synergies are easy to incorporate within the parallel iterative framework. In addition to the processors for the individual joints, we simply add a processor for each synergy. This processor knows about the combined effect of a number of changes, and it is invoked whenever that effect is desired. When it is invoked it attempts to change the joints it controls in the appropriate way. Since a given joint may be under the control of several synergies as well as having its own processor, some principled way of combining different influences is required. As with the coordination of reaching and balancing, the correct combination rule is simply to add together the changes requested by the various synergies and by the joint processor itself.

Figure 2b shows how reaching is improved by adding a synergy that controls arm length by coordinating changes at the shoulder and elbow. The synergy is invoked by the degree of alignment between the desire vector and the vector from the shoulder to the distal end of the arm. If the angle between these two vectors is ϕ, the changes requested by the synergy on the current iteration are:

$$\Delta A_{elbow} = 2.k_s.\cos(\phi).I$$
$$\Delta A_{shoulder} = -k_s.\cos(\phi).I$$

where k_s is a constant that determines the relative importance of synergies as compared with the primitive changes computed by the individual joint processors, and I is the effective inertia.

ADDING NEW SYNERGIES

The simple additive rule for combining the changes requested by the various synergies makes it very easy to add new ones. They can just be thrown into the pool of existing synergies. Provided each synergy has its own processor, and provided the basic joint processors can perform the necessary additions of the requested increments in parallel, additional synergies do not increase the time required per iteration.

THE FUNCTION OF SYNERGIES

The use of higher order synergies to speed up a parallel iterative search is quite different from the use of higher order schemas to eliminate surplus degrees of freedom by imposing constraints on lower level variables. In the parallel iterative scheme, there may well be a hierarchy of synergies, each influencing the values of lower level ones, but every synergy in the hierarchy has access to the global goals (e.g. the desire vector and the measure of imbalance). Because surplus degrees of freedom do not present a problem to the parallel computation, there is no need to place each lower order variable strictly under the control of a single higher order one. All the processors for all the different levels of variable can compute in parallel, and every synergy can be activated to the extent that it helps reduce the difference between the current state of the system and the desired final state.

5. THE ROLE OF FEEDBACK IN MODEL-BUILDING

In the preceding sections I have described some computational advances that help to solve some of the problems that Bernstein raised. This final section contains meta-level comments on the role of feedback in model-building and on a new source of potential feedback for theories of human motor control.

At the start of chapter 4 there is a long discussion of feedback from historical and philosophical perspectives. Bernstein points out (as Gibson did in the field of perception) that most of the interesting issues in motor control were ignored for many years because of the laboratory study of artificially simple situations. This is a familiar modern view which is probably as uncontroversial now as it was radical at the time. His philosophical views, however, are more questionable.

Bernstein correctly points out that internal symbolic representations

can be objectively correct, and that they do not require the implicit internal observer assumed by idealists (see Dennett, 1978, pp 89-108 for an excellent recent discussion of this issue). Bernstein also points out that the development of a valid internal model is aided by two-way interactions with the domain being modelled. This is a point of great practical significance, but Bernstein tries to elevate the point to a philosophical principle that feedback of the effects of actions is *essential* for objective knowledge. This is a far more dubious propositionel. If it is true astronomers are out of luck.

Returning to the practical significance of feedback, recent technological developments have created the potential for a new source of feedback about the adequacy of theories of motor control. Progress in robotics clearly yields insights that supplement the experimental data on biological motor control, but there is always the worry that many problems in robotics may be the result of using conventional digital computers and may have little relevance to biological computation. The fundamental differences between the architecture of a general purpose computer and the architecture of the brain have led some people to suspect that computation may work in very different ways in these two types of machine (Von Neumann, 1958; Hinton & Anderson, 1981). Even though general purpose computers can be made to simulate any other kind of computer, there is still the suspicion that the sensible way to do a particular computational task in a conventional computer may be quite unlike the sensible way to do it in the brain.

Until recently, computers were very expensive to design and very expensive to build. This meant that they had to be general purpose, because the cost of designing and building dedicated hardware for a specific task was prohibitive. It was much more cost-effective to use a general purpose computer which was tailored to a specific task by its program. Now, however, very large scale integration makes computers cheap to build, and advanced computer aided design makes it relatively easy to design novel hardware structures. This opens up the possibility of tailoring the hardware to specific tasks, and thus avoiding the sequential bottleneck that is imposed by insisting on the architecture of a general purpose machine. It is now possible to design and produce a chip with hundreds of simple processors all computing in parallel and all communicating directly with specific other processors. The sequential execution of stored instructions is no longer necessary, and it may be possible, using dedicated hardware, to perform computational tasks hundreds or thousands of times faster by making good use of the parallelism (Mead and Lewicki, 1982). There is just one snag. There are very few good ideas about how to organize parallel computations using a hard-wired network of local processors each of which is much simpler than a general purpose computer. Perhaps this will be an area where theories of human motor coordination can suggest practical hardware designs, and where experience with these designs can provide useful feedback.

ACKNOWLEDGEMENTS

I thank Peter Greene for helpful comments on the manuscript. Many people helped me formulate the ideas presented here. They include Paul Smolensky, Don Norman, Dave Rumelhart and Don Gentner in the skills group at the University of California, San Diego; Tim Shallice and Alan Wing of the Applied Psychology Unit in Cambridge; and Marc Raibert and Scott Fahlman at Carnegie-Mellon University.

REFERENCES

Adams, J.A. Issues for a closed loop theory of motor learning. In, G.E. Stelmach (Ed.), Motor Control: Issues and Trends. New York: Academic Press, 1976.

Asratyan, D.G. & Feldman, A.G. Functional tuning of the nervous system with control of movement or maintenence of a steady posture. 1. Mechanographic analysis of the work of the joint on execution of a postural task. Biophysics, 1965, 10, 925-935.

Benati, M., Gaglio, S., Morasso, P., Tagliasco, V. & Zaccaria Z. Anthropomorphic Robotics. 1. Representing Mechanical Complexity. Biological Cybernetics, 1980, 38, 125-140.

Benati, M., Gaglio, S., Morasso, Pl, Tagliasco, V. & Zaccaria, R. Anthropomorphic Robotics, 2. Analysis of manipulator dynamics and the output motor impedance. Biological Cybernetics, 1980, 38, 141-150.

Bizzi, E., Accornero, N., Chapple, W. & Hogan, N. Arm trajectory formation in monkeys. Experimental Brain Research, 1982, 40, 139-143.

Dennett, D.O. Brainstorms: Philosophical essays on mind and psychology. New York: Bradford Books, 1978.

Greene, P.H. Problems of organization of motor systems. In, R. Rosen and F.M. Snell (Eds.), Progress in theoretical biology, Volume 2. New York: Academic Press, 1972.

Hinton, G.E. & Anderson, J.A. Parallel models of associative memory. Hillsdale, NJ: Erlbaum, 1981.

Hogan, N. Mechanical impedance control in assistive devices and manipulators. In, Proceedings of the Joint Automatic Control Conference, Volume 1. San Francisco, 1980.

Hogan, N. Control and coordination of voluntary arm movements. In, Proceedings of the American Control Conference. Arlington, Virginia, 1982.

Horn, B.K.P. What is delaying the manipulator revolution? Working paper 161. Cambridge, MA: Artificial Intelligence Laboratory, MIT, 1978.

Houk, J.C. Regulation of stiffness by skeletometer reflexes. Annual Review of Physiology, 1979, 41, 99-114.

Lozano-Perez, T. A language for automatic mechanical assembly. In, P.H. Winston and R.H. Brown (Eds.), Artificial Intelligence: An MIT Perspective, Volume 2. Cambridge MA: MIT press, 1979.

Luh, J.Y.S., Walker, M.W. & Paul, R.P.C. On-line computational scheme for mechanical manipulators. ASME Journal of Dynamic Systems, Measurement, and Control, 1980, 102, 69-76.

Mason, M.T. Compliance and force control for computer controlled manipulators. IEEE Transactions on Systems, Man, and Cybernetics SMC-11, 1981.

Mead, C.A. & Lewicki, G. Sillicon Compilers and Foundries will Usher in User-designed VLSI. Electronics, 1982, August 11, 107-111.

Polit, A. & Bizzi, E. Characteristics of motor programs underlying arm movements in monkeys. Journal of Neurophysiology, 1978, 42, 183-194.

Raibert, M.H. & Sutherland, I. Machines that walk. Scientific American, 1983, January, 44-53.

Saltzman, E. Levels of sensorimotor representation. Journal of Mathematical Psychology, 1979, 20, 91-163.

Turvey, M.T., Shaw, R.E. & Mace, W. Issues in the theory of action: Degrees of freedom, Coordinative structures and coalitions. In, J. Requin (Ed.), Attention and Performance VII. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1978.

Uicker, J.J. On the dynamic analysis of spatial linkages using 4x4 matrices. Ph.D. Thesis, Northwestern University, 1965.

Von Neumann, J. The computer and the brain. New Haven: Yale University Press, 1958.