Journal of Motor Behavior
1984, Vol 16, No 2, 171-194

Parallel Computations for Controlling an Arm

Geoffrey Hinton
Computer Science Department
Carnegie-Mellon University

ABSTRACT. In order to control a reaching movement of the arm and body, several different computational problems must be solved. Some parallel methods that could be implemented in networks of neuron-like processors are described. Each method solves a different part of the overall task. First, a method is described for finding the torques necessary to follow a desired trajectory. The methods is more economical and more versatile than table look-up and requires very few sequential steps. Then a way of generating an internal representation of a desired trajectory is described. This method shows the trajectory one piece at a time by applying a large set of heuristic rules to a "motion blackboard" that represents the static and dynamic parameters of the state of the body at the current point in the trajectory. The computations are simplified by expressing the positions, orientations, and motions of parts of the body in terms of a single, non-accelerating, world-based frame of reference, rather than in terms of the joint-angles or an ego-centric frame based on the body itself.

SKILLED MOTOR CONTROL appears to require a considerable amount of computation. It is hard, for example, to compute how to move the arm and body so that the hand ends up in a desired location. It is even harder if balance must be maintained and obstacles must be avoided. Even if the desired movements can be computed it is hard to find the torques and forces required to cause them.

Work in robotics has suggested ways of solving some of these problems, but the solutions involve large amounts of sequential computation on a conventional digital computer. With a few notable exceptions (e.g., Raibert 1978; Benati, Gaglio, Morasso, Tagliasco, & Zaccari, 1980), the computational models produced in robotics and artificial intelligence seem inappropriate as psychological models because of the

I thank Marc Donner, Don Gentner, Peter Greene, Wynne Lee, Matt Mason, Don Norman, Dave Rumelhart, March Raibert, Tim Shallice, Paul Smolensky, Neil Swartz, Alan Wing, and an anonymous referee for helpful discussions. The research was supported by a grant from the System Development Foundation. Mail correspondence to Prof G. Hinton, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213

long sequences of accurate numerical operations that they require. The brain does not appear to be well suited to such computations (Von Neumann, 1958). However, considerable understanding of the computational problems has been gained by trying to build robots, and if this understanding of the task can be combined with a more appropriate form of computation, it may lead to more plausible psychological models.

This paper explores ways of performing computations in systems composed of large numbers of simple, slow, neuron-like processors each of which is connected to many others. The approach is theoretical and computational rather than empirical. The aim is to discover how computations that appear to require considerable sequential depth can be broken down into fragments which can all be performed in parallel. Two aspects of motor control are considered in detail. The first task is to compute the torques needed at the joints to make the arm and body follow a desired trajectory, assuming that the desired trajectory has already been decided. This is known in robotics as the *inverse dynamics* problem. It is hard because the torque required to produce a desired angular acceleration at a given joint depends on the angles and angular velocities of all the other joints. For example, the torque required to bend the elbow depends on the angular velocity at the shoulder because rotation about the shoulder causes centripetal forces in the lower-arm and hand. Also the behavior of the wrist-joint affects the required torque at the elbow, because torques applied by the lower-arm on the hand are balanced by equal and opposite torques applied by the hand on the lower-arm. These reactive interactions appear to be so complex that it is tempting to look for a method of motor control that does not require the torques to be computed. However, there are good reasons for wanting to know the torques, even in systems that make full use of the length-tension properties of muscles, and it is actually quite easy to compute the reactive interactions. Provided the appropriate frame of reference is used, the torques at each joint can be computed in parallel.

The second task is to determine the trajectory that is required to reach out for an object while standing up. This involves choosing a trajectory for the arm that gets the hand to the desired location whilst keeping the center of gravity above the foot. The task is difficult because several goals must be satisfied simultaneously and many surplus degrees of freedom must be controlled. A suitable style of computation is iterative approximation using separate parallel processors for each degree of freedom. Each processor has access to the "motion blackboard" which is an internal data-structure that contains a representation of the current state of the body and a few global measures of the difference between the current state and the desired final state. On each iteration, each processor suggests how to change its own degree of freedom so as to reduce the global difference measures. An interesting feature of this kind of iterative computation is that the number of iterations required can be greatly reduced by adding processors that coordinate several

joints at once. These extra processors do not dominate the processors for the individual degrees of freedom as they would in a strictly hierarchical system. Instead, many different processors simultaneously try to influence each degree of freedom and the net result is the sum of all their effects.

A: FOLLOWING DESIRED TRAJECTORIES

Endpoint-setting

A simple way to ensure that the arm adopts some desired final configuration is to set the length-tension characteristics of all the muscles in such a way that the opposing torques exerted by agonist and antagonist muscles at each joint are only in equilibrium when the arm is in the desired configuration. This method of achieving a configuration does not require any computation of the trajectory or even any knowledge of the current configuration of the arm. So long as the arm is not in the desired final configuration, the opposing muscles at a joint will not be exerting equal and opposite torques, and so the arm will move (Asatryan & Feldman, 1965).

The idea that desired final configurations might be reached by using the length-tension characteristics of muscles to set end-points is ingenious, but it cannot possibly be correct, because it does not allow the trajectory to be controlled. There are many reasons why people or robots need to be able to control the spatio-temporal trajectories of their arms rather than merely achieving static final configurations. To avoid obstacles the arm must often take a circuitous route to its final configuration. To hit a tennis ball the arm must have the right configuration and the right velocities at the right time. To throw a basketball, the direction and magnitude of the velocity vector at the time of release must be precisely controlled. It is conceivable that people use quite different control mechanisms for tasks as different as throwing and reaching around obstacles, but it would be more elegant to use a general mechanism for controlling trajectories.

Using length-tension characteristics for position-servoing

The simplest way to follow a desired trajectory is to use independent position-servos at the joints. For each joint, the desired trajectory specifies the desired angle at each moment and this angle can be used as the reference setting for the servo. Any deviation from the desired angle then generates a compensating torque. One way of implementing the servos is to set the length-tension functions of the agonist and antagonist muscles so that the desired joint-angle is the equilibrium angle at which the torques exerted by the opposing muscles are exactly balanced. Once the length-tension characteristics have been set, a deviation from the equilibrium angle reduces the length and hence the tension of the muscles pulling in the direction of the deviation and increases the tension in the opposing muscles. There is thus a net torque back towards the equilibrium angle.

Following trajectories by dynamically changing the length-tension characteristics of the muscles is an obvious extension of the mass-spring model of endpoint-setting. It is equivalent to treating each momentary configuration within the trajectory as a temporarily desired final configuration. Like the simple mass-spring model, it has the attractive feature that the feedback term is generated by the physics. Deviations from the desired joint-angle cause a restoring torque because of the physical properties of the muscles, and so there is no need to wait for neural circuits to detect the deviation and generate a compensating neural signal. This "instant feedback" property is highly desirable because it avoids the oscillations that occur with servos that have long delay times.

Unfortunately, this simple extension of the mass-spring model fails because the system of independent servos that it implements is inadequate for trajectory control (Horn, 1978). The problem is that the torques are generated by the differences between the desired and actual joint-angles. So to achieve the high torques needed for rapid movements there must be large differences (i.e., large errors), which means that the trajectories cannot be followed accurately. The only way of saving the simple servo method is to use very high gains. (The gain is the ratio of the torque to the error.) This is what is done in robot arms that use multiple independent servos, but it is inappropriate for humans, because the use of high gains removes the ability to independently control the compliance of the system (i.e., how it responds to external forces that are applied to it). Compliance control is very important for skilled manipulation (Mason, 1981), and we cannot afford to sacrifice it for the sake of accurate trajectory control.

The real problem with the pure servo approach is that it treats torques as entirely unknown quantities that must be generated by looking at the error term. This is the only possible way to deal with genuinely unpredictable quantities, but it is not the best way to handle quantities that can be predicted. It is a well established principle in control theory that if there is any kind of internal model of the dynamics of the system, even a very approximate one, feed-forward terms can be computed ahead of time, and feedback terms can be relegated to their appropriate role which is to cope with unpredictable events and with the discrepancies between the internal model and reality.

Using length-tension characteristics to implement pre-computed torques

Let us suppose that for a ballistic reaching task, a desired trajectory has been pre-computed precisely, but that the torques required to follow it have been computed using a somewhat inaccurate model of the dynamics of the system. What is then needed is a way of combining the pre-computed torques with a low-gain position servo. At first sight this appears to require an implementation with two components, one for implementing the precomputed torques, and another for the servo

mechanism. However, if the precomputed torques are implemented by setting the length-tension characteristics of the muscles, the servo component of the combined model is an inevitable consequence. Given a precomputed torque, a pair of length-tension functions must be found to generate it (assuming, for simplicity, that only two opposing muscles are involved). Now, the torque generated by the length-tension functions of opposing muscles depends on the joint-angle. So to choose an appropriate pair of functions, the anticipated value of this angle must be taken into account. If the actual trajectory then follows the desired one, the actual torque generated by the pair of length-tension functions will be precisely the precomputed one. If, however, the actual trajectory lags behind the desired one, the very same pair of length-tension functions will generate a different torque because the joint angle is different. The actual torque generated can thus be viewed as the sum of the precomputed torque and a compensating torque that is proportional to the difference between the actual joint-angle and the currently desired angle.

This is a much more interesting extension of the mass-spring model than the previous one in which a trajectory is treated as a mere sequence of final configurations. It retains the idea that the feedback is generated by the physics, but it abandons the anti-computational idea that there is no internal model of the dynamics.

Much of the evidence in favor of the simple endpoint-setting model or the position-servo model of trajectory control also supports this more sophisticated model. For a task like reaching, the terminal settings of the length-tension functions are the same in all three models, so experiments in which an endpoint is reached despite unexpected temporary loads (Schmidt & McGown, 1980) or lack of knowledge of the starting point (Polit & Bizzi, 1978) do not discriminate between the theories. The real test is to measure the length-tension functions during an arm movement. The model described above predicts that they should be set so as to generate just the right torques to move the arm through a smooth trajectory. This is just what Bizzi, Accornero, Chapple, and Hogan (1982) have found in an elegant experiment using deafferented monkeys. Bizzi et al. interpret their results in terms of a moving set point that leads the desired position during acceleration and lags behind it during deceleration. This model is mathematically equivalent to the idea that length-tension function provide automatic feedback when they are used to implement pre-computed torques.

B: COMPUTING TORQUES

It is not easy to compute the torques required to make a complex arm follow a desired trajectory, because of the interactions between different joints. The torque required at one joint depends on the angles and angular velocities of all the other joints. Raibert (1978) has shown that for a simple arm these complexities can be finessed by using a massive memory to hold lots of simple equations each of which is

tailored to very specific conditions. For any particular combination of joint-angles and joint angular velocities, the relationship between the torques applied at the joints and the angular accelerations is relatively simple. So a desired trajectory can be followed fairly accurately by sampling at closely spaced times and using the joint-angles and angular velocities at each moment to look up the local equations that govern this specific situation. The equations can then be used to compute the torques required to generate the desired angular accelerations.

An interesting feature of Raibert's model is that the enormous number of local equations that are stored in the table do not need to be computed. They are simple enough to be learned by observation of the dynamic behavior of the system. For each combination of joint-angles and joint angular velocities, it is only necessary to observe how the system responds to a few randomly chosen combinations of applied torques in order to derive the local equation that relates *any* combination of applied torques to the resulting angular accelerations.

If one is concerned about the relevance of work in artifical intelligence to human motor control, the use of a massive memory to avoid complex computations seems like a step in the right direction, and the fact that the contents of the memory can be easily learned from experience is an added bonus. However, Raibert's model may be too expensive even for the brain. For an arm with six degrees of freedom, the memory would need to cover a twelve-dimensional space of local equations, which is very costly even if the number of distinctions per dimension is kept small. Also, if the arm picks up an object all the equations change in a way that depends on the weight and size of the object (Benati, Gaglio, Morasso, Tagliasco, & Zaccari, 1980a, 1980b). This is a very serious problem because it seems unlikely that a different set of equations is stored for each object that might be held

Computing torques isn't very hard

The main motivation for Raibert's table look-up model was that it allowed torques to be generated in real time during a movement. An alternative way to meet the real-time constraint is to use parallel computation. Given a parallel method with very little sequential depth, the torques could be computed rapidly even in a neural network whose individual processors are relatively slow. Ideally, it should also be possible to learn parameter values like the masses and lengths that have to be used in the computation.

The parallel method that is presented here is similar to a recent sequential algorithm (Luh, Walker, & Paul, 1980). They have shown that the computation of the torques necessary to achieve particular angular accelerations at the joints can be done in a number of steps that is simply proportional to the number of joints in the arm. The mathematics of their method is fairly complex, but the physics on which it is based is relatively straightforward, because all the kinematic and dynamic information is expressed relative to a single, non-accelerating frame of

reference. This eliminates complicating factors such as Coriolis forces which only arise when accelerating frames of reference are used. The natural coordinate system provided by the joints themselves involves accelerating frames because angular velocities at one joint cause angular accelerations at all more distal joints.

The algorithm used by Luh et al. starts by converting information about positions, velocities, and accelerations from the natural joint coordinate system to a single "world-based" reference frame. This is done by starting at the proximal end of the arm (whose relation to the world is fixed), and working outwards. If the motion of segment 1 relative to the global reference frame has been computed, and if the motion of segment 2 relative to segment 1 is known it is straightforward to compute the motion of segment 2 relative to the global frame. Thus the motions of all the segments in turn can be computed. Once this has been done it is possible to solve for the forces and torques required to cause this motion by starting with the most *distal* segment and working inwards. Assuming that externally applied and gravitational forces are known, the desired linear and angular acceleration of the most distal segment provides enough information to solve for the forces and torques between it and the penultimate segment. Once these forces and torques are known, the desired accelerations of the penultimate segment can be used to solve for the forces and torques at its proximal end, and so on. The equations of motion are simple because a single non-accelerating frame is used.

The first stage of the Luh, Walker, and Paul algorithm can be omitted if the information about the required trajectory is already expressed relative to a single world-based frame. This possibility is described in detail later.

The stage in which the equations of motion are solved for one segment at a time *appears* to be inherently sequential. It looks as though the forces and torques at a joint can only be computed *after* the forces and torques have been found for its more distal neighbor. Fortunately, a simple physical argument shows that this is not so. The entire portion of the system that is distal to a particular joint has a total angular momentum about the point in space currently occupied by that joint (see Figure 1). By considering how this angular momentum is changing, it is possible to solve for the torque at the joint without knowing the torques at more distal joints.

The angular momentum of the entire distal portion about a joint is not affected by internally developed torques exerted at more distal joints. It can only be changed by torques at the joint itself, or by external forces or torques applied to the more distal segments. So if we compute the rate of change of angular momentum about the joint and we make allowance for torques exerted by gravity and external forces, the remaining rate of change of angular momentum must be due to the torque exerted at the joint. The linear forces at the joint do not need to be computed because they cannot change the angular momentum about the point in space currently occupied by the joint.

I have checked the physical argument given above by implementing a 2-D version of the parallel algorithm and checking that it gives the same answers as a direct implementation of the Luh, Walker, and Paul method.

The fact that the torques can be computed in parallel allows a method with much less sequential depth than the algorithm of Luh et al but there is a price to pay in terms of the number of primitive computational operations that have to be performed. To compute the rate of change of angular momentum about P, the point in space currently occupied by a joint, it is necessary to add together the contributions from all the more distal segments. Each of these segments contributes in two ways. First, it has a rate of change of angular momentum about the point currently occupied by its own center of gravity, and all these contributions must be summed for all the distal segments. Second, it has a rate of change of linear momentum and the contribution of this vector to the rate of change of angular momentum about P depends on the relationship between the vector and P, as shown in Figure 1.

Given a fast sequential computer, the cheap way to add all these contributions together is to wait until the sum has been computed for the immediately distal neighbor and then to simply add in the effect of the intermediate segment (after allowing for the fact that the angular momentum is being computed about a different point). However, this is a sequential method and to avoid the delays it entails we need direct communication of the relevant quantities from all the more distal segments and a way of adding together many contributions in one step. The price of a truly parallel method is therefore a number of connections that is proportional to the square of the number of joints, and adders that can combine many inputs at once. Neither of these seem implausible for the brain.

Once the necessary torque at a joint has been computed, it can be resolved into components that align with degrees of rotational freedom at the joint and components that are orthogonal to degrees of rotational freedom. The former components must be generated by the relevant muscles, and the later components will be generated automatically by the mechanical constraints.

Given that the computation of torques is not as difficult as it at first appeared, it may well be possible to run an internal simulation of the dynamics. It is relatively easy to incorporate rigid objects that the hand might hold into the model because these act just like an extra segment, so if their physical parameters are known, their effect on the dynamics can be computed in just the same way as for a segment. The dependence of the dynamics on these objects then ceases to be a major problem.

The use of a single, non-accelerating, world-based frame of reference seems to be essential for this parallel approach to the inverse dynamics. It makes it possible, for example, to use simple addition to combine the rates of change of angular momentum of all the distal segments when computing the required torque at a joint. These quantities can only be

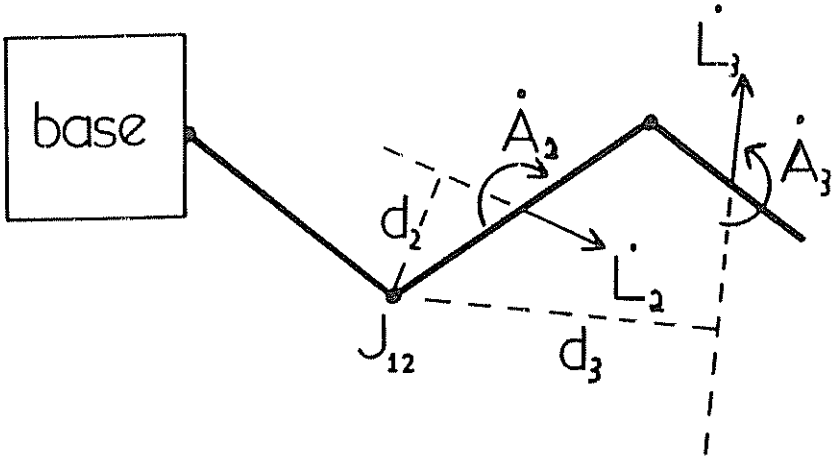


Fig. 1—This shows the how to compute the required torque, T_{12}, at the joint between the first and second segments of a three segment arm attached to a stationary base. The torque depends on the desired rates of change of the angular and linear momenta of the more distal segments. These quantities can be summed without having to first compute the torques at more distal joints. The rates of change of the linear momentum of distal segments must be multiplied by the perpendicular distance from the joint to the line of action of the vector before they are summed. Thus the equation for the torque is:

$$T_{12} = \dot{A}_2^* + d_2\dot{L}_2^* + \dot{A}_3^* + d_3\dot{L}_3^*$$

added if they are expressed with respect to a common reference frame. If a sequential method is used, it is not so important to use a world-based frame because a coordinate transformation can be performed at each step in the sequence, and this allows information about remote segments that use different frames to be combined economically. Indeed, Hollerbach (1982) points out that recursive, *sequential* methods of computing the inverse dynamics are more or less equally efficient whichever coordinate system is used.

The whole idea of having a *correct* internal model of the dynamics is, of course, rather speculative when it is applied to biological systems. If there is a simpler way of achieving a good approximation evolution may well have discovered it. Table look-up appears to be too cumbersome, but there are other alternatives that cannot be dismissed so easily. Greene (1982), for example, has described a way of using only 216 coefficients to approximate the behavior of a simple arm, and if this approach can be extended to cope with more degrees of freedom, it will cast serious doubt on the need for a physically correct internal model.

C: WORLD-BASED FRAMES

The method of computing torques described in the previous section presupposes that the desired configuration and motion of the arm are known in terms of a Newtonian or "world-based" frame of reference. It is not enough to just know the desired angles, angular velocities, and angular accelerations of the joints—these quantities must all be converted into the desired motions of the segments of the arm relative to a single, common, non-accelerating frame of reference.

Choosing a world-based frame

To choose a world-based frame, it is necessary to choose a position and velocity for the origin and orientations for the axes. For each particular action, some choices of an origin will be more appropriate than others. In reaching for a stationary object, for example, it would be sensible to use the position of the object to define the origin of the world-based frame. Then even if one had to walk towards the object to pick it up there would be a fixed frame for the computation. In catching a falling object, however, it would be a mistake to tie the origin to the object because the frame of reference would then accelerate with the object which would greatly complicate the dynamics.

Generally, it makes sense to use the direction of gravity in defining the "vertical" orientation of the world-based frame. Any scheme which attempts to use a movable object, like the body, to define orientations leads to a poor representation that requires frequent updating. Relative to a "bodybased" (i.e., ego-centric) frame, stationary objects keep moving around and rotating and the direction of gravity keeps changing as the orientation of the body varies.

Converting between reference frames

Since kinesthetic information and the mechanical constraints imposed by the joints are naturally expressed in terms of the joint-based coordinate system, it is necessary to be able to convert between reference frames. In a conventional computer, the conversion would be handled by two sets of procedures. The "forward kinematics" procedures would convert information from the joint based system to the world-based one, and the "inverse kinematics" would convert the other way. Provided the state of the arm is fully specified in terms of one frame, it is relatively easy to convert to the other one. The difficulties arise when the state of the arm is only partially specified. If, for example, the desired position and orientation of the hand and of the feet are known with respect to the world-based frame, it is non-trivial to discover a set of joint-angles for the rest of the body that allow these world-based constraints to be satisfied. Even with the hand and feet fixed in space, the body has many residual degrees of freedom and so there are generally many possible solutions, and which one is chosen must depend on factors other than the kinematics.

D: TRAJECTORY FORMATION

In current robotics work, reaching movements are typically computed in several phases. First a desired path for the hand is chosen, then this is converted into a desired sequence of joint-angles (Brady, 1982). The problem of surplus degrees of freedom is typically avoided by using an arm that has only six degrees of freedom and can therefore only achieve a particular position and orientation of the hand in one way. Some attempts have been made to take the dynamics into account when choosing the desired path, but this is so computationally expensive on a conventional computer that the path is generally chosen without any detailed consideration of whether it allows a dynamically good trajectory.

This section describes an alternative approach which does not first choose a path for the hand and then find a trajectory that implements it. Instead, the trajectory is generated one piece at a time by applying a large set of heuristic rules to the current state of the arm and the current goals of the system. Each applicable rule suggests how the joint-angles should be changed in order to help satisfy the goals. The actual changes are the combined effects of all the separate suggestions. This allows the system to satisfy several goals at once, like reaching out to a target and maintaing its balance. If the state of the arm included dynamic information, the system should be able to choose dynamically good trajectories. Using this approach the trajectory can be formed in real time and does not need to be stored.

To illustrate how this style of computation can be used for trajectory formation, I have implemented a very simple version of the general approach. The program works in two-dimensions rather than three and glosses over all the complexities of the dynamics, but it does show how many separate heuristic rules can work together to generate a reaching movement whilst maintaining balance. It also demonstrates the advantages of using "synergies"—combinations of primitive movements whose side-effects cancel out. I first describe how the program works and then discuss how a more elaborate version of the same computational style might cope with the dynamics.

A Simplified Trajectory-Formation Task

The task is simply to generate a trajectory that allows a standing stick-figure to reach out to a target from any starting configuration without loosing its balance (see Figure 2). The figure only has one arm and one leg, and its foot always remains fixed. The tip of the arm is the *distal* end of the stick-figure and the foot is the *proximal* end. The joint-angles all have maximum and minimum limits, and the segments have masses that are roughly appropriate. The stick-figure is said to be in balance if the center of gravity is vertically above some part of the foot.

Reaching alone

Initially we shall ignore the problem of balance and focus on the pro-

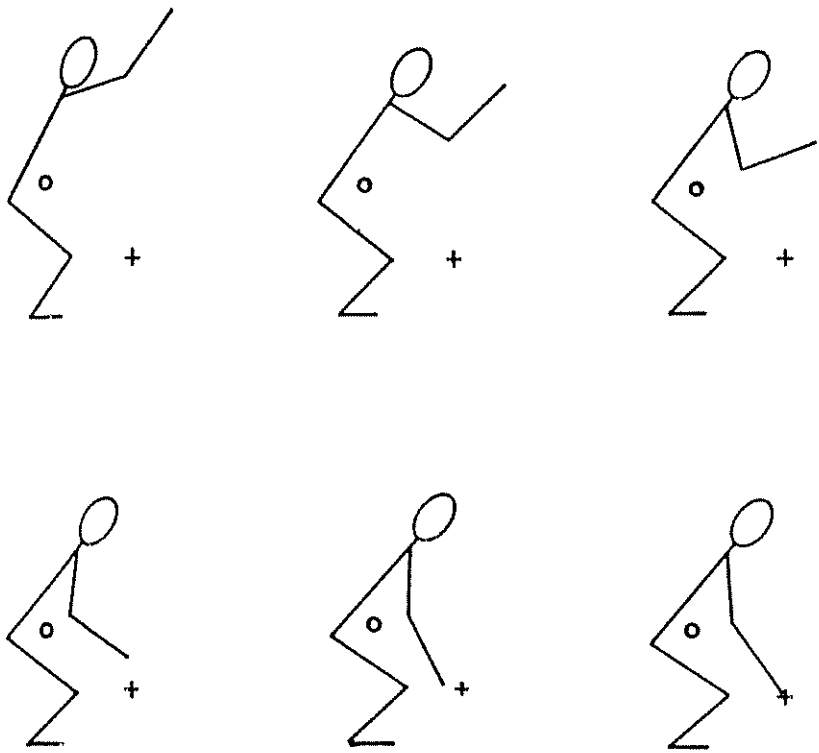


Fig. 2—This shows a sequence of configurations generated by the iterative algorithm. The small circle represents the center of gravity of the whole stick-figure, and the cross represents the goal to be reached. The "head" was not included in the simulation and therefore has zero mass. It has been included here solely to help the reader perceive the relationship between the line segments and a human figure. 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 cooefficients to be used without causing divergent oscillations. Extra rules which control several joints at once were used in this example. These extra rules are described later in the text.

blem of getting the tip of the arm to the target position. The difficulty is that there are five degrees of freedom in the body and the position of the target only provides two degrees of constraint, so it is impossible to "solve" for the trajectory. Even if we were to insist that the hand followed a straight line path to the target, there would still be many alternative trajectories for the whole body. However, a simple physical analogy suggests a way of performing the computation that is not hampered by surplus degrees of freedom. If we took a real pin-jointed stick-figure and connected the tip to the target with a rubber-band, the physics would "solve" the reaching problem. Perhaps we can simulate a simplified version of the physics.

Suppose that the joints are very viscous compared with the inertial mass of the stick-figure. The rubber-band (the "desire vector") then generates a torque about each joint that is given by:

$$T_i = d\, r_i$$

where d is the magnitude of the desire vector and r_i is the perpendicular distance from the i^{th} joint to the line along which the desire vector acts (see Figure 3). In each small time interval, each joint is incremented by a small amount that is proportional to the torque. To ensure that light segments move more easily than heavy ones, we also make the angular increment, $\Delta\alpha_i$, inversely proportional to the moment of inertia, I_i, of the distal portion of the system about the i^{th} joint:

$$\Delta\alpha_i = k_r T_i / I_i$$

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

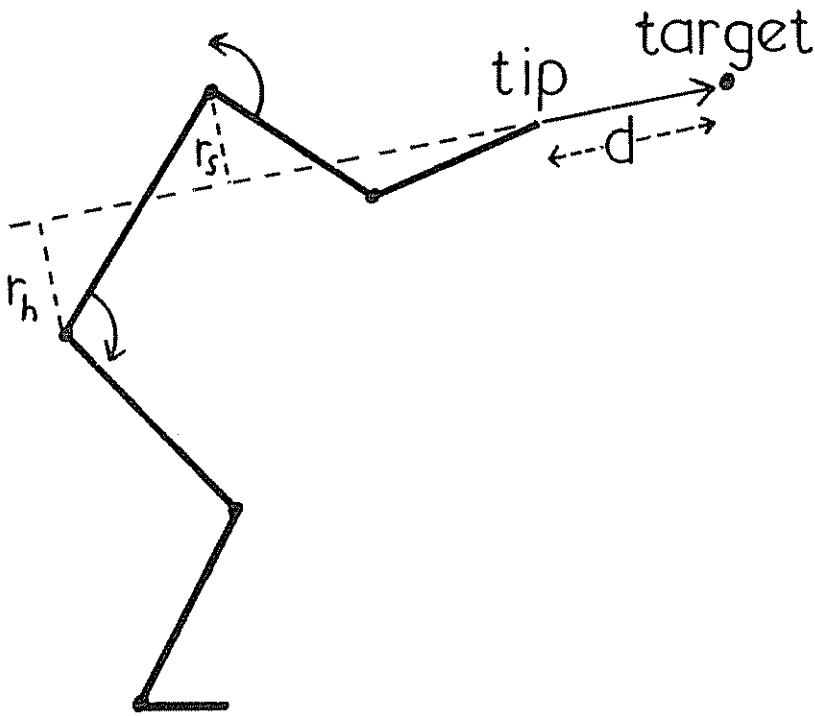


Fig. 3—This shows the fictional torques that would be exerted by the "desire vector" from the tip of the arm to the target. At each joint, the fictional torque is the product of the length of the desire vector, d, and the perpendicular distance from the desire vector to the joint. Thus the torque at the shoulder is $r_s d$ and the torque at the hip is $r_h d$. These are not the real torques that would be required to move a physical system. They are fictional torques that would be exerted by a rubber band stretched between the tip of the arm and the target. These fictional torques are used for computing the desired changes in the joint angles.

The Motion Blackboard

Repeated parallel iterations of Eq 1 will get the tip to the target and the trajectory will be reasonably sensible (but not optimal and not like a real human trajectory). To compute the fictional torques exerted by the desire vector, it is necessary to know where each joint is in space, so after each iteration, the joint positions must be updated. This can be done by using a data-structure called the "motion blackboard" which receives instructions about how to increment particular joint-angles, and automatically maintains consistent representations of joint-positions. In other words it ensures that the current internal representations of the joint angles and their positions in space are consistent with one another. The motion blackboard also maintains the centers of gravity of the various segments and it uses these to maintain the angular inertia about each joint of the whole portion of the system that is distal to that joint. (This quantity is needed in Eq 1.) Finally, the blackboard maintains the current "desire vector" from the tip of the arm to the target, since this is needed for computing the fictional torques

In a truly parallel system, the motion blackboard would maintain consistency using a parallel constraint-satisfaction method. In the actual program, however, the ability of the blackboard to maintain consistent representations is implemented by a set of serial procedures. The new joint-positions, for example, are computed by starting at the foot and following the chain of segment-lengths and joint-angles. The new centers of gravity are then computed by starting at the tip and working backwards, adding in the contribution of one segment at a time. Although these procedures appear to be inherently sequential, they can be made parallel at the cost of some extra computation, and some communication between parameters for non-adjacent segments. Suppose, for example, the blackboard maintains a vector for each segment, that represents the difference between its endpoints. The length of this vector is fixed, and its orientation is simply the sum of the more proximal joint-angles. So the new value of the vector can be found in parallel, provided there are parallel adders that can sum many joint-angles at once. Once the new vectors are known, the position of any joint can be found by adding together all the proximal vectors.

The rule for determining the increments in the joint-angles can be implemented by many separate parallel processes, one for each joint. Each process continually inspects the relevant variables in the blackboard, computes an increment, and tells the blackboard to update the joint-angle. This way of describing the operation of the program is intended to emphasize its underlying similarity to the Hearsay architecture (Erman, Hayes-Roth, Lesser, & Reddy, 1980) used in speech recognition. Many autonomous processes inspect a blackboard and decide how its contents should be changed. The blackboard itself maintains the consistency of its representations.

Balancing Alone

We shall ignore the interesting and complicated problems of dynamic balance and just focus on the problem of choosing combinations of joint-angles that keep the center of gravity above the foot. (This is not intended as a model of how balance actually occurs. It is just being used as a simple illustration of a computational style.)

The goal is to minimize the distance between the center of gravity and the vertical line through the middle of the foot. This can be done by incrementing each joint-angle, in parallel, so as to reduce this distance. The size of the increments is proportional to how much they help in achieving the goal. The j^{th} joint controls the position of the center of gravity of the whole portion of the system that is distal to it. The horizontal amount by which this center of gravity moves when the joint-angle changes is proportional to the *vertical* distance, V_j, between the joint and the center of gravity of the whole distal portion, and the effect on the whole body's center of gravity also depends on the mass, M_j, of the distal portion. So to help maintain static balance, the angular increment at the j^{th} joint is given by:

$$\Delta\alpha_j = k_b V_j M_j / I_j$$

where k_b is a constant that determines the importance of maintaining balance and I_j is the moment of inertia of the distal portion about the j^{th} joint. The intertia is included to ensure that the cost of moving the distal portion is taken into account (though this only provides a crude measure).

Combining Reaching with Balancing

Given two simultaneous goals, like reaching and maintaining static balance, different rules often make conflicting suggestions about how to change an individual joint-angle. These conflicts can be resolved by a very simple procedure that leads to a smooth compromise—all the suggested increments for a joint are just added together. The relative values of k_r and k_b determine the relative sizes of conflicting increments. If these coefficients are set appropriately, joints near the foot are primarily influenced by the balancing goal because they control a large distal mass, whereas joints near the tip are primarily governed by the reaching goal. Naturally, if there are goals which cannot be achieved simultaneously it is necessary to use some competitive mechanism to select one rather than blending together the increments that would achieve each goal separately.

The goals of reaching and balancing interact in a rather simple way in the computer simulation. Small angular increments are made to help reach the target, and these increments disturb the balance. As a result, further increments are made to help restore balance. Thus balancing is maintained by simply reacting to the effects of reaching. It is important to realize that all this goes on within an *internal model* that is being used

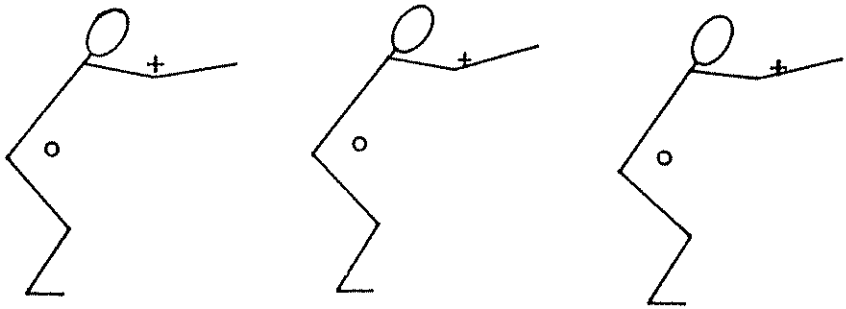
to construct a desired ballistic trajectory. The delay time in the reactive loop can therefore be much less than the time required to physically affect the body and then sense the effects. The distinction between the feedback time within the internal computation and the feedback time for real mechanical events is crucial for the plausibility of any incremental computation of the type proposed here.

Synergies

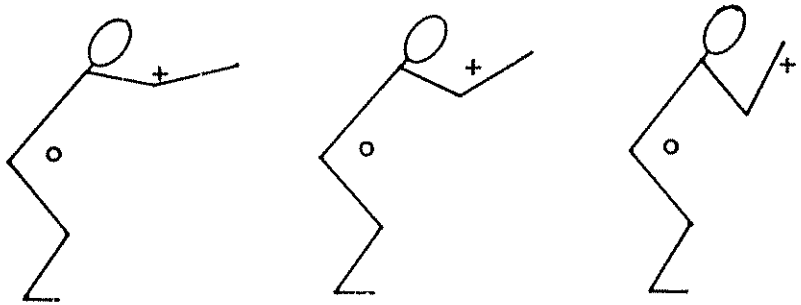
A serious drawback of this iterative style of computation is that it can require a large number of iterations in certain situations. The computation treats the individual increments as if they made *independent* contributions to the task of moving the tip towards the target. If the increments are small enough this is very nearly true. With large increments, however, changes in one joint-angle significantly alter the desire vector, and they also alter the way in which the position of the tip is affected by changes in other joint-angles. So the combined effect on the tip of all the increments is not simply the sum of the effects that each increment would have caused if it had been made alone.

This problem can be avoided by always using small increments. The interactions are then automatically handled correctly by the process of updating the desire vector and the positions of the joints in space. An increment at one joint changes the positions of all more distal joints. This changes the fictional torque that the desire vector exerts about those joints, and thus changes the subsequent increments that are chosen for them. This solution is simple but slow because it requires many sequential steps, with all the angles being changed by a small amount on each step.

An interesting alternative is to introduce more sophisticated rules that make allowance for the interactions. A change in one joint-angle typically causes the tip to make some progress towards the target but also some movement orthogonal to this direction. This "sideways" movement is the source of much of the undesirable interaction, because it changes the direction of the desire vector. It can be largely eliminated by using "synergies"—combinations of joint increments whose side-effects cancel each other out. (See Lee, 1984, for a discussion of the empirical evidence for synergies). Consider, for example, the configuration shown in Figure 4a. Changes at either the shoulder or elbow cause the tip to make a small movement in the direction of the target and a large sideways movement. Moreover, a change in one angle can reverse the direction in which the other angle must be changed in order to make the tip approach the target. If, however, a clockwise change of 2θ at the elbow is accompanied by an anticlockwise change of θ at the shoulder, the tip will move straight towards the target. Such synergies are easy to incorporate within the general style of computation described above. They simply require extra processes that "know about" the combined effect of several changes and are invoked whenever this effect is desired. In the actual program, the



(a)



(b)

Fig. 4—(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 behavior 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.

"arm-extension" synergy, for example, checks the blackboard to see how well the vector from the shoulder to the tip aligns with the desire vector. It then suggests changes that are proportional in size to the projection of the desire vector onto the vector from the shoulder to the tip. The suggested change at the shoulder is half as big and opposite in sense to the suggested change at the elbow.

Since a given joint may now be under the control of several different rules, we need a good way of combining their suggestions. As with the coordination of reaching and balancing, a sensible combination rule is to simply add together all the various suggested increments. The simplicity of the combination rule makes it very easy to add new synergies. There is no need to remove or inhibit existing rules. All that is required is access to the blackboard, a processor for the new rule, and a parallel adder within the background so that it can combine many suggestions at once. The use of simple addition as a method of resolving conflicts between several autonomous rules contrasts sharply with the conflict-resolution procedures that are generally used in more discrete domains like problem-solving. In these domains, a single rule wins and the others are suppressed.

The way synergies are used in this style of computation is quite different from a superficially similar idea in which higher-order constraints are used to eliminate surplus degrees of freedom (Greene, 1972; Turvey, Shaw, & Mace 1978). In the simplest version of that approach, constraints are introduced to create a "virtual body" which has fewer degrees of freedom than the real body. The idea is that the virtual body should be easier for central processes to control, particularly if there is a large repertoire of virtual bodies and one has already been selected that is appropriate for the task at hand. Commands to the virtual body are implemented by lower level processes that are built up by prolonged experience, but the central processes don't need to know about all the gory details.

With the blackboard approach, life is even easier for the central planning processes. Instead of selecting an appropriate virtual body and then sending high-level commands to it, the central processes merely put the desired goals on the blackboard. The autonomous trajectory formation rules then do the rest, leaving the central planning routines free to concentrate on what spatio-temporal goals to create. Since the style of computation has no difficulty with surplus degrees of freedom, there is no need to try to eliminate them. That is not what the synergies are for. They are there to reduce the number of iterations and to give better trajectories (especially when the dynamics must also be taken into account).

Getting stuck at local optima

If the initial configuration has the arm raised above the head and slightly behind it, and the target to be reached is at ground level behind the foot, the stick-figure will try to reach the target by bending over

backwards. The limits on the joint-angles make it impossible to reach the target this way so it gets stuck. The target can be reached perfectly well by rotating the arm forwards and squatting down, but this involves moving the tip *away* from the target in the initial phase. The existence of local optima in the space of possible configurations rules out this kind of local iterative computation as a *global* search method. However, it can still be useful when combined with more qualitative schematic knowledge that specifies some approximate intermediate points on the trajectory. Given these intermediate points, iterative computation can be used to generate a precise trajectory, so the schematic knowledge can be minimized.

E: GENERATING DYNAMICALLY GOOD TRAJECTORIES

There are many different criteria for what constitutes a good trajectory, and a motor control system should, ideally, be capable of optimizing any weighted combination of these criteria. For movements to be as economical and accurate as possible, each trajectory must be chosen so as to minimize quantities like the magnitude of the torques needed to follow it and the speed with which these torques must change. If the hand is holding an object it may also be desirable to minize the accelerations or jerks applied to this object. If the joints are to last, it may be desirable to minimize the mechanical stresses on them.

So far, I have avoided the problem of how to generate a trajectory which not only gets the hand to the target, but does so in a dynamically optimal or near optimal way. The computer simulation described above completely ignores this problem. Generating a dynamically good trajectory is much harder than just generating a smooth trajectory because it is generally impossible to decide whether the first part of a trajectory is *optimal* without considering the remainder of the trajectory, so the idea of generating a trajectory one piece at a time seems doomed, and it looks as though the "motion blackboard" approach to trajectory formation must be abandoned because it cannot handle the dynamics.

However, there is no good evidence that people generate *optimal* trajectories for complex movements. Indeed, when reaching around a barrier people appear to use trajectories that are composed of several separate sub-movements that are smoothly blended together. The velocity profiles typically have distinct bell-shaped curves for each submovement (Abend, Bizzi, & Morasso, 1982). This is just the kind of trajectory that would be produced if a "module" that knew about obstacle avoidance created some intermediate points or regions to be reached, and the trajectory was formed by starting with the first intermediate point as the target. As this point was approached, its goal could be faded out and the goal for the next intermediate point faded in. Naturally, the inertia of the arm itself would provide some of the smoothing.

The trajectories for simple one-joint movements are near optima with respect to minimizing jerk (Hogan, 1982), but this could be the result of learning heuristic rules that work well in simple cases. People have

years in which to explore the space of possible trajectories, and provided they are able to keep some kind of record of what they have learned, they should be able to discover which trajectories are graceful and which are awkward. Such discoveries are only useful if they can be stored in a way that allows them to be accessed and used appropriately in the future. A literal catalog of past movements would be too large to store and too specific—few movements are never repeated precisely. Past experience needs to be boiled down into a set of easily applied heuristics that can be used for constructing new, dynamically good trajectories. If the motion blackboard is expanded to include dynamic as well as static information, it should be possible to add more rules that inspect and modify this dynamic information and thus generate good (but not optimal) trajectories. The trajectory formation rules would encode knowledge about the spatio-temporal properties of desirable trajectories, but they would not have to specify the torques needed to to follow these trajectories, since the torques would be computed by the parallel procedures described earlier.

It would be possible to keep the trajectory formation rules quite separate from the procedures for computing the torques, but a complete separation seems wasteful because much of the dynamic information that is needed for trajectory formation is also needed for computing the torques required to follow the trajectory, if the model presented earlier is anything like correct. It would therefore be sensible to have a single motion blackboard containing quantities like the linear and angular momentum of various combinations of segments. Both the trajectory formation rules and the procedures for computing torques would inspect these quantities, but only the trajectory formation rules would change them.

Given access to the appropriate dynamic quantities, trajectory formation rules could be fairly simple. For example, the aim of a reaching movement is to have the tip of the arm at the target, the center of gravity above the foot, and no linear or angular momentum. It is fairly clear that if the tip is almost at the target but the arm has a lot of forwards linear momentum, the arm should be decelerated. Similarly, it is clear that if the center of gravity is above the foot, but the whole body has considerable clockwise angular momentum about the foot, something needs to be done to get rid of this angular momentum, preferably before the center of gravity is too far outside the region above the foot. I have not yet implemented rules that help to satisfy dynamic goals, but I can see no reason why the motion blackboard approach should not work as well as it does when only static variables are being considered and modified. Further work, however, is needed to substantiate this view. The quality of the resulting trajectories would, of course, depend on having good trajectory formation rules that worked smoothly together. These rules might take a long time to acquire and might be specific enough so that unpracticed trajectories were initially far from optimal, but this seems to be characteristic of human performance.

F: OBSTACLE AVOIDANCE

Obstacle avoidance is a hard problem. It requires a representation of the space occupied by the parts of the body and it also requires the whole trajectory to be considered at once. Feasible trajectories cannot be grown one piece at a time because the only way to avoid cul de sacs is to consider the feasibility of the end part of the trajectory while deciding on the beginning part. It is conceivable that some kind of backtracking search is performed, but it is unlikely that the brain has time to do much backtracking. It would be better to use a search technique that homes in on feasible trajectories without sequential exploration of the alternatives.

A feasible trajectory must satisfy two different kinds of constraint simultaneously. First, parts of the body must not occupy the same space as obstacles or each other. Second, the dispositions of the different parts of the body must satisfy the mechanical constraints imposed by the joints. The space occupancy constraints are most naturally expressed in terms of a frame of reference based on the world, and the mechanical constraints are most easily expressed by using the joint-angles as a coordinate system.

An interesting recent idea is to perform all the computations in "configuration space" whose dimensions are defined by the joint-angles (Lozano-Perez, 1982). Each point in configuration-space represents a particular combination of values for the joint-angles. Since an obstacle rules out certain configurations (ones that would involve parts of the body occupying the same space as the obstacle) it is possible to represent the obstacle as the set of all configurations of the body that it rules out. Each of these configurations is a point in configuration-space, so an obstacle corresponds to a forbidden region in configuration space. An obstacle-avoiding trajectory is a path from the point that represents the initial configuration to a point that represents a configuration which satisfies the task requirements.

The configuration space approach is mathematically elegant, but it is hard to see how it can be given a direct parallel implementation. The dimensionality of configuration space is equal to the number of degrees of freedom of the body, so it is impossible to explicitly represent all the zones in configuration space, and reasoning about feasible paths must be done in lower-dimensional projections of the space. An alternative scheme that is more amenable to parallel computation is needed.

It may well be a mistake to assume that the aim of the obstacle avoidance computation is to discover a single feasible trajectory. To keep the problem modular, it is helpful to ignore dynamics when solving the obstacle avoidance problem, but this means that any single trajectory that is chosen to avoid obstacles may not be very good dynamically. One way out of this dilemma is to allow the obstacle avoidance computation to return a whole swathe of similar trajectories all of which avoid the obstacles. The heuristics for choosing a

dynamically good trajectory could then grow a single trajectory, one fragment at a time, from within this swathe

SUMMARY

It is difficult to interpret the experimental data on motor control without having some idea of the kinds of computational mechanisms that might be involved. One powerful constraint on these mechanisms is that they have to be capable of performing the task. Another is that they have to be implemented in the brain which is a highly parallel computer consisting of millions of relatively slow processors that are richly interconnected. This paper has described some computational schemes that are designed to make good use of the brain's parallelism.

The simplest piece of the reaching task to be considered is the computation of the torques required to follow a particular trajectory. This can be performed in a relatively shallow network. The existence of such a method casts serious doubt on the need for table look-up schemes which avoid the computation by using a massive memory, though the problem of learning the parameters may still be easier with table look-up (Raibert, personal communication). Once the torques are known, it is still a complex problem to decide how to set the length-tension characteristics of the muscles to achieve these torques, because muscles may act about several joints, and their leverage about a joint typically depends on the joint-angle. However, this is a separate, modular problem that can probably be solved after the desired torques are known.

A much harder problem is to find a dynamically good trajectory for reaching to a target while maintaining dynamic balance. This is hard because the criterion of goodness typically involves the dynamics as well as the kinematics, so it is hard to keep a clean separation between choosing a trajectory and finding the torques needed to follow it. Also, considerations about the end of the trajectory determine whether a particular way of starting it will be good, so it is hard to grow *optimal* trajectories one piece at a time. Nevertheless, it may be possible to use heuristic rules to grow fairly good trajectories one piece at a time. The heuristics would examine the dynamic as well as the static properties of the current state of the system. The heuristics would encapsulate knowledge about good trajectories, but they would not need to generate the torques. Instead they would generate increments in the joint angles or angular velocities, and the computation of torques would be left to a separate module. This method of trajectory formation has not been implemented, but a simpler version which only considers the statics has been programmed to show how many autonomous rules can work together to coordinate many degrees of freedom in achieving multiple goals whilst satisfying the kinematic constraints.

One interesting discovery that came from the simulation was that the number of iterations required can be reduced by adding synergies—rules that know about combinations of basic actions whose side-effects cancel. Synergies are invoked whenever the effects they

control would be helpful; although synergies influence the increments in the joint-angles they do not override other rules. Instead, the increments computed by all the various rules are simply added together to produce a smooth blend

The computations benefit from using a single, global frame of reference that is based on the world rather than an ego-centric frame that is based on the body. In a real movement, all parts of the body move, and if any part is used to define a global reference frame then, relative to this part, stationary obstacles move around and the direction of gravity changes. The use of a single world-based frame for motor control simplifies the kinematics and dynamics and it also makes it easier to integrate visual information that is represented relative to a world-based frame (Hinton, 1981).

References

Abend, W., Bizzi, E., & Marasso, P. (1982). Human Arm Trajectory Formation. *Brain, 105*, 331–348.

Asatryan, D. G., & Feldman, A. G. (1965). Functional tuning of the nervous system with control of movement or maintenance of a steady posture. 1. Mechanographic analysis of the work of the joint on execution of a postural task. *Biophysics, 10*, 925–935.

Benati, M., Gaglio, S., Morasso, P., Tagliasco, V., & Zaccaria, R. (1980a). Anthropomorphic Robotics. 1. Representing Mechanical Complexity. *Biological Cybernetics, 38*, 125–140.

Benati, M., Gaglio, S., Morasso, P., Tagliasco, V., & Zaccaria, R. (1980b). Anthropomorphic Robotics. 2. Analysis of manipulator dynamics and the output motor impedance. *Biological Cybernetics, 38*, 141–150.

Bizzi, E., Accornero, W., Chapple, W., & Hogan, N. (1982). Arm trajectory formation in monkeys. *Experimental Brain Research, 46*, 139–143.

Brady, M. Trajectory Planning. In Brady, M., Hollerbach, J. M., Johnson, T. L., Lozano-Perez, T., & Mason, M. T. (1982). (Eds.) *Robot Motion: Planning and Control*. Cambridge, Mass: MIT Press.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., & Reddy, D. R. (1980). The Hearsay II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys, 12*, 213–253.

Greene, P. H. (1982). Why is it easy to control your arms? *Journal of Motor Behavior, 14*, 260–286.

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

Hinton, G. E. (August 1981). Shape representation in parallel systems. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence, 2*. Vancouver BC, Canada.

Hogan, N. (June 1982). Control and coordination of voluntary arm movements. *1982 American Control Conference*. Arlington VA.

Hollerbach, J. M. (1982). Dynamics. In Brady, M., Hollerbach, J. M., Johnson, T. L., Lozano-Perez, T., & Mason, M. T. (Eds.) *Robot Motion: Planning and Control*. Cambridge, Mass: MIT Press.

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

Lee, W. (1984). Neuromotor Synergies as a basis for coordinated intentional action. *Journal of Motor Behavior, 16*, 135–170.

Lozano-Perez, T. (1982). Task Planning. In Brady, M., Hollerbach, J. M., Johnson, T. L., Lozano-Perez, T., & Mason, M. T. (Eds.) *Robot Motion: Planning and Control*. Cambridge, Mass: MIT Press.

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

Mason, M. T. (1981). Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-11*, 418–432.

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

Raibert, M. H. (1978). A model for sensorimotor control and learning. *Biological Cybernetics, 29*, 29–36.

Schmidt, R. A., & McGown, C. (1980). Terminal accuracy of unexpectedly loaded rapid movements: Evidence for a mass-spring mechanism in programming. *Journal of Motor Behavior, 12*, 149–161.

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

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

Submitted October, 1983