

- [Kawamura *et al.*, 1988] S. Kawamura, F. Miyazaki, and S. Arimoto, “Realization of Robot Motion Based on a Learning Method,” *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1), 1988.
- [Lloyd and Hayward, 1991] J. Lloyd and V. Hayward, “Real-Time Trajectory Generation Using Blend Functions,” In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, April 1991.
- [Morasso and Ivaldi, 1982] P. Morasso and M. Ivaldi, “Trajectory Formation and Handwriting: A Computational Model,” *Biological Cybernetics*, 1982.
- [Qu *et al.*, 1991] Z. Qu, J. Dorsey, D. Dawson, and W. Johnson, “A New Learning Scheme for Robots,” In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, April 1991.
- [Rimey and Brown, 1991a] R. Rimey and C. Brown, “Controlling Eye Movements with Hidden Markov Models,” *International Journal of Computer Vision*, November 1991.
- [Rimey and Brown, 1990a] R. D. Rimey and C. M. Brown, “Selective Attention as Sequential Behavior: Modeling Eye Movements with an Augmented Hidden Markov Model,” Technical Report 327 (revised), Department of Computer Science, University of Rochester, April 1990.
- [Rimey and Brown, 1990b] R. D. Rimey and C. M. Brown, “Selective Attention as Sequential Behavior: Modeling Eye Movements with an Augmented Hidden Markov Model,” In *Proceedings: DARPA Image Understanding Workshop*, pages 840–849, 1990, (Shorter version of UR CS TR 327).
- [Rimey and Brown, 1991b] R. D. Rimey and C. M. Brown, “Controlling Eye Movements with Hidden Markov Models,” *International Journal of Computer Vision*, 7(1):47–66, November 1991.
- [Rosenfeld and Kak, 1976] A. Rosenfeld and A. Kak, *Digital Picture Processing*, Academic Press, 1976.
- [Simard, 1991] P. Simard, *Learning State Space Dynamics in Recurrent Networks*, PhD thesis, University of Rochester, June 1991.
- [Sutton, 1990] R. Sutton, “Integrated Architecture for Learning, Planning, and Reacting Based on Approximating Dynamic Programming,” In *Seventh International Conference on Machine Learning*, 1990.

References

- [Aboaf *et al.*, 1989] E. Aboaf, S. Drucker, and C. Atkeson, “Juggling a Tennis Ball More Accurately,” In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, 1989.
- [Albus, 1975] J. Albus, “A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC),” *Journal of Dynamic Systems, Measures, and Controls*, 1975.
- [Astrom and Wittenmark, 1984] K. Astrom and B. Wittenmark, *Computer Controlled Systems*, Prentice-Hall, 1984.
- [Atkeson, 1991] C. Atkeson, “Using Locally Weighted Regression for Robot Learning,” In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, April 1991.
- [Atkeson *et al.*, 1988] C. Atkeson, E. Aboaf, J. McIntyre, and D. Reinkensmeyer, “Model Based Robot Learning,” In *Proceedings of the 4th International Symposium on Robotics Research*, 1988.
- [Bondi *et al.*, 1988] P. Bondi, G. Casalino, and L. Dambardella, “On the Iterative Learning Control Theory for Robotic Manipulators,” *IEEE Journal of Robotics and Automation*, February 1988.
- [Branicky, 1991] M. Branicky, “Task-Level Learning: Experiments and Extensions,” In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, April 1991.
- [Brooks, 1987] Rodney Brooks, “Intelligence without Representation,” In *Proceedings of the Workshop on the Foundations of Artificial Intelligence*, 1987.
- [Brooks, 1991] Rodney Brooks, “Intelligence without Reason,” In *Proceedings: International Joint Conference on Artificial Intelligence*, 1991.
- [Connell, 1992] J. H. Connell, “SSS, A hybrid architecture applied to robot navigation,” In *Proceedings: IEEE International Conference on Robotics and Automation*, pages 2711–1718, Nice, France, May 1992.
- [Gelfand *et al.*, 1992] J. Gelfand, M. Flax, R. Endres, S. Lane, and D. Handelman, “Acquisition of Automatic Activity Through Practice: Changes in Sensory Input,” In *AAAI-92*, 1992.
- [Hollerbach, 1981] J. Hollerbach, “An Oscillation Theory of Handwriting,” *Biological Cybernetics*, 1981.
- [Jeannerod, 1988] M. Jeannerod, *The Neural and Behavioural Organization of Goal-Directed Movements*, Clarendon Press, 1988.

Acknowledgements

The authors would like to thank Jason Chu for his work on the simulator used to do virtually all the experiments in this report. He was responsible for the dynamic simulation code and a graphic display that provided additional insights during task executions. His time and effort is greatly appreciated.

algorithm, but the number of actual executions required from the robot. Requiring more robot executions at each learning iteration is acceptable if the number of learning iterations is significantly reduced.

It is possible that one set of basis functions can be found to cover most of the tasks a robot needs to perform. It seems more likely, however, that different tasks will be better performed with different basis functions. In either case, heuristics for choosing basis functions that will perform well are hard to develop. We know the basis function must be capable of representing the desired trajectories. Of course using this fact to make a choice assumes we have some idea of what the desired trajectories should look like and that may not always be the case. The basis function parameters should not cause degenerate trajectories when they are moved over large ranges. This problem was most apparent with the offset and standard deviation parameters in the gaussian basis sets. We also want changes in the parameters to produce linear changes in the task output. Unfortunately, it is difficult to choose a basis set to meet these guidelines. A possibility for future work is building more intelligence into the learning algorithm so it can make up for deficiencies in basis function choice. An algorithm that monitors its own progress and uses that information to decide how many iterations should be run at one range before attempting another would help cut down the number of iterations. It might also be possible to target the “problem parameters” such as time offsets and give them special handling when range increases are attempted. For example, compressing the range of allowable time offsets before attempting a greater range of outputs might allow these offsets to be used without creating degenerate trajectories.

When skill learning is done on a real robot, it will not be possible to try several basis functions to see which is the best. It seems that a choice would have to be made before the learning begins. The problem in choosing basis functions is that we must make the choice with little information about the trajectories we want to represent. The algorithm presented in this report assumes the basis set to be given, with the learning task being to search that space for the best trajectories. If we are given a set of desired trajectories and want to find the best basis set, there are algorithms to do that: this would be the case if we had an extensive set of examples to learn from. The work reported here assumes that it is the learner’s job to find the best trajectories. Therefore, the most general problem is to discover a good basis set while using it to find trajectories to do the task efficiently. Future plans are to devise a bootstrapping algorithm that will allow a robot to learn a skill and find the basis functions that are necessary to ensure efficient learning simultaneously. A first step may be devising an algorithm to determine whether the current basis function should be restricted or made more powerful.

Finally, it would be useful to implement other learning algorithms as a baseline against which to compare the algorithms presented here. It may be possible for other learning schemes to work with fewer iterations if they, too, are given the benefit of appropriate basis functions.

and extrapolation. This is important because it is not feasible for real robots to perform a large number of trials in order to learn a skill. The biggest drawback to the learning scheme appears to be its dependence on a linear relationship between the trajectories it produces and the output it hopes to achieve with those trajectories.

Basis functions are one way to escape the restrictions caused by a linear learning method. The experiments show that basis functions have a large effect on skill learning. Depending on the basis function chosen, the linear relationship may or may not exist. A more common question is not whether the relationship exists, its over what ranges of outputs can it be expected to hold.

When the learning algorithm seems to be incapable of finding appropriate trajectory sets, a change in basis functions may solve the problem. There are two underlying motives in choosing a new base. If the learning appears to be “on the right track” but moving too slowly, choosing a linear, or some restrictive nonlinear, basis set can reduce the dimensionality of the search space and thus speed up the learning process. Alternatively, it may appear that the current basis set is not capable of representing the desired trajectories. Then, the choice should be to a nonlinear basis set that has additional expressive power. Of course, there is no simple way to decide which of the two cases holds.

There are several ways to evaluate basis functions. The evaluation can just be a combination of the error and effort values over some range of outputs. Another way is to test the maximum attainable range within some number of learning iterations. Finally, the number of iterations required to get a reasonable result can be used to choose the best. For this learning algorithm to be useful with real robots, it is probably necessary for a basis function to perform well against all these metrics.

The control scheme being used can be just as important as the basis set. Since the dynamic simulator can tabulate average torque used, it was possible to get a direct comparison between torque and spring control. Over the experiments run, spring control required only about half the effort of torque control (as defined by sum of squared torques). When a robot seems to be incapable of learning a certain skill, it may be necessary to change the control scheme rather than the basis function. Of course, an alternative is to redefine the interpretation process so that it emulates a new control scheme.

7 Future Work

Given our experience to date and the framework above, certain future projects suggest themselves. One constraint that has allowed progress but may also be a barrier to generalization is the restriction to one-dimensional tasks. Extending the paradigm to higher dimensions requires some significant work. The simple extension is to add a change vector for each dimension, but that is unlikely to work well without additional modifications.

One change that may be necessitated by two and higher dimensional tasks, if not already necessary, is the use of more complex Weight Equations. The choice of a linear equation was made primarily because of the small number of points available to do the fit. It may turn out, though, that the benefits of a better Weight Equation outweigh the cost of needing extra data points to fit it. The real constraint is not the number of iterations of the learning

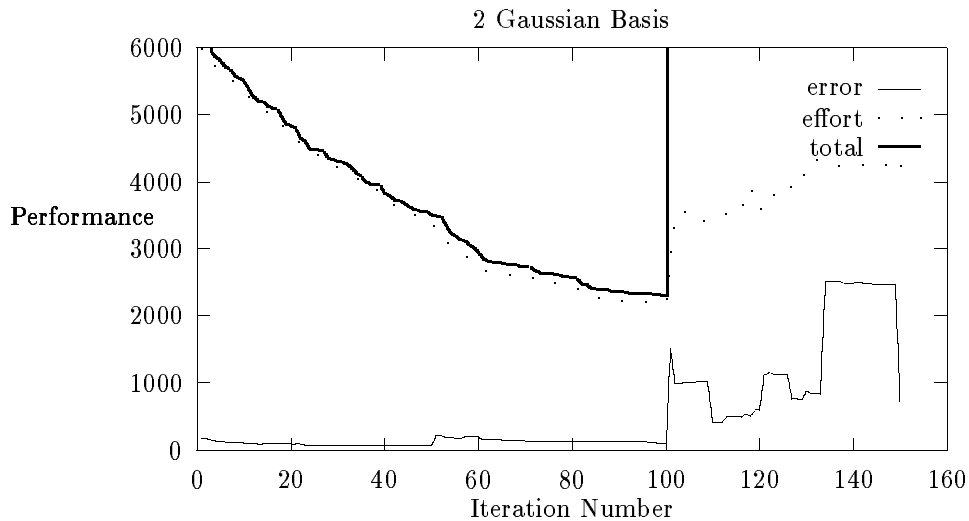


Figure 25: Attempting distances 1600, 2000, and 2500 with 2 gaussian basis spring control

curve is the sum of those two plus a penalty for not being able to obtain the full range of outputs.

All 20 delta basis executions succeeded in producing the full range of 1600 - 2500. As seen previously, the linear basis function outperformed the delta basis to a distance of 1600 with an average 42% less for its performance metric. However, only 16 of the 20 executions were able to throw a distance 2000 at iteration 100 and only 4 of those were ever able to reach 2500. This indicates that making the spring setpoints a linear sequence was too restrictive for greater distances. The quadratic basis set fared better with 19 of its executions reaching distance 2500. The gaussian basis outperformed the delta basis out to a distance of 2000 at iteration 100. Only half of the executions were able to get distance 2500, though. These results indicate that the different space spanned by the gaussian basis can better represent trajectories for distances 1600 - 2000, but becomes unmanageable for greater distances.

Spring control is comparable to velocity control with respect to range of distance achieved. In both cases, use of the delta basis resulted in a distance of 2500 being reached by both forms of control in all trials. The quadratic basis also performed similarly with only one run from either control method failing to reach 2500. The other basis functions were generally unable to get distances beyond 2000 regardless of the control method.

6 Conclusions

A simple learning scheme is presented that allows robots to learn skills by practicing. The important feature of the algorithm is that it learns tunable skills. Rather than just acquiring a single trajectory for a zero-dimensional task, it learns a whole class of trajectories. Its ability to learn with only a small number of iterations comes from its use of interpolation

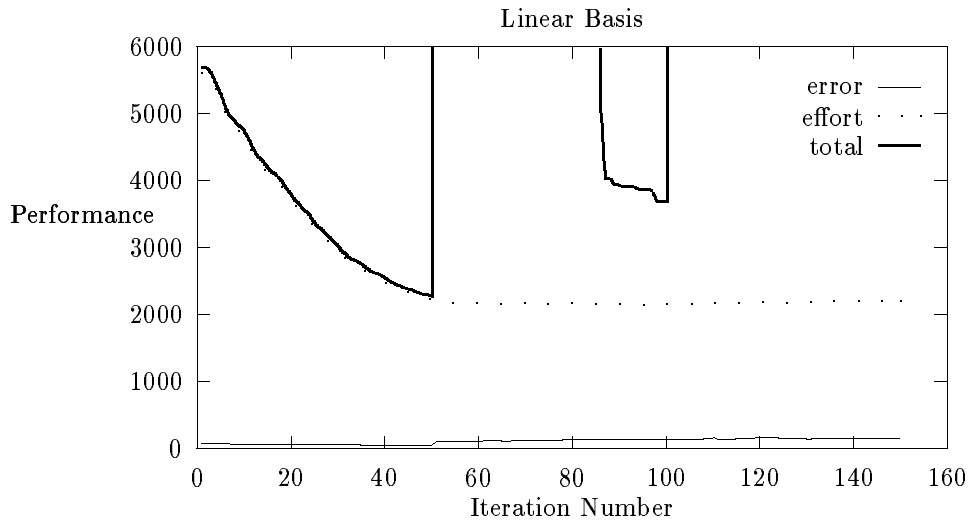


Figure 23: Attempting distances 1600, 2000, and 2500 with linear basis spring control

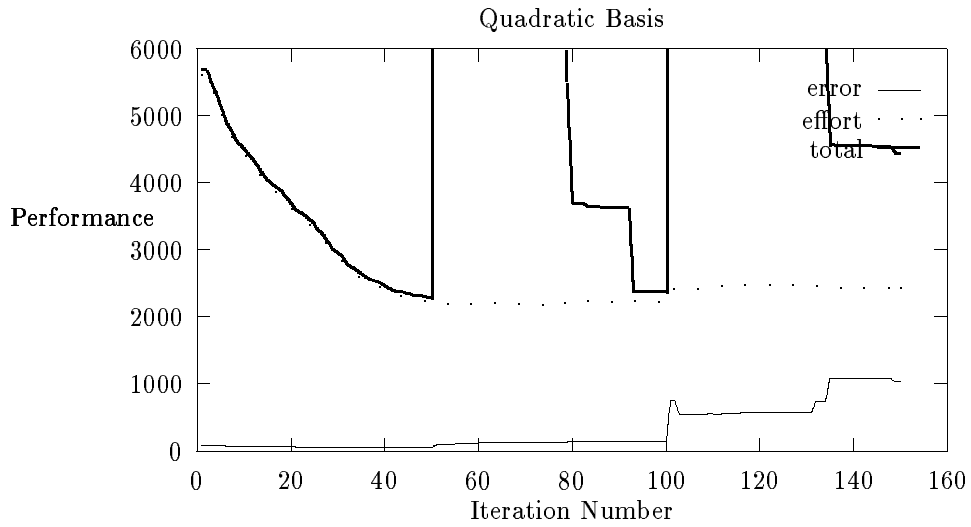


Figure 24: Attempting distances 1600, 2000, and 2500 with quadratic basis spring control

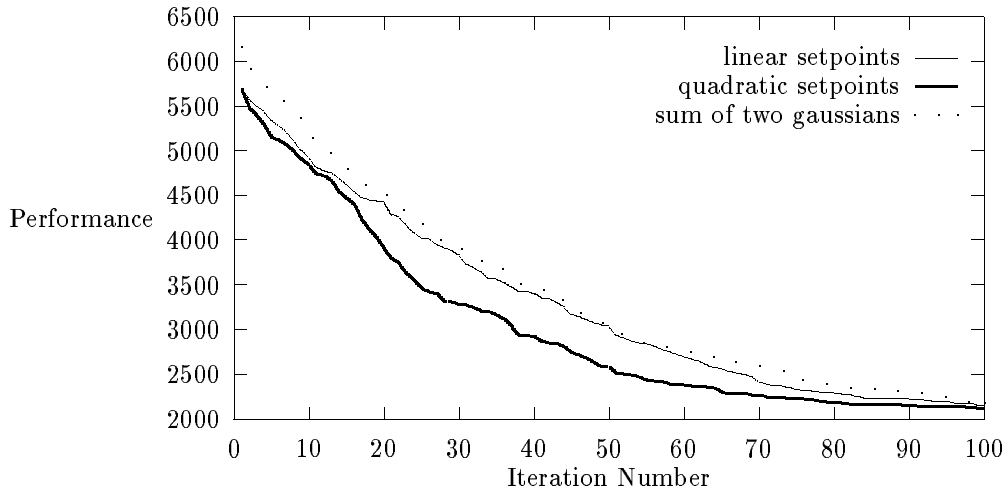


Figure 21: Comparison of Basis Functions on Spring Control: SMOOTH only.

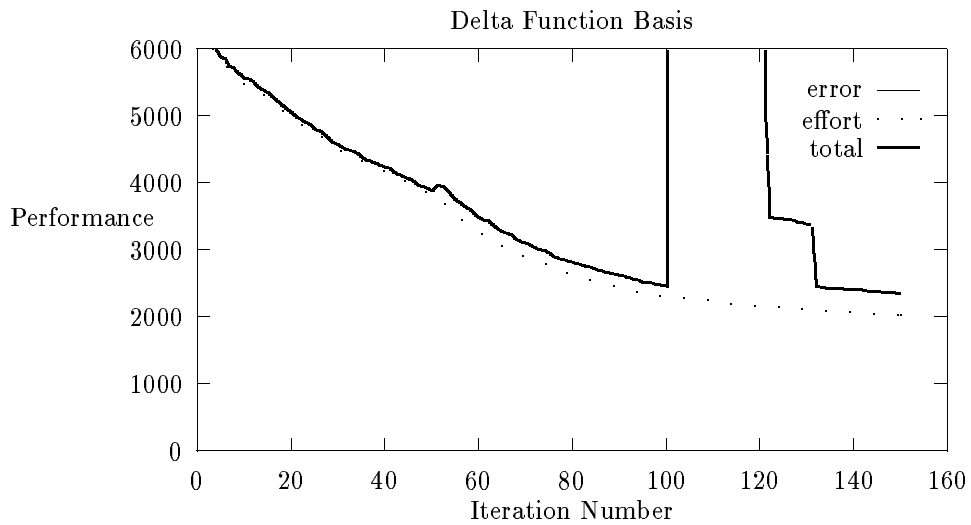


Figure 22: Attempting distances 1600, 2000, and 2500 with delta function basis spring control

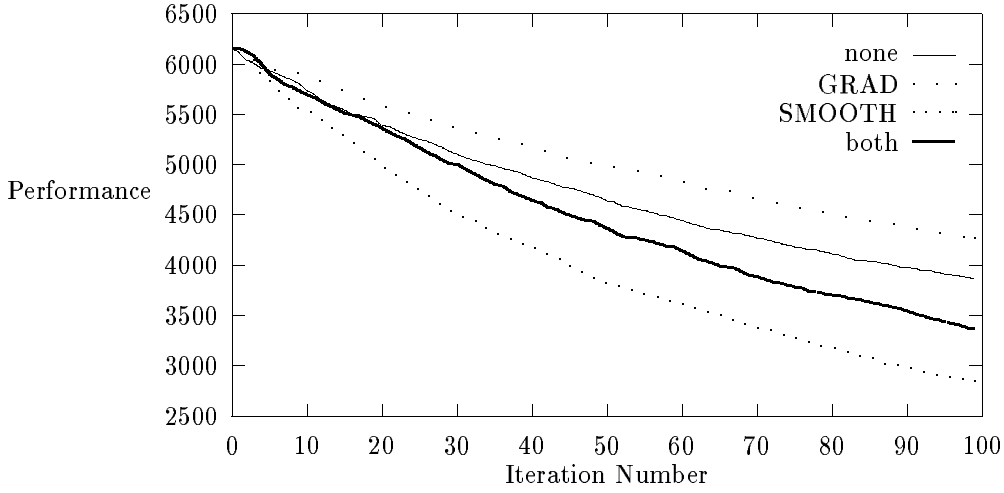


Figure 20: Comparison of Learning Algorithm Options on Spring Control

the same position. The dynamics of the system will likely cause a position and/or velocity offset to remain at the end of the trajectory, but wild fluctuations are impossible because of the stabilizing effects of a damped spring.

Linear, quadratic, and gaussian basis functions were tested with spring control. The gaussian basis is the sum of two 3-parameter gaussians as used in the velocity control experiments. SMOOTH was turned on and GRAD was left off. Fig. 21 shows the averages of 20 runs for each basis set. All three of the basis sets performed better than the delta function basis. (2847 vs 2114-2178). No optimal values are available, but individual runs attained performance metrics below 1900. It was interesting to note that the variance between runs was much higher for the gaussian basis set than for the other two. From that, we can speculate that it is better able to represent low effort trajectory sets, but individual runs were victims of the offset shifting problems described earlier.

5.3 Comparison to Velocity Profiles

The previous section compared spring control with torque control based on the amount of required torque. Torque comparison is not possible with velocity profiles because the effort metric differs significantly. Also, the accuracy results can not be compared directly since different weights were applied when computing the total metric. The closest fair comparison comes by testing for attainable ranges. The experiments described in sections 3 and 4 were repeated for delta, linear, quadratic, and gaussian basis sets with spring control. The results are shown in Figs. 22 through 25.

The graphs can be compared directly to the ones done with velocity profiles. The error curve is the second term in equation (10) and the effort curve is the first term. The total

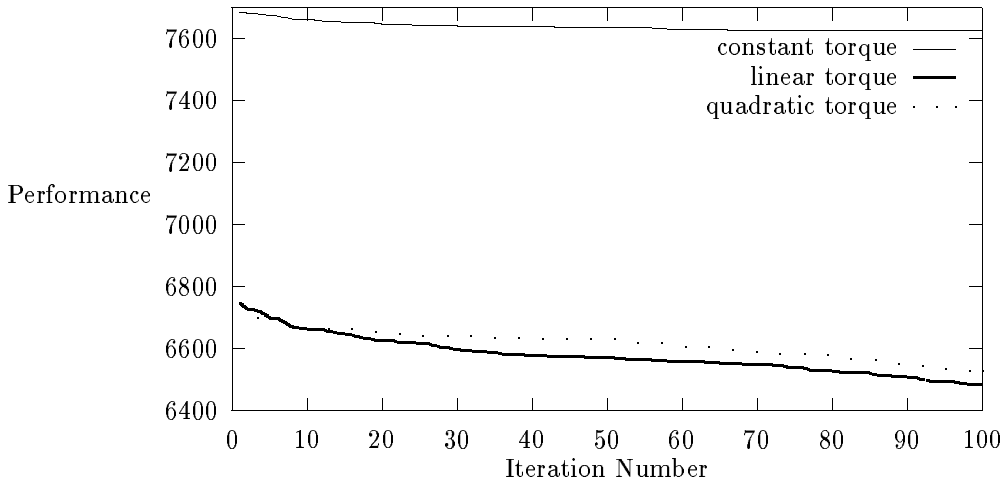


Figure 19: Comparison of Basis Functions on Torque Control

Additionally, results in the next section show much lower squared torque values.

5.2 Spring Control

Six values are used for each joint under spring control (2 joints). Each value represents the spring’s positional setpoint for a duration of 0.5 seconds. The spring constants are a fixed part of the robot’s physical description. The performance metric, H , is exactly the same as for torque control. While the dynamic simulator computes the joint movements, it also tabulates the torque being provided by the motors. This allows a direct comparison between torque and spring control. As before, the first experiment tested the various combinations of learning parameters. Fig. 20 shows the results.

The use of spring control allowed a large improvement in the performance metric. The best case under spring control is 55% lower than for torque control. This demonstrates that choice of control mechanism is just as important as choice of basis in the interpretation process. Logically, the control scheme could be considered to be part of the interpretation since robots usually send currents to motors regardless of the control scheme the user sees. SMOOTH helps learning under spring control. A possible explanation is that smoothing the setpoints tends to create trajectories that do not put setpoints opposite the current motion. Doing so would create unnecessarily large torques. Again, GRAD only hindered performance.

The additional stability provided by spring control turned into large improvements in learning. To understand why spring control is more stable, consider a trajectory with a small alteration near the beginning. The previous section showed how disastrous that can be with torque control. With spring control the alteration will create a position and velocity offset. However, the unchanged setpoints later in the trajectory will always work to obtain

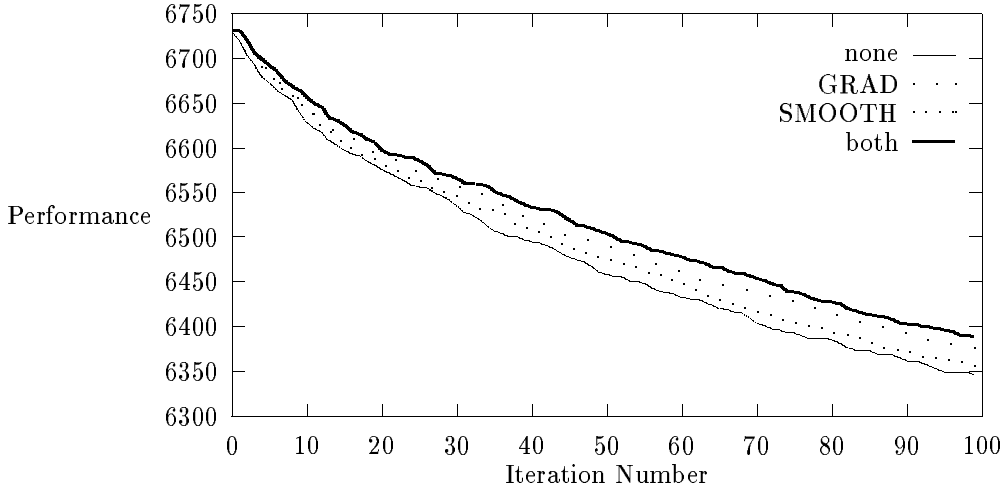


Figure 18: Comparison of Learning Algorithm Options on Torque Control

not include jerk. GRAD failed to improve results just as it did with the velocity profiles for similar reasons. The most important result, however, was the overall poor performance of the learning algorithm. The best combination of learning parameters yielded less than 6% average reduction in H as compared to 86% with the velocity profiles.

A closer look at the execution traces showed that many of the new trajectories attempted caused the robot to fly out of control. Since skills are run open-loop there is no opportunity to correct wild fluctuations in robot motion. Consider a small alteration near the beginning of a joint velocity profile vs a joint torque profile. In the former case, a positional offset remains throughout the rest of the execution. In the latter, a velocity offset remains and thus positional offset increases. Additionally, the difference in position later in the trajectory completely changes the affects of later torques even though the torques themselves are unmodified.

Despite the problems with open loop torque control, experiments were performed using constant, linear, and quadratic basis functions. In each case the torque profile for each joint was represented by the coefficient(s) of the indicated function. Both SMOOTH and GRAD were turned off for these experiments. The results are shown in Fig. 19.

The graph shows the effect of basis function choice. The linear and quadratic basis functions start at a level 24% lower than the constant basis. It makes sense that a basis function covering a larger set of torque profiles would be able to represent a lower effort set than one covering a small set such as those with constant torque. Unfortunately, none of the experiments showed a significant reduction in the metric, H . The quadratic and linear basis functions were the best with about 4% reduction. Unfortunately, we do not have optimal values for comparison as we had with velocity control. The small improvement due to learning combined with the large difference observed with the use of different basis functions is a good indication that the linear and quadratic results are not close to optimal.

5 Other Forms of Plant Control

The experimental results presented so far have assumed a massless robot with kinematic (velocity) control. The performance of basis functions also depends on the type of robot control being used. Therefore, it is desirable to test the learning algorithm presented with other forms of control. This section presents results for two other types. Torque control is an alternate method that is more low-level than the velocity control used previously. Even with kinematic control, the plant controller eventually has to convert to a form of torque control. Usually, this must be done closed-loop because of the quick accumulation of error due to things such as inaccurate plant modeling. The results here also attest to the difficulty of open-loop torque control. Skills are by definition open-loop and no form of closed-loop position or velocity monitoring is possible.

The poor performance achieved with torque control led to the search for other control schemes. Dynamic simulation was already available and some of the motivation for this research is a quest for information about how the human motor system might work. Since spring models have been proposed for human muscles, the existing dynamic simulator was modified to allow a form of spring control. For these experiments, the spring constants are a fixed part of the task plant (but need not be). The control variables are the spring setpoints. Experiments using spring control showed a significant improvement over torque control in the ability to learn the one-dimensional throwing task.

5.1 Torque Control

As with the joint velocity control experiments, the simulator uses a delta function as the default basis for torque control. Again six values were used for each robot joint (2 joints for these experiments). The torques were specified at 0.5 second intervals. The effort level was computed as the sum of squared torques rather than the sum of squared accelerations used previously. Accuracy was also included in the metric:

$$\mathbf{j} = H(\tau) = \left(\sum_{j=0}^1 \sum_{i=1}^5 \tau_{j,i}^2 \right) + kE \quad (10)$$

where τ is the entire torque vector, E is the average error in output distance and k is a constant used to provide a reasonable weighting between the two parts of the metric. Just as it was for the velocity profiles, the performance metric is defined such that lower values are better.

The first experiment tested the eight variations of the CHANGE, SMOOTH, and GRAD options for the learning algorithm. The results are shown in Fig. 18. The executions with CHANGE turned off were left out because each performed slightly worse than with CHANGE on. There are several conclusions to draw from the graph. First, the best results were obtained without GRAD or SMOOTH. It is not surprising that SMOOTH did not help since that algorithm was devised with velocity profiles in mind. SMOOTH would probably help if the performance metric included a measurement of “jerk” (third derivative of position or first derivative of torque) as suggested by some authors, but the H defined above does

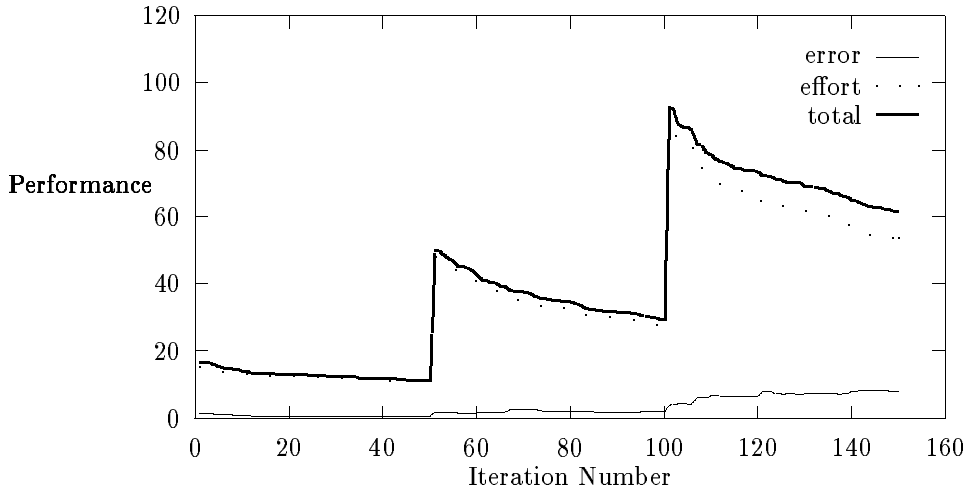


Figure 17: Results for Release Position and Velocity Basis

because of the base and change vector scheme used to produce the range of output values, it is just as important to ask whether changes in the basis function parameters will lead to degenerate trajectories. A related issue is whether another form can be found for the Weight Equation (eq. (2)) that maps task instances into weights for the change vector that better suits the parameters of a desirable basis set.

4.3 Terminal Position and Velocity

As stated earlier, the kinematics of the robot, the effort measurement function, and the task itself make the final joint positions and velocities the only important features of a trajectory. Those things alone determine the landing position of the ball, and from the final positions and velocities the optimal trajectory can be computed. This suggests another basis possibility. Each joint velocity profile is represented only by the velocity and position at time six. Here is one example of computation that can be done in the interpretation process that does not fit usual definitions of a basis function. Fig. 17 shows the results for this basis set.

This basis set outperforms all the ones attempted for distances over 2000. One problem encountered by the other basis sets was their inability to represent the optimal trajectories. Even though the learning process had the optimal trajectories available, the trajectory generation process was not able to regenerate them because of the basis function chosen. The simplifications in representation that were used to create a simple (requires few iterations) learning algorithm also restrict the sets of trajectories that can be represented. The choice of basis function determines whether that restriction will damage the final results.

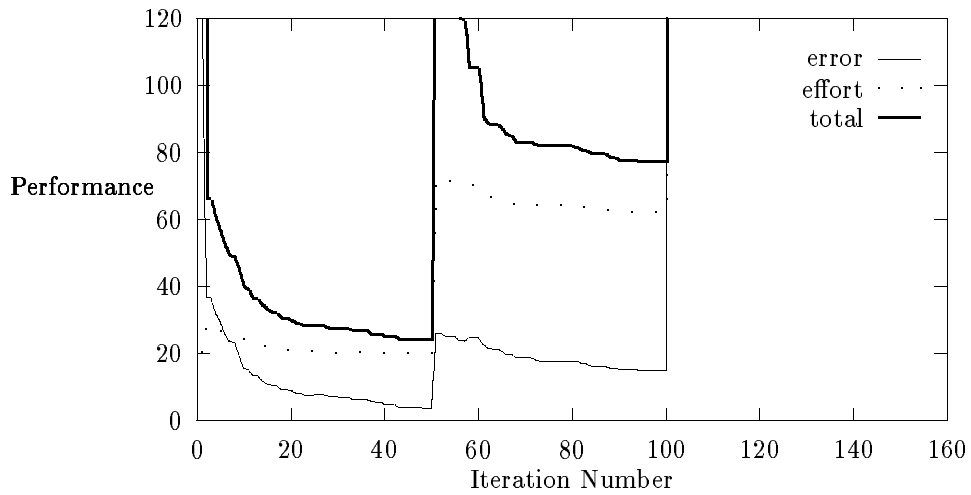


Figure 15: Results for Gaussians Fixed at 2 and 5

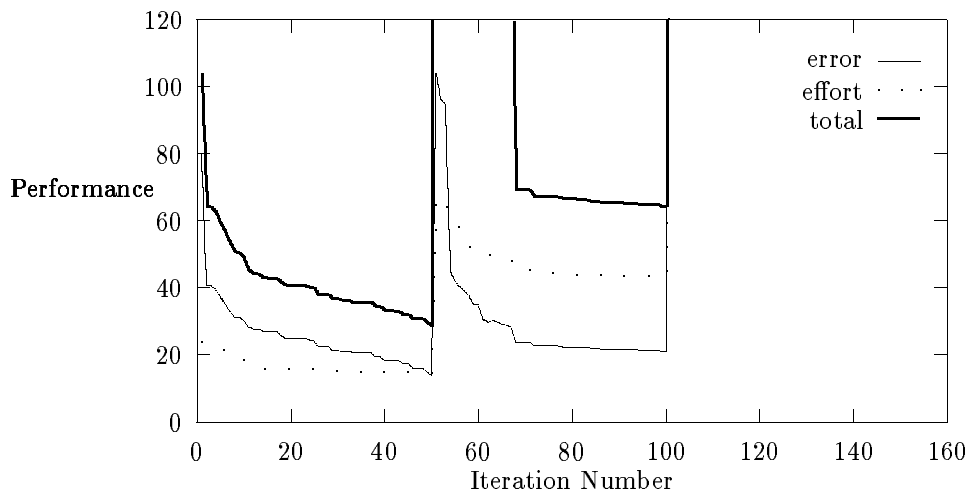


Figure 16: Results for Gaussians Fixed at 1, 3, and 5

out to be the offset parameter that is part of the basis function description. As the weight on the change vector is increased, the position of the basis function is shifted in time. The shifting seems like a good thing to do and works well over short ranges. As the weight continues to increase, however, the gaussian, or velocity ramp, gets shifted right out of the useful range of the trajectory. The trajectory always starts at time zero and the ball is always released at time six. Therefore, any part of the basis function outside that range is ignored. Only the velocity profile generated between times zero and six is sent to the robot.

For the delta basis, the physical constraints of the robot limited performance, but for the gaussian and step acceleration basis functions the trajectory generation scheme often is not capable of creating trajectories that reach the robot's limits. Before that happens, the basis functions are time shifted away and degenerate trajectories result. The shifting problem is particularly bad for increasing the range of performance. It makes sense that the most accurate, lowest effort way of generating trajectories for a given distance range might be to use the time shift to move the basis functions throughout the whole valid time range. Unfortunately when the change vector weights are increased in the hopes of increasing performance to a new range, the old generation scheme would almost immediately shift the basis functions outside the zero to six time range and begin creating degenerate trajectories.

There are several possible ways to handle the shifting problem. One is to make additions to the learning algorithm that make more intelligent use of time shifts, especially when attempting to extend the performance range. For example, the full range of time offsets under the current scheme could be compressed by modifying the interpreter to map them onto a smaller range. Then, when a greater distance is attempted, the effective time offset would still have a reasonable value. Another method would be to substitute a quadratic or other more complex function into the Weight Equation as mentioned earlier. Doing so might allow the full use of time offsets without letting them get out of a reasonable range. Finally, the problem can be avoided completely by using basis functions that do not make use of time offsets.

The last option led to experiments on two new basis sets that contain gaussians at fixed time offsets. The first uses two gaussians fixed at times 2 and 5 while the other uses three gaussians at times 1, 3, and 5. Figs. 15 and 16 show the results of using fixed gaussians.

A quick look at the graphs shows that fixing the time offsets did not solve the problem. In both cases, ranges greater than 2000 were still not consistently obtainable. By fixing the offsets, we make a step toward just using the delta basis we started with. For example, if the basis set was six fixed gaussians at even intervals and with small values of σ , the six velocity commands sent to the robot would just be the six magnitudes of the gaussians. Therefore, the performance would probably approach that of the delta basis as the number of fixed-offset gaussians was increased to match the number of points in the velocity profile of the delta basis set. Another problem is the standard deviation parameter. Just as the offsets could shift out of their valid range, it was possible for the width to be shrunk to insignificance. It was also possible for them to pass through zero to take on negative values. The conclusion is that there are two representation issues to be considered when choosing basis functions. First, whether the basis function is likely to generate the trajectories desired. That may mean reducing the search space to help learning or producing trajectories more complex than can be generated with an equal number of linear parameters. Secondly,

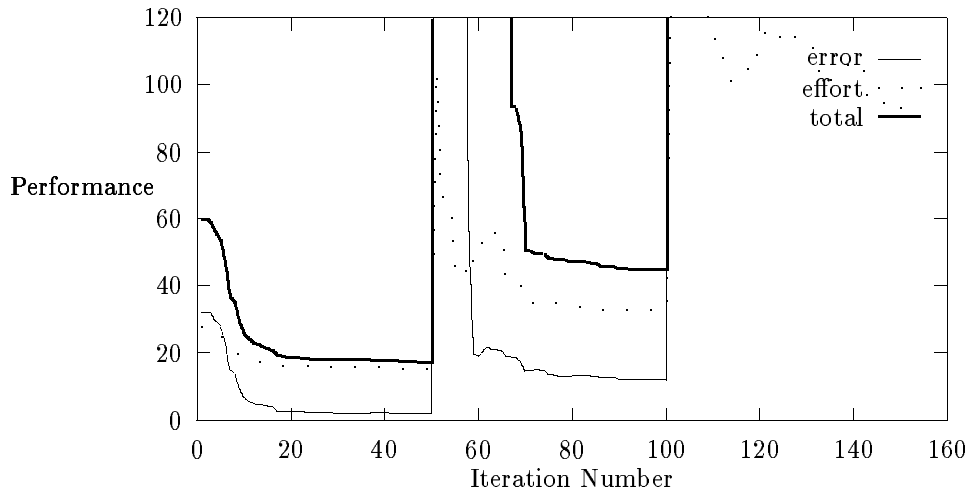


Figure 14: Results for 2 3-parameter Gaussian Basis

quadratics as the best, followed by step accelerations and then the delta basis for short distances. However, another metric might be ability to extend range to greater distances. Step accelerations do not appear to be any more useful than quadratics when attempting distances over 2000 since such a larger number of the experiments for both basis sets failed to generate throws as far as 2500. Finally, we might also consider the number of learning iterations required. A closer look at Fig. 13 shows that almost all the improvement was made in iterations 1-10 and 51-60. By changing the number of iterations at each range, we could learn a good scheme to a range of 2000 in only 20 iterations with step accelerations. This contrasts the results of the quadratic basis where little improvement was made because the curves started so low to begin with. This difference between the different basis function curves indicates that it may be better to observe the performance of the learner and change the number of iterations it runs accordingly rather than just running a fixed number.

4.2 Gaussians

A basis function that seems likely to produce low effort trajectories is gaussians. In order to have six parameters total, two gaussians with three parameters each were used: offset, magnitude, and standard deviation. The sum of two gaussians form the velocity profiles for each joint. The results of using the gaussian basis function are shown in Fig. 14.

The performance to a range of 1600 is the same as that of the step accelerations, while the performance to a range of 2000 is the same as for the delta basis. As with step accelerations and quadratics, a range of 2500 was not consistently attainable.

After looking at the results of the gaussian and step basis functions, one might wonder what went wrong? Neither of them was able to get a range as good as the delta basis even though they seemed to be a reasonable choice as a basis function. Part of the problem turns

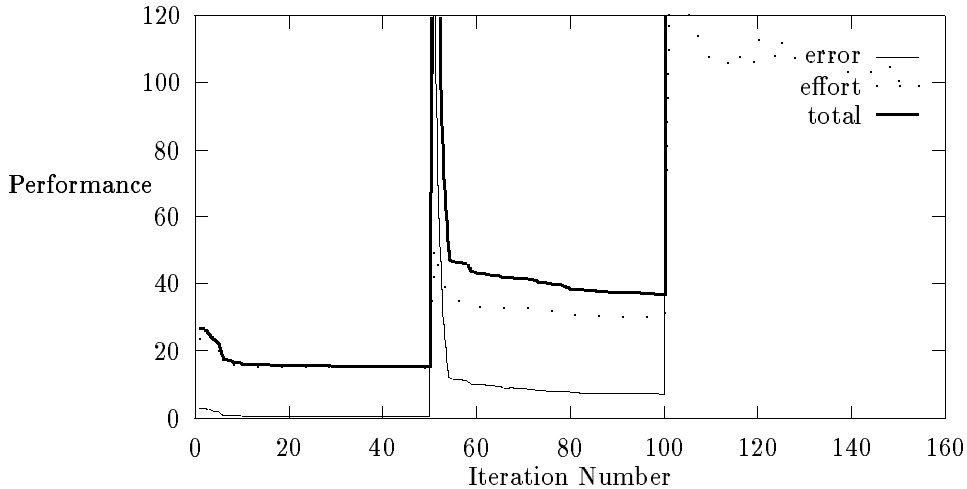


Figure 13: Results for 3 Step Acceleration Basis

ories for a given task. The basis functions in this section were chosen because they appear to generate trajectories that rate well on the performance metric, eq. (4).

4.1 Step Accelerations

Step accelerations are physically intuitive basis functions closely related to the effort criterion. The trajectories consist of 3 step accelerations. Each has two parameters: a time offset and a magnitude. Note that the time offset is the parameter that makes step accelerations a nonlinear basis set. The velocity profile for each joint is generated by combining the step accelerations (each producing a velocity ramp). Using three gives a total of six parameters per joint, which is the same number used in the delta basis. The results are shown in Fig. 13.

The first thing to notice from these results is that iteration 1 is much better than for the delta function basis (though not better than iteration 1 of the quadratic basis). This difference is just one of the effects of changing the basis function. To create the initial base and change pair, the velocity profiles used to start the delta basis experiments were input to a gradient descent algorithm to find a set of step accelerations to fit them. It turned out that these were much better even before the learning began. The starting point is a good indicator of how well a given basis function fits with the trajectory generation scheme and often predicts how the learning will go.

At the end of 50 iterations, the total performance effort value was slightly over half of the value obtained with the delta basis (i.e. it was about twice as good). At the end of 100 it is marginally lower, while at 150 a significant number of the 20 executions still were not able to reach the full range of 2500. These differences indicate several ways that basis functions can be evaluated. The obvious evaluation of effort and accuracy points to

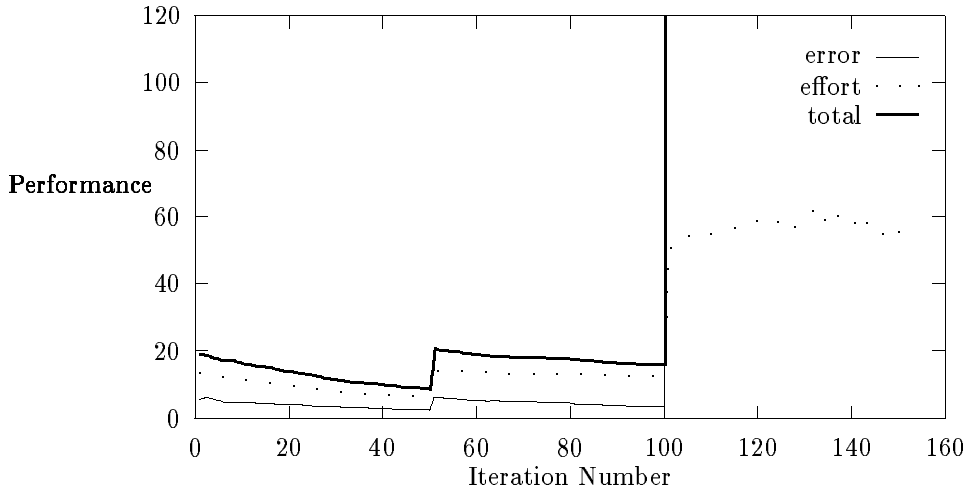


Figure 12: Results for Quadratic Basis

This basis function proved to give the best accuracy and lowest effort of any basis set tested for distances of 2000 and less. However, it was poor at ranges up to 2500. Most of the runs never succeeded in attaining a range of 2500. Although the quadratic form easily represents the optimal trajectories, the parameterization did not lend itself to a linear mapping between desired outputs and quadratic parameters. A look at the optimal trajectories provides a possible explanation. The best ending velocity for joint 1 is near zero for the short throws. It increases to a value of about 2.5 degrees per second at a range of 2100. From there it decreases again. The parameterization of a quadratic used above generally causes the final velocity for a joint to increase as the change vector weight increases throughout the entire range of acceptable weights. Therefore, it is difficult for this basis function to make accurate throws past a distance of 2100. The results for the quadratic basis function demonstrate a disappointing fact. Performance at shorter ranges is not a good predictor of performance at greater ranges. The quadratic basis results make a case for trying a more complex Weight Equation for mapping problem instances into change vector weights. In this case, replacing eq. (2) with a quadratic form would allow the ending velocity for joint 1 to be handled better.

4 Non-linear Basis Functions

This section considers some nonlinear basis functions. The class of trajectory sets representable with these functions includes at least some not representable with linear basis functions. Any single trajectory can always be represented with the delta basis, but there are sets of trajectories that can not be generated as the linear combination of a single base and change vector using that basis. Choosing a nonlinear function might be based on the assumption that the delta basis is not powerful enough to generate the desired set of trajec-

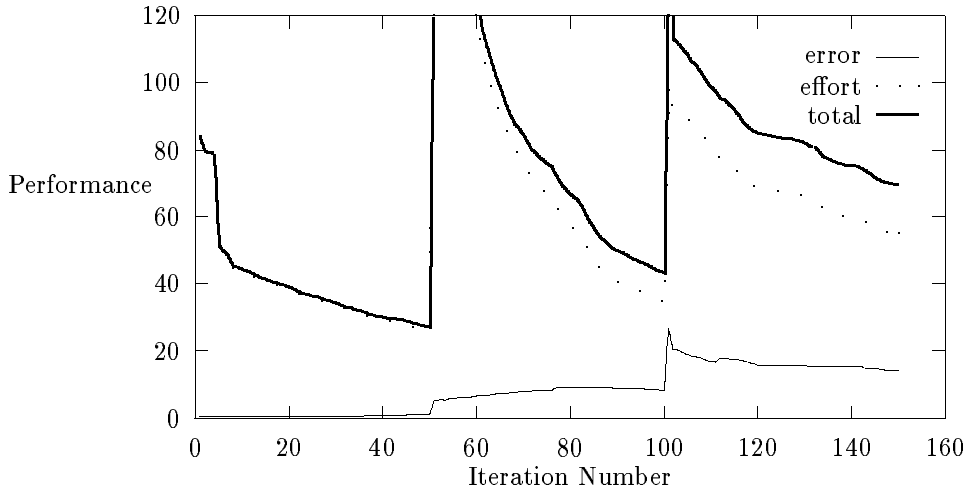


Figure 11: Results for Delta Function Basis

distances that are likely to fail because the current trajectories are not close enough to what is needed. In the mean time no progress is made toward improving accuracy and effort. Alternatively, trying to make too many small increases in range causes too many iterations to be spent readjusting to a new range as shown in the first few iterations after 50 and after 100 in Fig. 11. Therefore, the choice of three intermediate ranges is an empirically determined compromise.

An important property of basis functions that can not be seen in the performance graphs is what limits the maximum distance thrown for a particular scheme. In the case of the delta function basis, the limit is almost always the physical limitations of the robot. The simulator sets limits on the positions and velocities each joint can attain. As the weight on the change vector is increased, the resulting trajectory throws a greater distance. This can continue until the scheme produces trajectories that violate the physical constraints on the robot. Extending the range of throws implies a search for new trajectories that throw farther without exceeding the robot's capabilities. Later sections describe other factors depending on the basis function that limit the robot's maximum output distance.

3.2 Quadratic

It is desirable to find basis sets that can be used regardless of the task and robot. The equations for optimal trajectories show that the optimal velocity profile for each joint is a quadratic intersecting the time axis (velocity = 0) at -1. The solution would remain quadratic even if the physical characteristics of the robot were changed as long as the task only depended on the final velocities and positions. It seems reasonable, then, to choose a quadratic for the basis function. The quadratic has only two parameters as it is fixed to pass through the point (-1,0). The results of using a quadratic basis are shown in Fig. 12.

of trajectories for a given task. In that case it is necessary to expand the expressive power with nonlinear bases (those not expressible in the form of eq. (5)). Some nonlinear bases are discussed in Section 4.

We conjectured that the choice of basis functions can have a large effect on the performance of the learning algorithms presented earlier. The experiments in this section are altered versions of those described earlier. The range over which the robot is expected to throw is larger. One of the goals in skill acquisition is learning to expand the range of performance. Rather than running 100 iterations in an attempt to throw accurately over the range 1300-1600, 3 sets of 50 iterations for the ranges 1300-1600, 1300-2000, and 1300-2500 were run. The second and third sets start with the results of the previous set while the first set starts with the same base and change used in the earlier experiments. Previously, amount of effort was the only metric used to rate base and change vector pairs. Over a small range accuracy is not a problem, but as the range increases so does the amount of error caused by the linear fit of change vector weights to resulting output. Therefore, it was necessary to include the average distance error in the overall metric. Since a variety of basis function options were tested, it was not practical to try all the learning parameter options as well. The parameter settings that produced the best results in the previous sections were used. These are the closed form kinematic optimization, change vector modification, and smoothing, but no form of gradient descent.

3.1 Delta Basis Function

The velocity profile control in the previous experiments uses a delta function basis. (The coding of the kinematic portion of the simulator makes the delta basis implicit when no other basis function is used explicitly) This, along with the optimal values listed in Table 1, will serve as a baseline to compare other basis functions against. The averages of 20 executions are in Fig. 11.

The error curve in Fig. 11 is the average difference between the desired and actual distance thrown. The effort curve comes from equation 4. The total curve is the sum of those two plus a penalty if the entire range of outputs could not be attained. The addition of this penalty makes it clear when a basis function has failed to reach the full range of desired outputs. The first 50 iterations show a decline in effort level similar to that seen in Figs. 8 and 10. Note that the error is extremely low. That is why accuracy was not taken into account until this basis function comparison. At iteration 51 there is a large jump in all three values. This is where the first attempt at extending the range to 2000 is made. The effort and total then decline steadily until iteration 101. There the range is extended to 2500.

The final goal for the robot in these experiments is to throw accurately, with minimal effort over the range 1300-2500. An alternative path to the goal might be to just start at that range and run 150 iterations. Several experiments were tried to test different methods of reaching the final goal range. The initial base and change pair that the learning process starts with are only capable of producing trajectories out to a range of 1900 and extending the range of a scheme is not trivial. The experiments showed that when the full range is attempted immediately, many of the iterations are wasted by random attempts at greater

where the F_i are the basis functions, x is time, and the a_i are the parameters to the basis functions. Note that all the polynomial basis functions can be represented this way and thus are linear bases. The learning algorithm must then choose the basis function parameters; it uses linear combinations of the base and change vectors: $\langle a_1^b \cdots a_n^b \rangle$ and $\langle a_1^c \cdots a_n^c \rangle$. Using the linear combination rule of eq. (3), the range of producible trajectories is:

$$\sum_{i=1}^n (a_i^b + w a_i^c) F_i(x) \quad (6)$$

for all real valued w . The simulated robot takes a discrete time sequence of joint velocities as its control signal regardless of the basis function being used. Therefore, a single trajectory is generated by evaluating eq. (6) at m (the number of trajectory time steps) values of x for a fixed w .

For discrete time control the class of trajectories produced by any linear basis function is a subset of the class producible with the delta basis function. Assuming some basis function defined by eq. (5), we can produce the equivalent set of trajectories with a delta basis by using the correct base and change pair. Create $\langle \hat{a}_1^b \cdots \hat{a}_m^b \rangle$ and $\langle \hat{a}_1^c \cdots \hat{a}_m^c \rangle$ by evaluating:

$$\sum_{i=1}^n a_i^b F_i(x) \text{ and } \sum_{i=1}^n a_i^c F_i(x) \quad (7)$$

at m values of x where m is the number of time steps in the desired velocity profile. By the same linear combination rule, eq. (3), a single trajectory is:

$$\langle \hat{a}_1^b + w \hat{a}_1^c \cdots \hat{a}_m^b + w \hat{a}_m^c \rangle \quad (8)$$

and the whole set of trajectories is generated by ranging w over the reals. Substituting the m evaluations of eq. (7) into eq. (8) shows that the resulting class of trajectories is equivalent to those generated with eq. (6).

$$\langle \sum_{i=1}^n a_i^b F_i(1) + w \sum_{i=1}^n a_i^c F_i(1) \cdots \sum_{i=1}^n a_i^b F_i(m) + w \sum_{i=1}^n a_i^c F_i(m) \rangle \quad (9)$$

Therefore, no linear basis function allows representation of more trajectory classes than does the delta function basis. Since a basis function may allow a trajectory to be represented more efficiently than in the delta function basis, linear basis functions are usually thought of as restricting the set of producible trajectories. This can be an advantage if the restriction does not exclude desirable trajectories and the number of parameters has been reduced. Learning with a delta basis function requires search in an m -dimensional space while learning with a generic linear basis function is a search in an n -dimensional space. For example, consider trajectories with 6 time steps. Using a delta basis function yields a 6-dimensional search. Using a quadratic (requires 3 $F_i(x)$: $x^2, x, 1$) yields a 3-dimensional search for basis function parameters. It is this search space reduction that can speed up the learning. Alternatively, even the delta basis function may not be able to represent an acceptable set

Range	Avg Effort
1300 - 1600	4.56
1300 - 2000	10.42
1300 - 2500	24.38

Table 1: Optimal average efforts for three task ranges

with GRAD are not displayed since it actually hurts the overall performance. Using this method had little effect on the final averages at the end of 100 iterations. Only the “change” line finishes significantly lower. The important difference is the large initial drop in effort level within the first 10 iterations seen when using the task information. The difference can be valuable when the algorithm is run on a real robot and running many iterations is not practical.

Making use of the additional task information also made it possible to find a good approximation to the true optimal trajectories for this task. Since the optimal 12-dimensional trajectory can be found from any 4-dimensional final joint position/velocity vector, a brute force search of trajectories within a reasonable range of ending positions and velocities was done. The final position values were quantized into 0.25 degree increments and the final velocity values into 0.10 degree increments. Finally, the output distance was quantized into bins of size 10. After trying all trajectories the best one for each distance bin was recorded. With that data the true optimal effort level for several ranges of distances was found. The results are listed in Table 1. Only the first row is relevant to the experiments presented so far. The best combination of options produced a result between two and three times the optimal level (12.53 vs 4.56 optimal). The other two rows contain information for comparison with results in the next section.

3 Basis Functions for Velocity Profiles: Linear Bases

A trajectory basis function is used by the interpreter to convert the output of a skill to an input for the plant controller. The experiments reported in the previous section were done with a delta function basis. The base and change vectors specified points on the joint velocity profiles that the robot should produce, and the resulting joint velocities were piecewise constant. It is possible, however, to have the values in the base and change vectors represent a set of parameters for a function that specifies the velocity profile. As an example consider a gaussian curve. A six element vector could represent the offsets, magnitudes, and standard deviations of two gaussians. The sum of those two would then give the desired velocity profile. Any parameterizable function can be used in a basis set for trajectories.

One subset of possible basis functions is the functions that produce results linear in their parameters (linear bases). These functions all have the form:

$$\mathbf{B} = \sum_{i=1}^n a_i F_i(x) \tag{5}$$

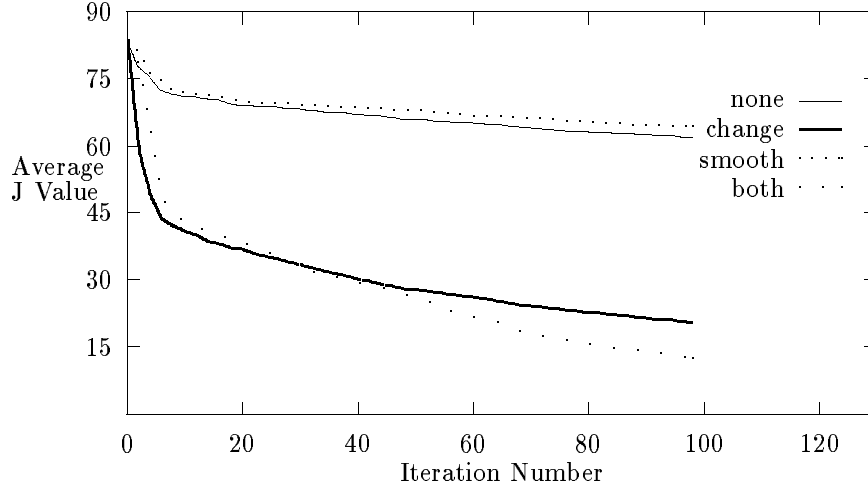


Figure 10: Comparison Alternatives with Task Information

Minimize

$$F = \sum_{i=1}^{joints} (x_{i,1}^2 + \sum_{j=2}^{timesteps} (x_{i,j} - x_{i,j-1})^2)$$

Subject to

$$G_1 = startpos_1 - finalpos_1 + \sum_{j=1}^{timesteps} x_{1,j} = 0$$

$$G_2 = startpos_2 - finalpos_2 + \sum_{j=1}^{timesteps} x_{2,j} = 0$$

Note that the final velocity in each of the two time sequences is a given (along with the start and final position). Using Lagrangian multipliers, it is straightforward to solve for the remaining joint velocities.

Each time we generate a new trajectory, we can immediately replace it with the optimal trajectory that produces the same end position and velocity. Doing this effectively allows the algorithm to search a 4-dimensional space (2 joints, position and velocity) instead of the 12-dimensional space that represents the size of a single trajectory. This technique is important since the most desirable algorithm is one that easily makes use of additional task information when available, but does not depend on its existence.

To demonstrate the usefulness of this reduction in the search space, the experiments described in Fig. 8 were repeated. The results are shown in Fig. 10. Again, the executions

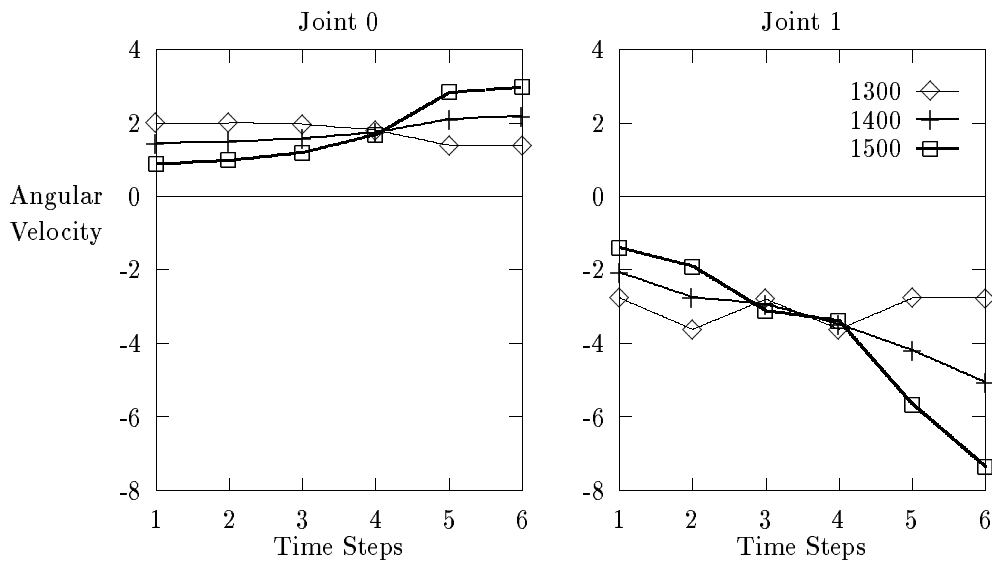


Figure 9: Final Trajectories

It is surprising that turning SMOOTH on without CHANGE does almost no good. Fortunately, when CHANGE is on, the addition of SMOOTH produces a significant improvement. It appears that both of those two methods are useful to the Practice Control Algorithm and the best result is obtained when they are used together. In this configuration an average reduction of effort of 86% is observed. It may still be necessary to test other common optimization techniques, but this method is likely to compare favorably considering the small number of robot trajectory executions required.

Fig. 9 is a sample of trajectories generated by the base and change vectors resulting after one of the tests with CHANGE and SMOOTH on. The generator was asked to produce trajectories that gave the same output distance as those in Fig. 7. A comparison of the two figures qualitatively shows how the second set of trajectories outperforms the first by using smaller accelerations. Note the smaller range on the y axis of Fig. 9.

2.4 Use of Kinematic Information

So far we have assumed no knowledge of the robot kinematics or of the task to assist the learning process. There are two pieces of information that can be used to help the learning process. The first is to note that the landing point of the ball depends only on the position and velocity of the end effector at the time of release. The second is that for a given final end effector position and velocity, it is possible to use the effort equation and the kinematic equations to get a closed form solution for the best trajectory. The following constrained optimization can be done:

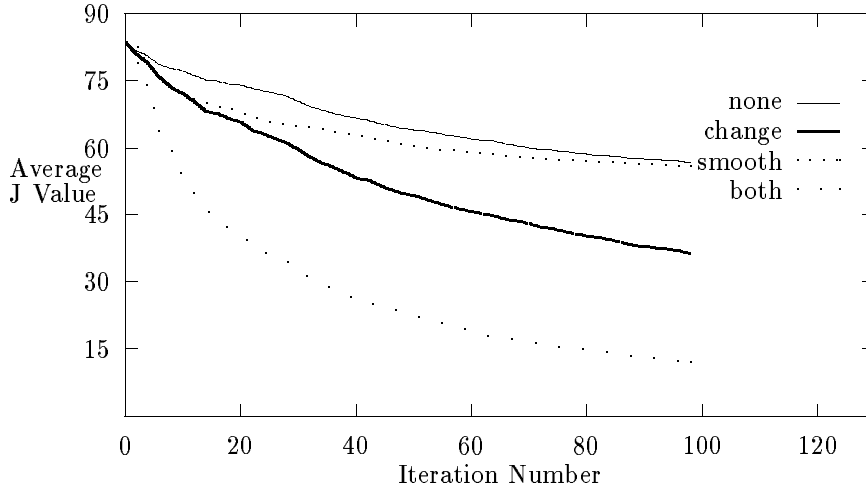


Figure 8: Comparison of Practice Algorithm Alternatives

(CHANGE)

- The use of velocity profile smoothing to lower effort level (SMOOTH)
- The use of the gradient descent described in reference to Fig. 4 to increase convergence speed (GRAD)

The ability to turn these on or off independently gives eight variations of the Practice Algorithm to test. Random trajectory generation was always turned on since that is the default method to use at Step 1 when none of the other options can be chosen. For each strategy combination 20 tests were run using different seeds to the random number generator. Each test began with the same initial base and change vector and ran 100 iterations of the main loop in the Practice Control Algorithm. For each test, the j value after each iteration was recorded and the average over 20 tests was computed. Fig. 8 is a plot of some of the results.

The first thing to be noted is that none of the four configurations plotted uses GRAD. The reason is that in every case the performance was almost identical to the same configuration without GRAD except that GRAD made it slightly worse. As it turned out, applying GRAD at Step 1 of the Practice Control Algorithm almost never resulted in improvement at Step 3. Therefore, turning this option on just wasted an iteration that could have been spent trying a new possibility. Since the random number generator was started with the same seed across all strategy combinations, the GRAD executions got the same random trajectories to try except they were delayed by the number of iterations wasted on GRAD. The disappointing performance of GRAD is not completely surprising for two reasons. The first is the nonlinear relationship between a trajectory and its corresponding point on a performance vs distance plot. Secondly, the direction used is not the true gradient direction. It is only an approximation based on the testing of a single direction in the space.

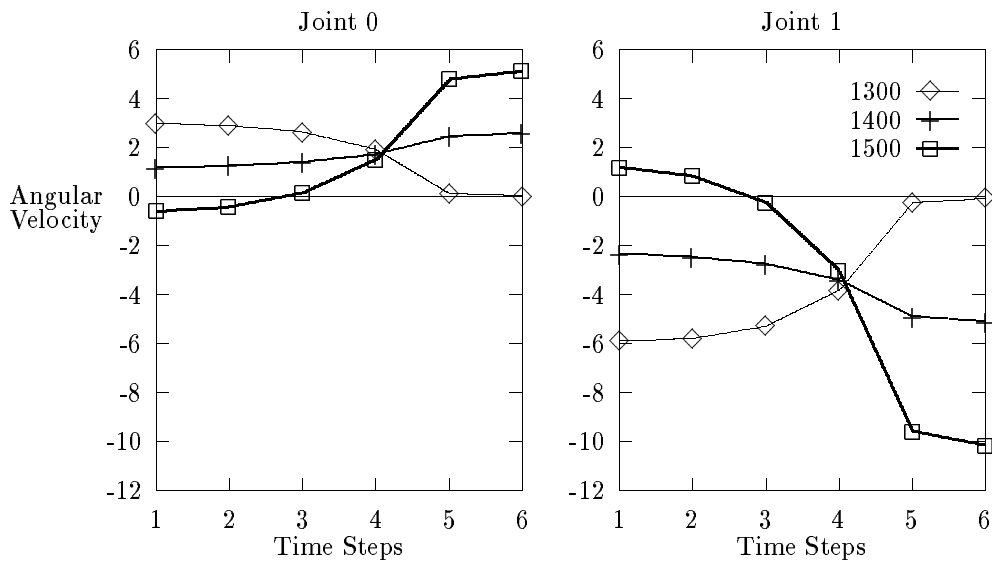


Figure 7: Trajectories from Initial Base and Change Pair

trajectories was 27%. That seems like a significant amount except that the goal of this first portion is just to get a reasonable initial base and change vector.

The real test follows with Steps 4-6 of the initialization. It is there that the system attempts to find an appropriate linear fit for the Weight Equation (eq. (2)) that maps task instances into weights for the change trajectory vector. After doing that, the Trajectory Generation Equation (eq. (1)) can be used to try some sample executions. Trajectories for a set of eleven different output values were requested. The system executed the trajectories and observed the resulting outputs. The actual outputs differed from the requested outputs by an average of less than 1%. This result shows that the linear approximation for the inverse function, G , could produce acceptable accuracy for the one-dimensional throwing task over a limited range. Fig. 7 shows three of the trajectories generated by the initial base and change vectors. The key lists how far the ball is thrown with the corresponding trajectory.

From the plot we can see that the trajectory labeled “1300” is likely to throw the ball only a short distance since it has high joint velocities initially, but slows nearly to a stop by the time of release. Alternatively, the trajectory labeled “1500” is likely to impart the greatest velocity on the ball since its joint velocities are increasing right up to the time of release. Also note that these trajectories are not likely to perform well against the objective function, H .

So far, the system has done nothing to optimize according H . Optimization is performed by the Practice Control Algorithm. There are three different strategies that can be turned on or off independently:

- The use of Steps 7 and 8 in the Practice Control Algorithm to alter the change vector

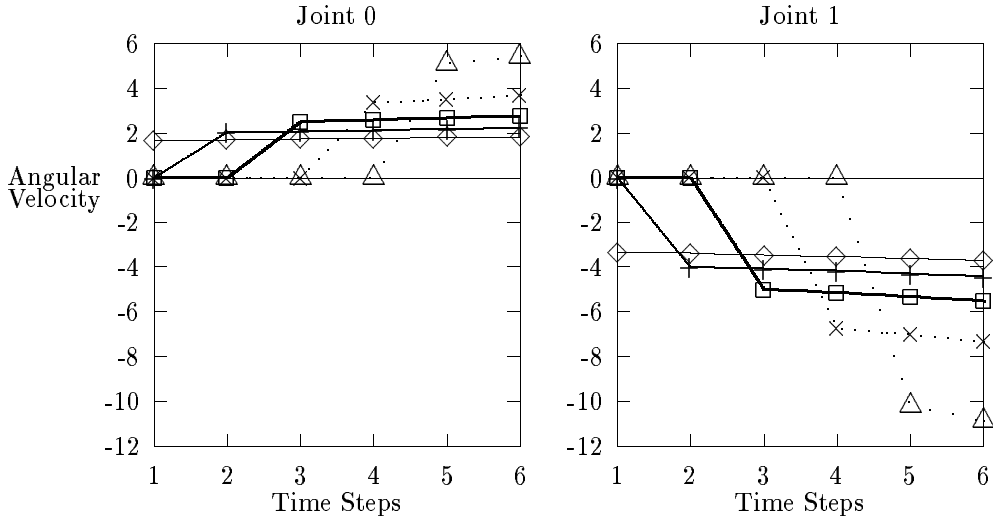


Figure 6: Initial Trajectories

direction required to use the *MIT Rule* from Model-Reference Adaptive Control [Astrom and Wittenmark, 1984]. Notice that point 3 does not fall along the line specified by point 0 and 1. Even though the three trajectories are colinear in trajectory space, they may not produce linear points in this graph because the transformation from trajectory to performance, \mathbf{j} , and to distance, \mathbf{y} , is not necessarily linear. This will be referred to as the GRAD algorithm.

2.3 Results of Experiments: Velocity Trajectories

First the initialization algorithm and the use of principal components to generate initial base and change vectors was tested. The five input trajectories are shown in Fig. 6. The lines on the left refer to joint 0, while the ones on the right are for joint 1. A heuristic of waiting longer and then jumping to a higher velocity was used to produce a sequence of trajectories that throw the ball a greater distance. Notice that the large single jump in velocity is likely to cause an excessively high value in the performance metric chosen.

This set was used as input to the initialization algorithm. The principal components analysis produced a base and a change vector as described. The base and change vectors were used in an attempt to reproduce the original five vectors.

$$\mathbf{x}_i \approx \mathbf{b} + v_i \mathbf{c}$$

where the i indicates which of the initial vectors and the v_i is a value determined by a least squares fit. The average error is defined by:

$$\frac{\sum_{i=0}^4 |(\mathbf{x}_i^* - \mathbf{x}_i)|}{\sum_{i=0}^4 |\mathbf{x}_i|}$$

where the star indicates the actual trajectories generated using the v_i and the unstarred variables represent the originals. The average error over all the coefficients of the five initial

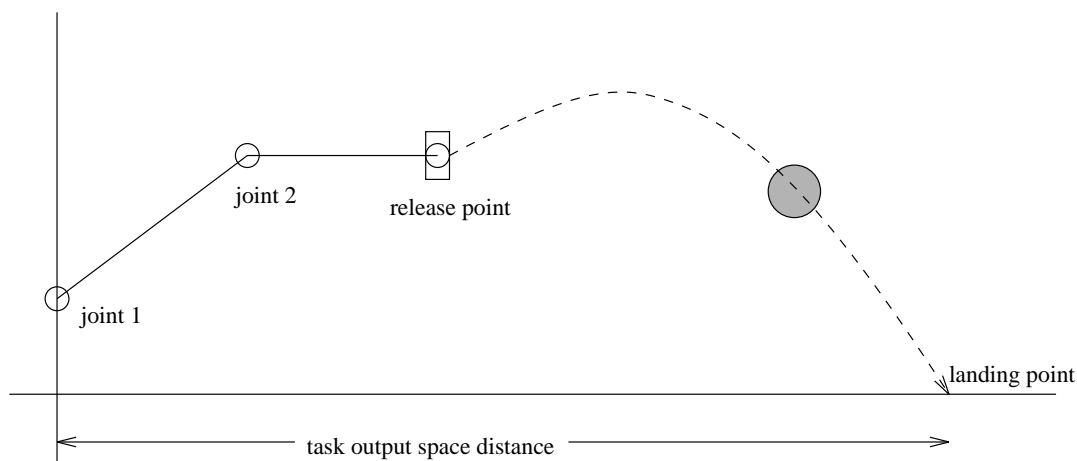


Figure 5: Diagram of a One-Dimensional Throwing Task

where i ranges over the six time steps, j ranges over the two joints and the conversion between the two subscripts of x and the single dimension of \mathbf{x} is as expected. The term “approximate amount of energy” is used because the metric actually sums the squares of accelerations. Computing the torques, and thus the actual energy, necessary to achieve those accelerations would include an analysis of the robot’s dynamics. The simulator is capable of doing the dynamic computations, but they were not done because of execution time considerations. Note that the term “performance metric” is slightly deceiving since lower values are better.

Step 1 of the Practice Control Algorithm requires that a new trajectory be generated. Several schemes have been implemented for this. The simplest is to generate randomly a modification of some previously generated trajectory. The robot remembers all its past performances and just chooses one at random to modify.

Another possibility is to use knowledge about the objective function to suggest good heuristics for trajectory improvement. For instance, the current objective function measures squares of accelerations. One obvious heuristic is to take a previous trajectory and smooth its velocity profile to reduce accelerations. For each element of a trajectory, x_{ij} , compute a new value with the formula:

$$x_{ij} = (2x_{ij} + x_{i-1,j} + x_{i+1,j})/4.0$$

where i is a time step and j is a joint number. For this computation, assume that all joint velocities are 0 at time -1 and drop the $i + 1$ term for the last time step. The hope is that the \mathbf{j} value can be reduced without a bad effect on the result, \mathbf{y} . This will be referred to as the SMOOTH algorithm.

Finally, it is possible to implement a crude form of gradient descent. Suppose a switch has just been made to scheme D in Fig. 4. At the next iteration consider point 0, the trajectory from curve B that produces the same output as point 1. Take the difference between these two trajectories and add it to point 1 to produce point 3. Point 3 will be the new trajectory to be tried at Step 2. The difference vector obtained by subtracting the trajectory at point 0 from the one at point 1 is an approximation to the gradient

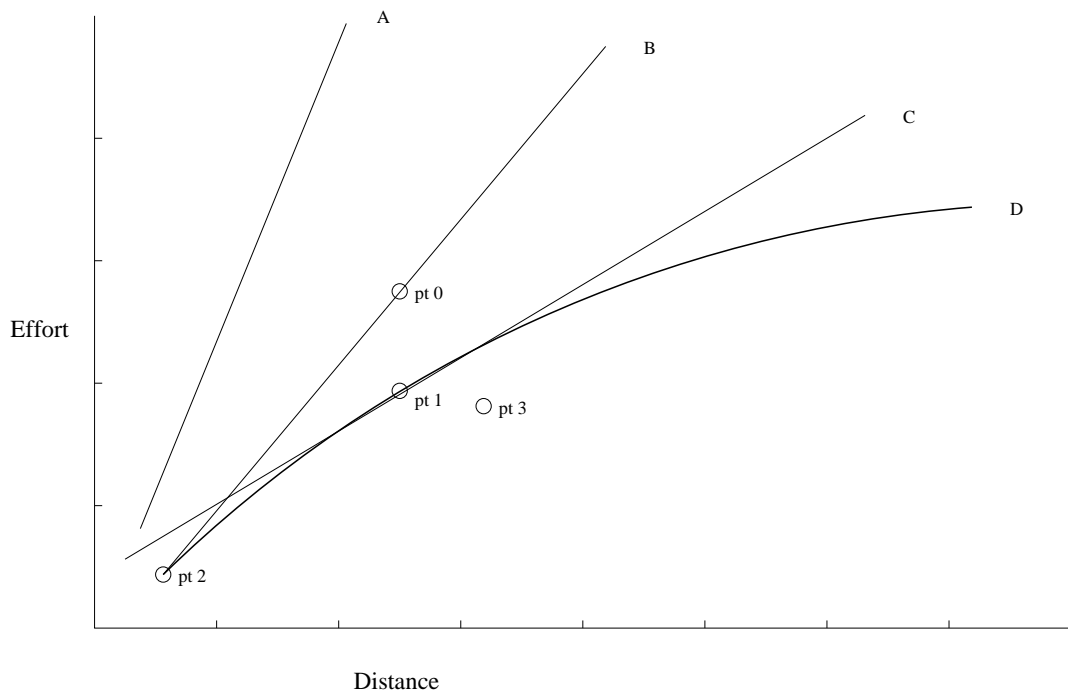


Figure 4: Performance vs Distance for 4 Base and Change Vector Pairs

ball always lands on a single line extending outward from the base of the robot. The length of its two links is 1000 and the first one is fixed to the base at a height of 100.

The output space vector, \mathbf{y} , is just a scalar in this task. It is the distance the ball lands from the base of the robot. Initially, the trajectories were represented as sequences of joint velocities executed in six time steps. (Later sections describe experiments using joint torques and spring models.) At each time step the joint velocities are set to a new value. Trajectories of this form are not physically realizable, but can be approximated and are easy to simulate. Since the robot has two joints and the trajectories span six time steps, a total of twelve coefficients are necessary to represent a single trajectory. This representation means that \mathbf{x} can be thought of as a six by two array with one dimension indicating joint number and the other indicating time step. A typical array might look like:

$$\{\{1.2, 1.4, 1.6, 1.9, 2.1, 2.5\}, \{-2.3, -2.8, -3.4, -4.0, -4.7, -5.5\}\}$$

In Figure 5, this trajectory would cause link 0 to accelerate downward at a low velocity, while link 1 accelerates upward at a higher velocity. For this task the starting point of the arm is assumed to be fixed and not subject to modification by the learning scheme. When computing the flight of the ball, the simulator considers only the effects of gravity and the initial velocity vector given to the ball at release. Among other things, air friction and spin are ignored. The performance metric is based on the approximate amount of energy required to execute a trajectory:

$$\mathbf{j} = H(\mathbf{x}) = \sum_{j=0}^1 (x_{j,0}^2 + \sum_{i=1}^5 (x_{j,i} - x_{j,i-1})^2) \quad (4)$$

That indicates that one base/change pair is better at producing trajectories for one range of distances while the second is better for another range. A metric must be defined that allows trajectories to be compared on a single dimensional scale so a choice can always be made. The current metric is to determine the average value over the length of the curve. A simple algorithm to improve the performance level is presented below.

Practice Control Algorithm

1. Generate a new trajectory that might not be possible from the existing base and change combination; call it x' .
2. Execute the new trajectory on the robot and observe the results
3. If the new trajectory performs worse than the trajectory that would be generated under the old scheme then go to Step 1
4. Replace the old base vector with the newly tested vector; $b = x'$.
5. Using the same change vector, generate a small set of trajectories and test them with the performance metric, J .
6. If the new set is better for all tested points in the output space accept the new base vector, keep the old change vector, and goto Step 10
7. Find some point on the old scheme that outperforms the new one, x'' , and set the change vector to the difference between that vector and the new base vector; $c = b - x''$.
8. Again, generate a small set of trajectories and test them for amount of effort
9. Using an overall goodness criterion choose between the original generation scheme, the one tested in Step 5, and the one tested in Step 8
10. Go to Step 1

The execution of this algorithm is better explained by referring to Fig. 4. It is a performance vs output graph as described above. Each separate line represents an entire class of trajectories that are generated by a pair of base and change vectors. Suppose that the initial base and change vector combination produces curve A . At some iteration of the algorithm we get to step 5 and it generates curve B . This improvement has been realized by changing the base vector only and we proceed to step 10. As a second example, suppose we already have curve B and we have randomly created a trajectory that falls at point 1. Substituting this as the new base vector we generate curve C . Note that curve C is better than curve B in most places, but not at the far left. Now Step 7 generates a new change vector by computing the difference between the trajectory that produced point 1 and the trajectory for point 2 as generated by the base and change combination from curve B . At step 8, curve D is produced with the new base and change vectors. In this example curve D is clearly the best, but the choice at Step 9 may not be this easy. Therefore, the choice is made by integrating over the entire curves to see which has a better average.

2.2 One-Dimensional Throwing Task

To experiment with the system, a simulator was used to perform one-dimensional throwing as shown in Fig. 5. The simulated robot has two coplanar joints, therefore the simulated

ones given is used. The normalization ensures that the first component from the resulting analysis can be effectively used as the change vector in the Trajectory Generation Equation. Steps 4-6 calibrate the system by using a few sample executions to choose an initial setting of m and b for the Weight Equation (eq. (2)).

Initialization Algorithm

1. Select one of the samples and make it the base vector
2. Normalize all vectors by subtracting the base vector from them
3. Run a principal components analysis and choose the first component to be the change vector
4. Regenerate a set of trajectories using the base vector and a range of weights on the change vector
5. Enter practice mode and record the output achieved when each trajectory is performed by the robot
6. Fit a line to obtained output data vs. change vector weight

Running the above algorithm sets all the necessary parameters to begin operation. The system is now ready to begin operating in either of the two main execution modes. Note that after having the robot perform only a small number of actions we expect to have a generator that produces reasonable trajectories over some of the output space range.

During operational mode goals are presented from an external source and the generator produces trajectories for the robot. The Practice Control Module is turned off during this mode. The modification module concerns itself only with accuracy and therefore ignores the effort data coming from the robot except to record it in a table. With each new data point (output data vs change vector weight), Step 6 of the initialization algorithm is repeated to improve the accuracy of the Trajectory Generator. In operational mode only the m and b parameters of the weight equation are changed, while practice mode may also change the \mathbf{b} and \mathbf{c} vectors. One drawback of adaptive systems is that they trade in optimality for adaptability. Operation mode does the equivalent to turning off the adaptability with respect to effort in order to improve performance with respect to accuracy. Intuitively, it is equivalent to the professional golfer who is unwilling to try a new swing in the middle of a tournament.

Practice Control Module and Practice Mode

During practice mode, the modifier attempts to minimize the objective metric \mathbf{j} that will likely include the amount of effort being spent by the robot. Given a set of base and change vectors, \mathbf{b} and \mathbf{c} , the \mathbf{j} value is completely determined for any realizable output distance, \mathbf{y} . Therefore, the modifier must alter the base and change vectors to realize an improvement in \mathbf{j} . Note that we can build a plot of \mathbf{j} vs \mathbf{y} if we are given a base and change vector pair. Each point on the graph represents a single trajectory and indicates the task output space result and the performance level, \mathbf{j} , attained by the robot. We want to adjust the base and change vectors such that we obtain the lowest curve in the \mathbf{j} vs \mathbf{y} graph (the performance metric has evolved such that lower values are more desirable). When we compare two curves it is possible that neither is strictly “lower” than the other (as in curves B and C in Fig. 4).

and the change vector is used to vary the output trajectory over the range of the output space. A trajectory is computed by finding a weight for the change vector and adding it to the base vector. This is done with the Trajectory Generation Equation (eq. (1)), \mathbf{x} and \mathbf{y} are the same as in the previous section):

$$\mathbf{x} = \mathbf{b} + w\mathbf{c} \quad (1)$$

where \mathbf{x} is the resulting trajectory vector, \mathbf{b} is the base vector, \mathbf{c} is the change vector and w is a weight determined by the desired output. The function that maps desired positions in output space to a weight is linear. It is called the Weight Equation (eq. (2)):

$$w = m\mathbf{y} + b \quad (2)$$

where w is from the previous equation, \mathbf{y} is the desired output and m and b are the linear function parameters. When a goal is submitted to the generator, it first determines the weight for the change vector. With the weight, it can generate a trajectory, \mathbf{x} . The equations are written separately because the determination of m and b happens separately from the determination of \mathbf{b} and \mathbf{c} . When combined, the inverse function G takes the form:

$$\mathbf{x} = \mathbf{b} + (m\mathbf{y} + b)\mathbf{c} \quad (3)$$

Generator Modification Module

The real work is done in the Generator Modification Module. It is here that the base and change vectors are determined and the two linear parameters mapping \mathbf{y} to w are found. This module runs two completely separate algorithms depending on whether the system is in operational or practice mode. In addition, it has a startup procedure that allows it to initialize a base and a change vector given input from an instructor.

The Modification Module generally receives data that allows it to compute H , the goodness of a task solution. In the example throwing task, two types of data are used. The first is effort data obtained from robot proprioception. This represents some measure of the energy expended to execute a given trajectory. The second type is accuracy data, simply the Euclidean difference between desired and actual output. All of the data is stored so it may be used at any later time during the skill acquisition process.

During startup, the system must be presented with at least two sample trajectories that produce differing results in the output space. With these samples the system executes the initialization algorithm listed below in practice mode. Steps 1-3 run a principal components analysis of the input vectors to derive an initial base and change vector combination. The decision to use only one change vector implies that only the first principal component will be taken from the analysis. Because of this, it is crucial that we normalize the vectors by subtracting one of them from the others. Often the normalization done before principal components analysis uses the average of all the samples. Using the average here is risky since the average might not represent a trajectory that produces a valid result in task output space. The practice algorithm presented later requires a valid base trajectory, so one of the

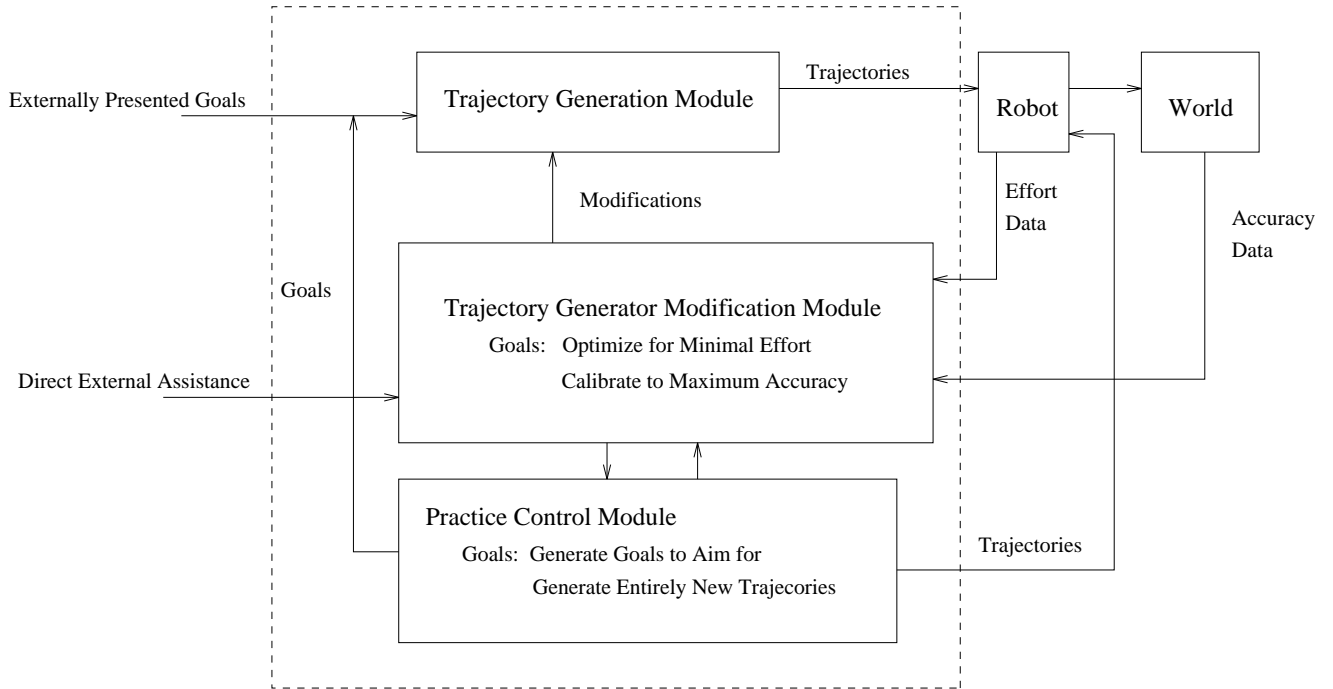


Figure 3: Robot Motor Skill Acquisition System

be). While in operational mode, modifications are only made to increase accuracy. While in practice mode changes may be intended to increase accuracy or to reduce robot effort. The difference arises from the assumption that we do not want to risk accuracy in a possibly unsuccessful attempt at effort reduction while running “executions that count.” It is similar to an athlete’s behavior on game day vs. at practice. The Practice Control Module only operates during practice mode. Its job is to work with the Modification Module to produce goals and trajectories that will be most useful in the optimization process. The following sections give a more specific description of the mechanisms to be used in each module.

Trajectory Generation Module

Given a trajectory, the robot performs its current approximation to the forward motor skill function, F , which produces some result in output space. The Trajectory Generation Module needs to compute the inverse function. There are numerous mechanisms for approximating inverse functions given a method of computing the original function. Some options are neural network approaches and lookup tables. These options have one serious drawback. The number of iterations required to get reasonable results is often prohibitive when dealing with real, mechanical systems.

One of the key simplifying assumptions in the work reported here is that of a one-dimensional output space. Given a desired scalar output value the skill is to produce a parameter trajectory vector. The current linear mapping from goals to trajectories finds an inverse function quickly. The parameter trajectory is generated from two vectors, called the base and the change. Intuitively, the base vector indicates how the task is done in general

inverse (so far restricted to a linear mapping), and must fill in all the parameters through experience.

Sutton’s DYNA architecture [Sutton, 1990] is an example of an approach that assumes CPU computation is cheap and interaction with the world is expensive. Sutton combines the ideas of reinforcement learning with the acquisition of a world model for planning. His system runs in two modes. While interacting with the world, it runs standard reinforcement learning and acquires a world model. Then it can run in a second mode that replaces world interaction with its world model. Interaction with the world model allows it to do a large amount of relatively cheap CPU computation to improve its action policies. The main difference between this and task-level learning is the form of the world model. In general, the unknown world confronting the task-learning robot described here contains the robot’s own kinematics and the structure and physics of the task domain. Any knowledge of kinematics or dynamics can be given directly to the robot controller. The other part of the world, the physics and structure of the task domain, is much harder to model than in the DYNA formalism. DYNA assumes it has a finite number of distinguishable states with a finite set of possible actions. In our domain, the choice of actions is multi-dimensional and continuous. This makes it difficult to build up an accurate world model. In DYNA’s terms, our approach is to set the policy (inverse: maps goal states to actions) function without using any model of the external world (forward: maps actions to states).

2 A System for Skill Acquisition

This section describes a simulated skill acquisition system and the results of experiments run with it. The skill to be acquired is a robot throwing a ball. The additional constraint is that the system should try to minimize robot effort over a range of throwing distances. First, the general algorithms are described and a high level description of the system operation is given. Then, the throwing task and the algorithmic parameters specific to it are described. Finally, the results of experiments are presented.

2.1 The Basic System

Fig. 3 is the block diagram for the basic system. The large dashed box represents the “skill,” “interpreter,” and “learning” boxes from Fig. 2. It consists of three tightly integrated modules and runs in two modes. The *operational mode* takes goals from an external source and produces trajectories that are expected to accomplish the goals (i.e. it computes the function G^*). The trajectories go to the robot where they are executed and the results are observed. The *practice mode* can be entered to improve performance more actively. Here, goals and/or trajectories are generated internally with the specific intention of improving performance. Again the robot is operated and the results are observed. The Trajectory Generation Module is for computing the inverse task function, G^* , and performing the interpretation into control signals. The goals come from an external source during operational mode and from the Practice Control Module during practice mode. The output trajectories always go to the robot. The Generator Modification Module is responsible for observing the robot’s performance and making changes to the generator (i.e. it determines what G^* will

HMM formalism, but rather a completely deterministic one. It seems reasonable that as the performance of manufactured intelligent systems becomes faster and more complex, there will be a need for the sort of fast and tunable behavior represented by motor skills of the sort we study. Already system architectures are being presented that have several layers, from low-level closed-loop servoing through reactive behaviors to high-level cognitive planning [Connell, 1992]. Learned skills are not necessary for biological survival or the accomplishment of incredibly complex sensori-motor feats (consider spiders). However, the importance of acquired skills to humans and higher mammals makes it clear that sensori-motor skills can play a central part in the repertoire of intelligent activity as well as day-to-day brute survival.

A large amount of work comes under the description “Trajectory Learning” (e.g. [Gelfand *et al.*, 1992; Simard, 1991]), whose goal is the acquisition of known trajectories. In each case some mechanism provides the learner copies of the trajectories it should acquire. Gelfand simulates the tasks of path planning with obstacle avoidance and basketball dribbling. In each case he has a simulated vision system that can already produce the desired trajectories. The goal is to train a CMAC [Albus, 1975] to produce the correct trajectories so the vision system can be turned off. Simard uses recurrent neural networks to store trajectories that are presented externally. The key differences between “Trajectory Learning” and task-level learning are two. First, the task-level learner has no knowledge of the explicit trajectories it must acquire. Second, trajectory learning is a zero-dimensional task.

A slightly different problem is to learn the control signals necessary to produce a known trajectory. One method is to learn through repeated attempts at producing the trajectory. At each iteration, the robot uses its sensors and the errors recorded in earlier trials to improve future trials.[Bondi *et al.*, 1988][Kawamura *et al.*, 1988][Qu *et al.*, 1991]. Other researchers have looked at the generation of classes of trajectories with a small set of basis functions [Hollerbach, 1981][Morasso and Ivaldi, 1982] and the smooth concatenation of trajectories [Lloyd and Hayward, 1991]. This work is relevant since it can be viewed as open loop trajectory generation. Again, as practiced so far the task has been zero-dimensional.

Research at MIT [Aboaf *et al.*, 1989][Atkeson *et al.*, 1988][Atkeson, 1991] [Branicky, 1991] seems closely related. The work by Branicky most resembles skill learning. It uses a throwing task with the goal of hitting a single target at a fixed distance (Again, a zero-dimensional task). He does, however, mention the need to generalize to higher dimensional tasks. In his formulation, he has chosen a robot and fixed the trajectory such that there is only one remaining variable to be adjusted. His search for the correct trajectory is thus in a one-dimensional space. This allows him to use numerical root finding methods to perform the search. The throwing experiments described below use robot arm trajectories specified by 12-dimensional vectors — a 12-dimensional search space. The algorithms used here impose no specific limits on the dimensionality of the space. Finally, Branicky starts with a model of the task. In the throwing case, he uses simple ballistic equations and the equations of his robot’s kinematics. What he must learn is to adjust for inaccuracies in the task modeling. This is similar to forms of adaptive control designed to deal with inaccurate plant models or time-varying plants. In the work described here no a priori model of the task is required. The acquisition of the inverse task equation does not depend on a learned or provided forward model. The task-learning algorithm begins with a generic form for the

5. quadratic
6. gaussian (velocity and spring)
7. kinematic optimal, given release point parameters (velocity only)

These functions were chosen after consulting the literature on goal-directed human and primate motions [Jeannerod, 1988], as well as to cover a standard set of low-order functions. As explained in more detail below, the quadratic basis functions were inspired by optimal velocity sequences found by a closed form solution to the optimal kinematics for a particular task instance.

The skill mapping method we have chosen is one that is linear in the various parameters. A skill uses two data structures, which are parameter trajectories similar to its output. One is called a base (not basis) trajectory and the other is called a change trajectory. The skill mapping adds a weighted version of the change trajectory to the base trajectory and the result is the output. The weight is linearly determined from the task parameter input. This linear underlying process was chosen in conjunction with the learning method, whose job it is to develop and improve the base and change trajectories. It was partially inspired by methods for optimal representation of image sets [Rosenfeld and Kak, 1976]. It also has the advantage of being intuitive and easily computable; other options such as neural nets and lookup tables (CMACS) are not so elegant, though they may have more power.

The skill mapping is learned initially from two or more examples provided by a teacher. The examples are subjected to a principal components analysis to provide base and change trajectories that fit the examples best given the linear nature of the skill mapping and its restriction to two trajectories. The skill mapping is improved by searching in trajectory space for better base and change trajectories, using various iterative improvement techniques that are described in Section 2.

It is not intuitively obvious that this simple linear mapping can succeed for arbitrary skills and robot configurations. For instance, the kinematics and dynamics of robots are usually nonlinear, even if the task does not appear to be. The task can easily involve nonlinearities (as does even the simple ball throwing task). One claim of this work is that nonlinear basis functions used in the interpretation process provide a solution to the “linear combination” problem. The learning algorithm can proceed with simple linear combinations of base and change vectors, relying on the transformation to actual trajectories to produce nonlinear results. The two main parts of this paper are a description of the learning algorithm applied to the throwing task, and a discussion of the effects of varying the basis functions used in the learning process.

1.4 Related Work

There has not been much research activity in AI circles on skill learning and performance. Some recent work in the acquisition and representation of explicit action sequences uses hidden Markov models (HMMs): see [Rimey and Brown, 1991b; Rimey and Brown, 1990a; Rimey and Brown, 1990b] and references contained therein for more motivation and history for such explicit representations. The work we describe below does not use the probabilistic

4. The function F is unknown. We may be given trajectories to try out (as in the training regime described later) and we assume a mechanism to observe the results of task execution and the various costs and benefits of the execution sequence. From the information we might attempt to build our own guess about what F is, but that guess can only be verified through additional trials. This leaves two possible avenues for solutions. One is to generate G^* without any concept of F , while the other would make use of a guess about F to get G^* .
5. Robots will have extra degrees of freedom of action. In the example to come, \mathbf{x} has twelve dimensions while \mathbf{y} has only one. Finding an \mathbf{x} that accomplishes a given \mathbf{y} may be relatively simple since many of them exist. On the other hand, finding an optimal \mathbf{x} is much harder since the space to be searched is huge even though the output space is small.
6. The cost of computation has a large influence on the choice of algorithms. The computation of F is expensive because it includes the operation of a physical device, the robot. Since the computation of F is our main source of data, it is apparent that we must make the best use of a limited amount of it. In comparison, the CPU cycles that process the data are much cheaper. An algorithm that requires millions of iterations for the CPU and a few robot trials is better than one that requires hundreds of robot trials and less CPU work.

1.3 Research Questions

The above background implicitly defines a research program by presenting certain technical questions.

1. What is a good choice of basis functions?
2. What are the characteristics of the skill mapping from task vector to parameter trajectories?
3. How is the mapping learned?

All these are open research questions, but in this document we have made some specific choices of answers. Our choices and their justifications are as follows.

We have explored several forms of basis function under three different assumptions of plant control. They are control via a timecourse of joint velocity commands, joint torque commands, and spring setpoint commands. The following basis functions have been tested.

1. delta function
2. step (velocity only)
3. constant (torque only)
4. linear (torque and spring)

Usually when a parameter trajectory specifies several basis functions, they are to be combined by addition, or superposition. Certain orthogonal basis functions (e.g. sinusoids as in Fourier analysis) can be combined to represent arbitrary functions to arbitrary precision. Coordinated action control commands are then represented by several trajectories, the outputs of the interpreter process, one for each SISO component of the plant, indexed on the same timeline.

The interpretation process can perform a much more complex computation than instantiating basis functions and combining them in a weighted sum. The interpretation may make use of kinematic information about the manipulator that requires iterative optimization methods. This might happen when one goal of the task is to find an “optimal solution,” but no closed form equations exist. For example, the equations given later in Section 2.4 may not be solvable in closed form for some tasks or manipulators. As another example, some robotic systems are controlled using a spring model where the control parameters are the spring constant and the setpoint. It may be desirable to mimic spring control for a plant that does not have it. To do so would require the interpreter to observe the current state of the plant to determine the correct torques to output. An example of a more complex interpretation process is given in Section 4.3.

Generally, actions are not costless and benefitless. We want task performance to be quantified along other dimensions than just achievement of the desired task parameters (for example, the task should be achieved as fast as possible, with minimal energy expenditure or maximum smoothness, grace, or even amusement value). We thus, in general, want a skill to optimize according to some objective function, $\mathbf{j} = H(G^*, F)$. G^* indicates the inverse function approximation the robot system has obtained as opposed to G , an actual inverse of F . H can include a variety of different terms. Usual metrics include average power required by the robot to perform the trajectories, accuracy of the inverse computation as measured by $\mathbf{y} - F(G^*(\mathbf{y}))$, the range of attainable values for \mathbf{y} , and direct functions on the form of G^* such as how long it takes to compute. Thus, we rate how well a system has acquired an open loop trajectory skill by the corresponding value of \mathbf{j} it has achieved. Of course redundant degrees of freedom are necessary to be able to optimize over anything but the “final error,” or accuracy of the achieved versus the commanded task parameters. In this usage, “degrees of freedom” refers to choices of actions, not necessarily to kinematic degrees of freedom.

Summary of Problem Properties and Assumptions

1. A skill is a mapping from an n-dimensional goal vector to a synchronized set of parameter trajectories, each describing (via the interpretation process) a timecourse of activity for a single actuator.
2. A parameter trajectory usually is made up of parameters for certain basis functions. The interpretation process generates the proper basis function instantiations, converts them to continuous functions, adds them if necessary, and the resulting sum is the input to a SISO controller for the actuator.
3. The skill mapping is learned from experience and from examples from a teacher.

A task output is specified by a parameterized description of desired output in some goal space: this description is also a vector, \mathbf{y} . For example if the task is to throw a ball, the vector might represent the landing position of the ball. Thus the learning we describe is “task-level learning,” not for example, explicit trajectory-level learning. The entire control system can be described by an input-output function, or a *forward function*, $\mathbf{y} = F(\mathbf{x})$. We may be able to write F down mathematically, but we only assume the ability to evaluate it for particular arguments. One way to evaluate F is to use a robot: If the task is to throw a ball, compute F by having the robot execute the trajectory \mathbf{x} and observe the resulting position of the ball, \mathbf{y} . The function F in general will involve a stochastic component, due to the inevitability of unrepeatability, noise, and error. We also define $G = F^{-1}$, or $\mathbf{x} = G(\mathbf{y})$ to be an inverse task function. The inverse function need not be unique and usually is not. Many task descriptions include an \mathbf{x} vector with more dimensions than the \mathbf{y} vector. That means the robot has redundant degrees of freedom of action. Combining all these definitions, a robot system is said to have acquired an open loop trajectory skill if it can accurately compute some G for the corresponding task.

A tunable robot skill accomplishes a task that is parameterized by one or more desired output values. The skill is called “ n -dimensional” if n is the number of parameters. A zero-dimensional task is exactly the same every time. Therefore, it is not tunable and does not fall under the definition of a skill used here. One example zero-dimensional task is the typical pick and place robot task: A factory robot that places windshields on a variety of different cars just performs a finite number of “zero-parameter” tasks. Each time the same model comes along, the robot’s motion is the same. This paper uses a simplified (one-dimensional) task of throwing a ball as an example. The tunable task parameter is the distance at which the ball should land. Having successfully acquired the skill means the robot can accurately perform over a range of task parameters (distances).

If the parameter trajectory is a sequence of joint velocities (thus delta function basis functions) its interpretation in terms of the desired velocity profile is explicit. However, the parameters in the trajectory may describe more general *basis functions*, which themselves are timecourses of quantities simply related to plant control, such as velocity or torque. The interpreter maps these parameters into controls. Basis functions have the potential of being reservoirs of “innate” (inherent, implicit) knowledge. For instance, a parameter trajectory of a single 3-tuple can encode a continuous sequence of joint velocities: the 3-tuple can give the mean (time of maximum), variance, and amplitude of a gaussian or the parameters of a quadratic curve, thus specifying a velocity for any time on the real line. This information is extracted by the interpretation process (given the interpretation process, the trajectory is an explicit representation of action). If the basis functions are provided to the system, they are a significant contribution of useful knowledge. On the other hand, it may be possible to learn useful basis functions, either general or task-specific. Arbitrary basis functions have different expressive power. Each basis function defines a subset of trajectory classes that can be represented with that basis function and the linear generation scheme. Differences in expressive power can help in two ways. First, it may be restrictive and limit the search to only those trajectory classes that are appropriate for your task (i.e. speed up learning by reducing the dimensionality of the search space). Second it may extend the power of the trajectory generation process beyond the limits imposed by the simple linear rules of combination that pervade the skill acquisition system described later.

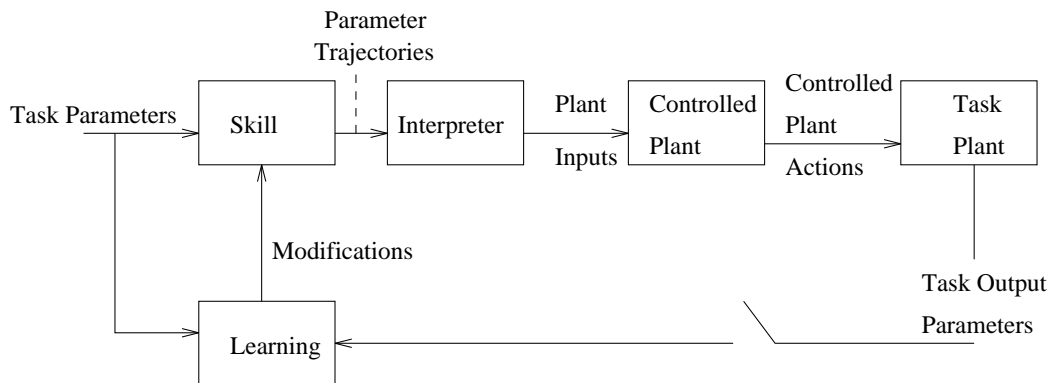


Figure 2: Same as Fig. 1, only explicitly showing the interpretation process.

plant’s output is the input to the task plant, and the output of the task plant can be interpreted in terms of the task input parameters to check for agreement. For instance, the task may be to putt a golf ball into a cup; the golfer and putter is the controlled plant, the golf green, the ball, wind, gravity, and possibly a seagull make up the task plant, and the final position of the ball is one of the task parameters that can be used to assess success (ball velocity near the hole is another parameter, useful for learning or aesthetic judgements). The goal of this work is to develop methods of representing skills, of learning them, and of tuning them to particular tasks.

A general way to describe skill output, which is the input to the controlled plant, is as a set of timecourses of activity, or set of temporally indexed sequences [Rimey and Brown, 1991a]. Therefore we describe a skill as a coupled set of *parameter trajectories*, sometimes called *trajectory vectors* or *trajectories*, one trajectory for each effector involved in the action. In this document the word “trajectory” never means “flight path.” We shall assume that each sequence in the set is converted by an interpreter into input to a SISO controller in the controlled plant, including the “null interpreter” and “null controller” that just pass the sequence along.

This interpretation step complicates the system diagram. Fig. 2 shows that the interpreter of parameter trajectories produces inputs to the controlled plant. Part of the contribution of this work is to discover parameter trajectories that are adequate to characterize the desired tunable skills, and corresponding interpreters to mediate between the trajectories and the controllable plant.

The trajectory is a finite sequence of n -tuples, \mathbf{x} . Usually each n -tuple is a set of parameters for *action basis functions*. For example, if the n -tuples are singleton values of joint velocities, the basis functions are delta functions, and the semantics of the trajectory is that of a discrete sequence of joint velocities. Depending on the controller of the plant, this representation may suffice for input, or perhaps the interpreter must interpolate the values with a continuous function, yielding a set of continuous joint velocity profiles, or possibly even the interpreter must go farther and convert the velocity profiles to torque inputs. A trajectory vector may also include a specific starting point or may be relative — the trajectory is executed from the current position.

knowledge of the plant to be controlled. Control theory includes learning in various forms under the general heading of “adaptive control”, and includes knowledge-based switching and melding of controllers under the general heading of “fuzzy control”. There is a staggering amount of formalism and mathematical tradition behind control theory.

- Reactive or reflexive systems [Brooks, 1987; Brooks, 1991] are built of combinations of innate open loop responses with various sorts of simple interactions (most notably subsumption, in which one reflexive action takes priority over another and inhibits it). Reactive behaviors can be learned (the Aeye system learns how to stabilize its gaze against head movement, for instance), but usually they are innate. Most reactive behaviors, at this writing, are SISO (single input, single output).

The motor skills we envision are of the following sort. Throwing, putting, bowling, and remote interception tasks generally; multi-arm cooperation in carrying or transferring loads, and curve-following parameterized by speed or weight of load. So far we have restricted efforts to one-dimensional tasks — those parameterized by a single number, for instance distance or weight.

Skills are closest to reactive behaviors, but they differ in quantitative and qualitative ways. A skill is a commanded, not a reflex, action. A skill may be built on pre-existing innate capabilities but it inherently has a component that is learned through practice. A skill is open loop by definition: closed-loop activity is thus by definition unskilled. The reason is that skills require too much speed and must be too smooth for feedback control. Skills are “over-learned,” in psychological terms. They are often learned while performing the commanded task under feedback control, but ultimately run open loop. Compare signing your name with copying someone else’s signature, or learning to play the piano with playing the piano. The learning component is also necessary to cope with aspects of the task that may be completely unknown a priori. As will be seen, our definition of skill acquisition allows use of prior knowledge about the controlled plant or uncontrolled aspects of the task but does not require them. A skill in general is MIMO. Thus the command to initiate the skilled behavior may have several dimensions. For example, hitting a golf ball in a commanded azimuth direction for a commanded distance on a commanded ball-flight altitude angle with a commanded amount of backspin (four dimensions). Likewise the output of the skill controls several effectors (so many that it is hard to estimate for the golfing example).

1.2 Assumptions and Definitions

This work assumes a controlled plant that has multiple interacting degrees of freedom, actuators, or capabilities (for example a robot arm with several actuated links). We assume that the task for which the skill is to be learned is parameterized: that is, the skill is to produce any of a family of solutions, as dictated by the particular task instance parameters (for example, being able to intercept or hit a target moving in some range of velocities or in some range of positions). The skill is the mapping that takes the task instance parameters as input and produces a representation of commands to the plant as output. The controlled

1 Motivation, Assumptions, and Definitions

1.1 Motivation

We present a computational, constructive theory of *tunable, open loop trajectory skills*. These skills are flexible in that they may be specialized to different task instances, and efficient, in that they run faster than closed-loop control would allow. A tunable open loop trajectory skill (henceforth skill) is an open loop, MIMO (multiple input, multiple output) controller whose inputs describe a motor task and whose outputs achieve the task. The task is parameterized by one or more desired output values. Aspects of the controller may be innate (given, hardwired), but to count as a skill, some aspects must also be adaptable to improve performance. The task is accomplished by the controlled plant, whose characteristics may be known a priori or not, with an uncontrolled and possibly unknown “task plant”, whose inputs are the controlled plant’s outputs and whose outputs are the task results. Fig. 1 shows these components, and also how a discrete-time feedback process comparing task output to task input parameters is used for one form of skill learning. (Another form of learning might be that sample task parameter and skill outputs are provided by a teacher).

This general definition will be constrained in various ways in the following report, reflecting certain decisions, experiences, and tastes of the authors.

Skills may be contrasted with three other paradigms that are well-known in artificial intelligence: planning, control, and reactive systems.

- Planning, as usually conceived and practiced in the AI community, is open loop, off-line, and cognitive (symbolic), using innate (not learned) abilities. There are some exceptions, but the planning community generally is more interested in issues of abstract reasoning than with the efficient accomplishment of low-level actions. Planning researchers have a certain amount of formalism at their disposal, of which logics are probably the most important component.
- Control theory is usually concerned with closed-loop systems and such issues as stability and generation of appropriate closed-loop response. Usually control theory assumes

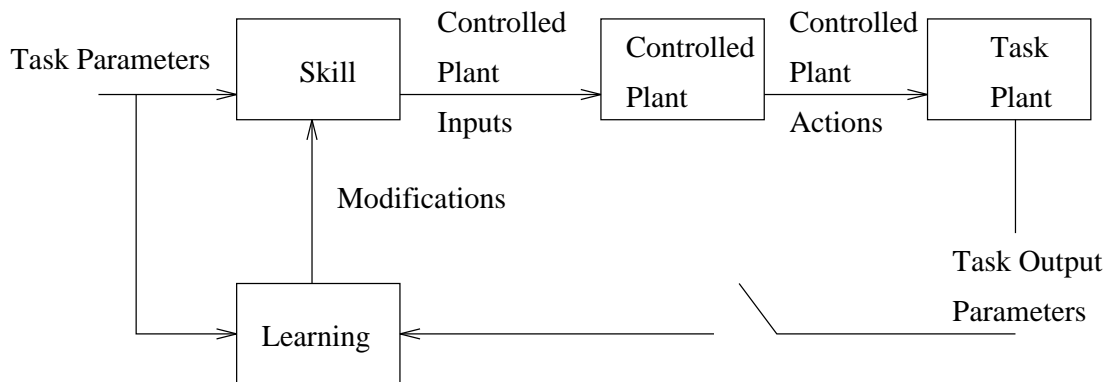


Figure 1: Components of motor skill, showing feedback for skill learning. Inputs and outputs are in general multi-valued.

Robot Skill Learning and the Effects of Basis Function Choice

J. G. Schneider and C. M. Brown

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report 437

September 1992

Abstract

We present a computational, constructive theory of *tunable, open loop trajectory skills*. A skill is a controller whose outputs achieve any task in a space characterized by n parameters, $n > 1$. Throwing a ball at a target is a 3-dimensional task if the target may be anywhere within a 3-dimensional volume. Repetitious pick and place tasks are zero-dimensional, and thus not skills. Skills are performed open loop for speed reasons: we assume the entire command sequence is generated before any feedback can become available. We do not assume prior knowledge of plant or task models, so skills must be at least partly learned. A skill output is a vector of values – in our work so far it is generated as the sum of a base vector and a weighted change vector whose weight accomplishes the tuning. Learning consists of a search for the best set of base and change vectors. An interpretation process maps skill outputs into sequences of commands for the plant by using basis functions (given a priori in this paper). The basis functions may be arbitrarily complex. We claim that appropriate basis functions can speed up the learning process and overcome the limitations of the linear trajectory tuning algorithm. This report describes a skill learning algorithm and experiments done with various basis functions and control methods for a one-dimensional throwing task. It concludes with a discussion of future work in learning basis functions, higher dimensional tasks, and comparisons against common learning and control algorithms.

This material is based on work supported by the National Science Foundation under Grants numbered IRI-8920771 and CDA-8822724, and by DARPA contract MDA972-92-J-1012. The government has certain rights in this material.