

Learning Dextrous Manipulation Skills Using the Evolution Strategy*

Olac Fuentes and Randal C. Nelson
Computer Science Department
University of Rochester
Rochester NY 14627
{fuentes,nelson}@cs.rochester.edu

Abstract

This paper presents an approach based on the evolution strategy for autonomous learning of dextrous manipulation primitives with a dextrous robot hand. We use heuristics derived from observations made on human hands to reduce the degrees of freedom of the task and make learning possible. Our system does not rely on simulation; all the experimentation is performed the 16-degree-of-freedom Utah/MIT hand. We present experimental results that show that accurate dextrous manipulation skills can be learned in a period of a few minutes. We also show the application of the learned primitives to perform an assembly task.

1 Introduction

A dextrous manipulator is a robotic system composed of two or more cooperating serial manipulators. Dextrous manipulators have potential applications in areas such as prosthetics and space and deep sea exploration where a single robotic system will be required to perform a variety of tasks and thus versatility rather than precision is the main requirement. However, programming these robots to solve complex tasks in the real world has remained an elusive goal. The complexity and unpredictability of the interactions of multiple effectors with objects is an important reason for this difficulty.

In complex and hard-to-model situations, such as dextrous manipulation, it would be desirable if the behaviors or skills exhibited by the robot could be learned autonomously by means of the robot's interaction with the world instead of being programmed by hand. However, machine learning of robotic tasks using dextrous manipulators is extremely difficult, due mainly to the high dimensionality of the parameter space of these robots. Conventional approaches to this problem face the well-known "curse of dimensionality" [3], which essentially states that the number of samples required to learn a task grows exponentially with the number of parameters of the task. Another problem is that autonomous experimentation with real robots is expensive both in terms of time and

equipment wear. For these reasons, most applications of machine learning to robotics have dealt with simple robots, and have concentrated on simple tasks with a few discrete states and actions.

A commonly used approach to make robot learning feasible despite the high dimensionality of the sensory and motor spaces is to run the learning algorithms using simulated environments. However, in situations involving complex robots and environments, it is difficult or impossible to gather enough knowledge to build a realistic simulation. Moreover, some physical events such as sliding and collisions are difficult to simulate even when there is complete knowledge. For these reasons, we believe that for the learned skills to be applicable by the physical robot in its environment, much of the learning and experimentation has to be carried out by the physical robot itself. Given the high cost of real-world experimentation, for the learning algorithms to be successfully applied, it is crucial that they converge within a reasonable number of trials.

Observations made on human hands offer some clues about how to deal with the problem of the high dimensionality of the parameter space of dextrous manipulators. Arbib *et al.* [1] introduced the concept of *virtual fingers* as a model for task representation at higher levels in the human central nervous system. In this model, a virtual finger is composed of one or more real fingers working together to solve a problem in a task. The use of virtual fingers limits the degrees of freedom to those needed for a task, rather than the number of physical degrees of freedom the hand, human or robotic, has. Iberall [9] has shown how the hand can be used as essentially three different grippers by changing the mapping of virtual fingers to real fingers. A generalization of virtual fingers, called a *virtual tool* [6, 14], has been proposed as a way of dealing with the redundant degrees of freedom of complex robotic systems.

An object translation by a human hand using a precision grasp, that is, a grasp where the only contacts occur at the fingertips, can be viewed as the action of two virtual fingers moving in the direction of the translation while maintaining a roughly constant force applied to the object. In general, the thumb will constitute one virtual finger, while one or more of the remaining four fingers work in conjunction and form the other virtual finger.

Using the virtual finger abstraction, the dimensionality of

*This material is based upon work supported by ONR grant N00014-93-I-0221 and NSF IIP Grant CDA-94-01142

learning a manipulation task is greatly reduced, since the task of the learning method is to find the required commands to the virtual fingers, instead of direct commands to physical actuators. Meanwhile, the mappings from virtual to real fingers can be learned as a separate problem or be provided by a human.

In this paper we present an approach for autonomous learning of dextrous manipulation skills that uses the concept of virtual fingers to limit the dimensionality of the search space. The approach consists of first learning and storing a few basic manipulation primitives for a few prototypical objects and then using a nearest-neighbor method to compute the required parameters for new objects and manipulations. The primitives are learned using a modified version of the evolution strategy, which allows us to deal with the noise normally present in tasks involving complex interactions between a robot and its environment. Our system does not rely on simulation or modeling; instead, all the experimentation is performed by the physical robot. In Section 2 we describe the learning method in detail, Section 3 presents experimental results using the Utah/MIT hand, and Section 4 discusses salient features of our approach and directions for future work.

2 Learning Manipulation Skills

The system first learns a set of primitives for performing basic translations and rotations of several different objects. New manipulations can be obtained by scaling and adding or subtracting the primitives. This method is similar to Speeter’s *motion primitives* [17], the main difference being that in his system the primitives were supplied by the programmer, while in ours they are learned automatically.

Normally, a primitive would consist of the changes in joint angles of the hand that are required to perform the desired manipulation; however, using the virtual finger observation, as explained in Section 1, we command identical changes to corresponding joints of each of the real fingers that are coupled to form a virtual finger. Essentially, the system learns to manipulate objects using two 3 degree-of-freedom virtual fingers, while the programmer provides the mappings from virtual finger parameters to the joint angles of the particular robot used. Besides efficiency, this has the advantage of making the learning mechanism manipulator-independent.

We assume that the programmer provides a *perceptual goal* in the form of the sensor readings that will be observed at the goal position. In the case of dextrous manipulation, a perceptual goal is given by the desired final position and orientation of the object and the forces applied by each of the fingers.

A perceptual goal has the form $\mathbf{g} = [x, y, z, \alpha, \beta, \gamma, p_1, \dots, p_n]$, where x, y, z encode the position of the object in 3-dimensional Cartesian space, α, β and γ are the Euler angles azimuth, elevation and roll defining the object’s orientation with respect to a hand-centered coordinate system, and p_1, \dots, p_n are the readings in the tactile sensors located at each of the n fingertips.

Let \mathbf{g} be a perceptual goal and \mathbf{x} be a vector encoding

virtual finger commands. Let $\mathbf{p}(\mathbf{x})$ be the perception vector obtained from reading the sensors after executing the robot command corresponding to \mathbf{x} . A metric f , which monotonically decreases with the quality of the manipulation encoded by \mathbf{x} , is given by

$$f(\mathbf{g}, \mathbf{p}(\mathbf{x})) = \sum_{i=1}^m w_i (g_i - p_i(\mathbf{x}))^2$$

where m is the dimensionality of the sensory space, w_1, \dots, w_m are the relative weights of the errors in the different elements of the perception vector and $\forall i \in \{1, \dots, m\} w_i \geq 0$.

Given the uncertainty present in sensors and effectors, and the fact that f may have several local minima, it is unlikely that a standard Newton or gradient-based minimization method would suffice for this problem. Therefore we have to resort to optimization techniques that are better at dealing with local minima and handling an apparently non-deterministic environment.

The optimization method we use is a modification of the well-known evolution strategy [15, 16], augmented with an extrapolation operation in addition to the standard mutation operator.

2.1 The evolution strategy

The evolution strategy is family of iterative probabilistic optimization algorithms loosely based on biological evolution. In its simplest form, the optimization starts with a *parent*, a real-valued vector which encodes a candidate solution to the problem at hand. The following two steps are then repeated until a termination condition is attained: 1) Create a descendant, by randomly changing the parent (mutation). 2) Select the better of parent and descendant as the parent for the next iteration (selection). The process terminates when a prespecified number of iterations is executed or a goal value in the objective function is attained. According to the biological observation that offspring are similar to their parents and that small changes occur more often than large ones, mutation is realized by adding to the parent a normally distributed random vector with expected value zero. A major feature of this family of algorithms is the dynamic updating of the standard deviation of the distribution used to obtain the descendant in response to the characteristics of the region of the objective function that is being explored. If successful mutations occur rarely, the search is likely to be near a minimum and thus it is a good idea to decrease the size of the neighborhood being searched, since the minimum must be nearby. If successful mutations occur very often, it means that convergence could be sped-up by increasing the step size.

The evolution strategy has been shown to be globally convergent given unbounded running time [4]. Similar results have also been shown for simulated annealing [2] and genetic algorithms [5]. Of more practical interest, the evolution strategy has been shown in many applications to converge quickly

and be relatively insensitive to local minima. The evolution strategy is generally preferable to genetic algorithms for solving problems that deal with the optimization of functions of real numbers¹. Its main advantage over simulated annealing lies in its adaptive step-length, realized by dynamically varying the standard deviation of the mutation vector in response to the characteristics of the objective function and the region being explored.

2.2 The learning algorithm

The algorithm we use for learning manipulation primitives uses an extrapolation operator as a heuristic to guide the search in the direction of decreasing value of the objective function in addition to the standard mutation operator. The idea behind the extrapolation operator is to use the values of the objective function in the previous iterations to estimate the direction of the gradient of the function and obtain a new descendant by extrapolating in that direction. The extrapolation step length is dynamically adapted in response to the local characteristics of the function being explored. As in the mutation case, we increase the step length when the probability of a successful extrapolation is above a threshold and decrease it otherwise.

Formally the algorithm we use for learning the virtual fingers commands that will execute the desired manipulation can be described as follows.

Let f be the objective function as defined earlier, let $M \subset \mathcal{R}^{k \times l}$ be the set of valid virtual finger commands, where k is the number of virtual fingers and l is the number of degrees of freedom of each virtual finger. Let $\mathbf{p}(\mathbf{x}) \in \mathcal{R}^m$, $\mathbf{x} \in M$ be the perception obtained after executing the virtual finger command \mathbf{x} , where m is the dimensionality of the sensor space.

Given a perceptual goal \mathbf{g} , the overall goal of the optimization procedure is to find a vector $\mathbf{x}^* \in M$ such that

$$(\forall \mathbf{x} \in M) f(\mathbf{g}, \mathbf{p}(\mathbf{x}^*)) \leq f(\mathbf{g}, \mathbf{p}(\mathbf{x}))$$

The algorithm starts with a *parent* $u^0 = \langle \mathbf{x}^0, \sigma^0, \lambda^0, \mathbf{z}^0 \rangle$, where $\mathbf{x}^0 \in M$ is a candidate virtual finger command, $\sigma^0 > 0$ is the standard deviation, $\lambda^0 > 0$ is the extrapolation step length and $\mathbf{z}^0 \in \mathcal{R}^{k \times l}$ is the estimated gradient vector. In the absence of prior information, these values can be initialized randomly.

In each iteration, a candidate virtual finger command \mathbf{x}_m^i is obtained by mutating the parent

$$\mathbf{x}_m^i = \mathbf{x}^i + \mathbf{r}(0, \sigma^i)$$

where $\mathbf{r}(0, \sigma^i)$ denotes a random vector with each element obtained from a normal distribution with zero mean and σ^i standard deviation.

Another candidate virtual finger command \mathbf{x}_e^i is then obtained by adding to the parent a vector with magnitude λ^i in the estimated direction of the gradient.

$$\mathbf{x}_e^i = \mathbf{x}^i + \lambda^i \frac{\mathbf{z}^i}{|\mathbf{z}^i|}$$

After executing the commands encoded by \mathbf{x}^i , \mathbf{x}_m^i and \mathbf{x}_e^i , we obtain the perceptions $\mathbf{p}(\mathbf{x}^i)$, $\mathbf{p}(\mathbf{x}_m^i)$ and $\mathbf{p}(\mathbf{x}_e^i)$ and the objective function values $f(\mathbf{g}, \mathbf{p}(\mathbf{x}^i))$, $f(\mathbf{g}, \mathbf{p}(\mathbf{x}_m^i))$ and $f(\mathbf{g}, \mathbf{p}(\mathbf{x}_e^i))$.

The individual $u^{i+1} = \langle \mathbf{x}^{i+1}, \sigma^{i+1}, \lambda^{i+1}, \mathbf{z}^{i+1} \rangle$ to be used as the parent in the next generation is given by:

$$\mathbf{x}^{i+1} = \operatorname{argmin} f(\mathbf{g}, \mathbf{p}(\mathbf{x})), \mathbf{x} \in \{ \mathbf{x}^i, \mathbf{x}_m^i, \mathbf{x}_e^i \}$$

$$\sigma^{i+1} = \begin{cases} \sigma^i * c_d & \text{if } p_m^i > \frac{1}{5} \\ \sigma^i / c_d & \text{otherwise} \end{cases}$$

$$\lambda^{i+1} = \begin{cases} \lambda^i * c_d & \text{if } p_e^i > \frac{1}{5} \\ \lambda^i / c_d & \text{otherwise} \end{cases}$$

$$\mathbf{z}^{i+1} = \begin{cases} \alpha * \frac{\mathbf{z}^i}{|\mathbf{z}^i|} + (1 - \alpha) \frac{\mathbf{x}^{i+1} - \mathbf{x}^i}{|\mathbf{x}^{i+1} - \mathbf{x}^i|} & \text{if } \mathbf{x}^{i+1} \neq \mathbf{x}^i \\ \mathbf{z}^i & \text{otherwise} \end{cases}$$

where $1 > \alpha > 0$, p_m^i is the estimated probability of having a successful mutation, that is, the ratio of times the fitness of the individual obtained by mutation is better to that of the parent to the number of iterations², p_e^i is the estimated probability of having a successful extrapolation, and $c_d > 1$ is a constant.

The choice of $\frac{1}{5}$ as a constant to modify σ and λ is based on Rechenberg's *1/5 success rule* [15] and was proven to be optimal for a restricted kind of object functions and has been observed to work well in practice.

The goal of the learning algorithm is to fill-up a table containing virtual finger commands and indexed by object and perceptual goal, as shown in table 1. In the table, \mathbf{j}_i is a vector encoding the joint angles of the hand after grasping the *i*th object in the prototype set. For each object the system stores the configuration of the hand \mathbf{j}_i after the grasping operation and then it learns the virtual finger commands $\mathbf{x}_{i,1}, \dots, \mathbf{x}_{i,m}$ required to attain perceptual goals $\mathbf{g}_1, \dots, \mathbf{g}_m$ for that object using the algorithm described earlier. In the look-up phase we simply compare the joint angles of the hand after grasping the object with each of the \mathbf{j}_i s and retrieve the set of primitives corresponding to the joint angles that are the most similar to those obtained after grasping the current object. These primitives can be scaled and combined by adding them for performing tasks, as done in [17]. Although this combination of primitives is not mathematically correct, experimental results using the Utah/MIT have shown that it works well in practice.

¹Some modifications to the original binary-valued representation have been proposed and used somewhat successfully [8]

²In the implementation we use a fixed length window of past results to estimate this probability

Joint Angles	Perceptual Goals			
	g_1	g_2	\dots	g_m
j_1	$x_{1,1}$	$x_{1,2}$	\dots	$x_{1,m}$
j_2	$x_{2,1}$	$x_{2,2}$	\dots	$x_{2,m}$
\vdots	\vdots	\vdots	\vdots	\vdots
j_n	$x_{n,1}$	$x_{n,2}$	\dots	$x_{n,m}$

Table 1: Table to be filled-up by the learning algorithm

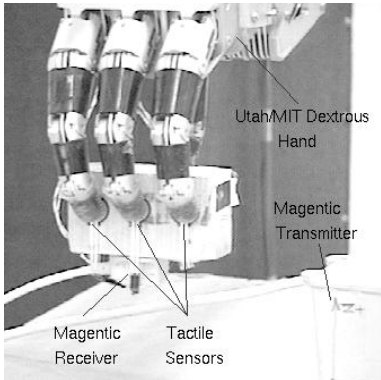


Figure 1: The experimental setup

3 Experimental Results

The algorithms described in the previous section were implemented in our vision and robotics lab using the Utah/MIT dextrous hand [10]. We used an Ascension flock of birds tm magnetic sensor attached to the object being manipulated for position and orientation sensing. For tactile sensing we used Interlink tm pressure-sensitive resistors, which were taped to the object. The experimental setup is shown in figure 1.

In the four-fingered Utah/MIT hand the thumb is permanently opposed to the index, middle and ring fingers, therefore it is natural to partition the real fingers into two virtual fingers, one composed by the thumb and the other by the remaining fingers. Each finger of the Utah/MIT hand is itself redundant, having four joints, three of which are coplanar. To solve this redundancy we use another observation made on human hands, namely, that the angles of the last two joints of each finger are roughly equal ³. This form of redundancy resolution and the use of virtual fingers, reduce the dimensionality of the parameter space from 16 to 6 and make autonomous learning in a reasonably short period of time feasible.

The system learned several basic manipulations consisting of translations along the three main axes and rotations about 2 of the main axes and several combinations of them. On average, each primitive was learned in about 50 generations using the algorithm described in Section 2. The algorithm was

³This observation has been used in other systems (e. g. [13]) for computing the inverse kinematics of the Utah/MIT hand.



Figure 2: Translation along the x axis

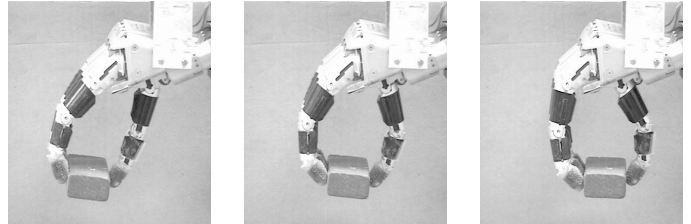


Figure 3: Translation along the y axis

run until the precision exceeded a predefined threshold. Each trial move took about one second. With three trials moves per generation and including other delays, each primitive was learned in a little over three minutes on average. The main bottleneck we faced was the hysteresis in the interlink tactile sensors, which forced us to wait for a few tenths of a second between moves to allow the sensors to return to their normal state. This alone accounted for about 40% of the running time.

Figure 2 shows the hand performing a translation along the x axis by sequentially moving to the previously learned goals $[x, y, z] = [-25, 0, 0]$, $[x, y, z] = [0, 0, 0]$ and $[x, y, z] = [25, 0, 0]$, where displacements are given in millimeters. Similarly, figure 3 shows a translation along the y axis using the sequence $[0, -25, 0]$, $[0, 0, 0]$, $[0, 25, 0]$. Figure 4 shows the sequence $[0, 0, 10]$, $[0, 0, 0]$, $[0, 0, -10]$ to perform a movement along the z axis. Intermediate positions (not shown) were obtained using the nearest neighbors approach, as explained in Section 2. These movements were learned with an object that is similar, but not identical, to the one used during execution (see figure 1.) It can be seen that the quality of the manipulation is quite good, showing also that the learned skills can be transferred between similar objects. Although few quantitative results for other systems have been reported, the quality of the manipulations seems comparable

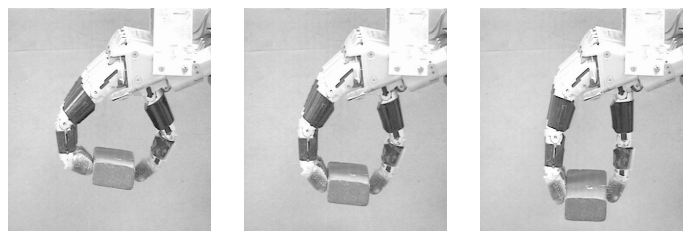


Figure 4: Translation along the z axis

to the one obtained by other systems where the manipulations are programmed by hand, such as [7, 11, 12, 17].

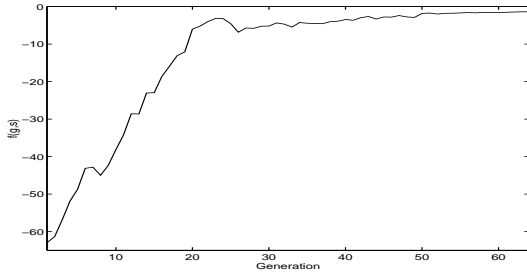


Figure 5: Manipulation quality as a function of generation for perceptual goal $[25, 0, 0, 0, 0, 0]$.

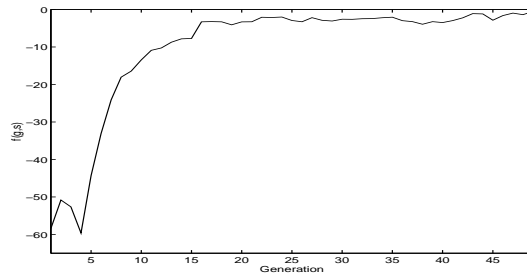


Figure 6: Manipulation quality as a function of generation for perceptual goal $[0, -25, 0, 0, 0, 0]$

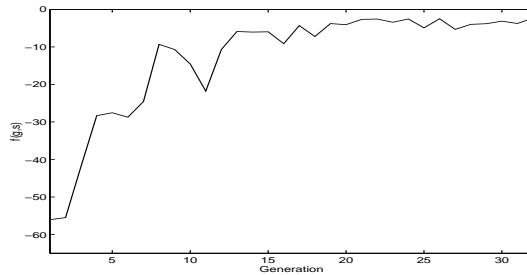


Figure 7: Manipulation quality as a function of generation for perceptual goal $[0, 0, -10, 0, 0, 0]$

Figure 5 shows the value of the manipulation quality function f for a translation of 25 mm along the x axis as a function of the generation. The perceptual goal also included maintaining a constant force applied by each finger, equal to the force measured at the start of the manipulation. After approximately 25 generations a good level of performance is attained. Near the goal, further exploration yields slower improvement, due in part to the fact that the noise makes the choices between two very similar parameter sets almost random. After 63 generations the prespecified accuracy was obtained and the program stopped. In this particular run the perception at that point was $\mathbf{p} = [24.940, 0.110, -1.208, 0.2406, -1.5355, -0.0688]$, yielding a final error of 1.21 mm in position and 1.56 degrees in orientation, which is remarkably accurate. Similarly,

Goal	Actual final position
$[25,0,0,0,0,0]$	$[24.9,0.11,-1.21,0.24,-1.54,-0.07]$
$[-25,0,0,0,0,0]$	$[-26.6,-3.41,1.87,-2.60,0.88,1.19]$
$[0,25,0,0,0,0]$	$[-1.76,24.5,0.99,-6.63,0.02,0.15]$
$[0,-25,0,0,0,0]$	$[-0.99,-26.3,0.11,0.24,-0.09,0.07]$
$[0,0,10,0,0,0]$	$[-1.54,-0.88,9.78,0.73,2.88,0.33]$
$[0,0,-10,0,0,0]$	$[0.11,-1.65,-9.01,-0.53,-7.45,0.33]$

Table 2: Goal positions and actual positions after learning for a few selected prototype movements. Positions are given in millimeters and orientation angles and errors in degrees.

figure 6 plots f for a translation of -25 mm along the y axis. In this case the convergence was a little quicker, exceeding the threshold after 48 generations. The final error was slightly larger in position and smaller in orientation, with a final perception of $\mathbf{p} = [-0.99, -26.28, 0.11, 0.243, -0.091, 0.066]$, an error of 1.6 mm in position and 0.26 degrees in orientation. Figure 7 shows the learning plot for $\mathbf{p} = [0, 0, -10, 0, 0, 0]$; the overall behavior of the optimization is similar to the previous two cases, although the final error is slightly higher. Table 2 shows goal positions, actual positions after learning and errors for a few selected manipulations. In general the results are consistent with the ones described above.

3.1 An insertion task

Figure 8 shows how a set of primitives can be used to perform a simple assembly task. The goal is to insert the hexagonal piece into the nearby hexagonal hole.

The system first learned 8 primitives, consisting of translations along the three main axes and rotation about the vertical axis in the positive and negative directions. The learned primitives could be invoked by a human teleoperator, using the keyboard in the controlling workstation, in order to solve the task.

4 Conclusions and Future Work

We have presented a method for machine learning of dextrous manipulation skills. We consider the following to be the most salient features of this work.

- Heuristics derived from observations made on human hands were used to reduce the degrees of freedom of dextrous manipulation with robotic hands. This significantly simplified the task and made autonomous learning feasible.
- Our system does not rely on simulation. Instead, all the experimentation is done by a physical robot. This is valuable in situations such as dextrous manipulation, where building a realistic and accurate simulator is extremely difficult.
- We used a modified version of the evolution strategy to learn manipulation primitives. This learning algorithm

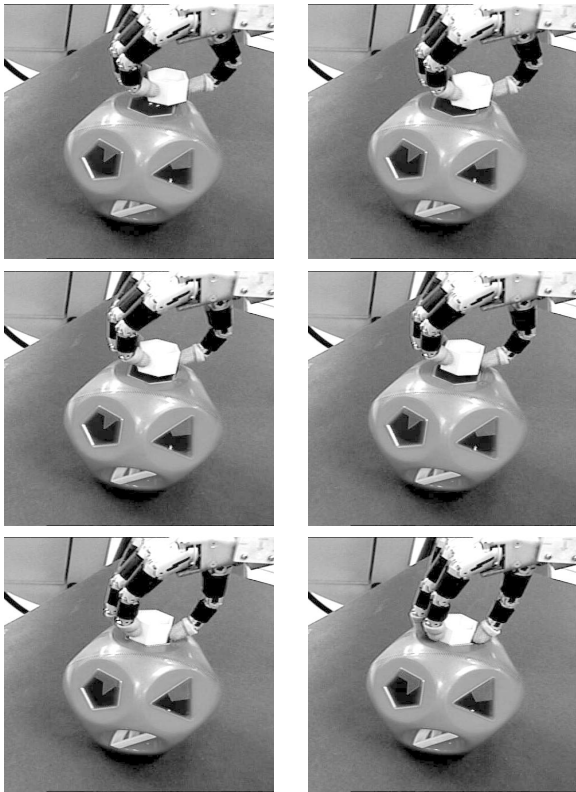


Figure 8: Performing a task using the learned primitives

successfully dealt with the noise in sensors and effectors and allowed the primitives to be learned in a period of a few minutes.

- We showed that the learned primitives can be combined to form general manipulations and perform more complex tasks.

Present and future work includes learning primitives that require repositioning the fingers on the surface of the object and using a more sophisticated version of the evolution strategy to learn the primitive skills in even shorter periods of time.

References

[1] M. Arbib, T. Iberall, and D. Lyons. Coordinated control programs for movements of the hand. Technical Report 83-25, Department of Computer and Information Science, University of Massachusetts at Amherst, Amherst, Massachusetts, 1983.

[2] E. H. L. Arts and J. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, Chichester, 1989.

[3] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[4] J. Born. *Evolutionsstrategien zur numerischen Lösung von Adaptationsaufgaben*. PhD thesis, Humboldt Universität, Berlin, Germany, 1978.

[5] A. E. Eiben, E. H. L. Aarts, and K. M. V. Hee. Global convergence of genetic algorithms: an infinite Markov chain analysis. In *Proceedings of the First International Conference on Parallel Problem Solving from Nature*, pages 4–12, Berlin, Germany, 1991. Springer.

[6] O. Fuentes and R. C. Nelson. Morphing hands and virtual tools (or what good is an extra degree of freedom?). Technical Report 551, Computer Science Department, University of Rochester, Rochester, New York, December 1994.

[7] O. Fuentes and R. C. Nelson. Experiments on dextrous manipulation without prior object models. Technical Report 606, Computer Science Department, University of Rochester, Rochester, New York, February 1996.

[8] T. Grossman and Y. Davidor. An investigation of a genetic algorithm in continuous parameter space. Technical Report CS92-20, The Weizmann Institute of Science, Department of Applied Mathematics and Computer Science, Rehovot, Israel, October 1992.

[9] T. Iberall. The nature of human prehension: Three dextrous hands in one. In *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, pages 396–401, Raleigh, North Carolina, 1987.

[10] S. Jacobsen, E. Iversen, D. Knutti, R. Johnson, and K. Bigger. Design of the Utah/MIT Dextrous Hand. In *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, pages 96–102, 1986.

[11] M. Jägersand, O. Fuentes, and R. C. Nelson. Acquiring visual-motor models for precision manipulation with robot hands. In *Proceedings of the Fourth European Conference on Computer Vision*, Cambridge, U. K., 1996.

[12] P. Michelman and P. Allen. Compliant manipulation with a dextrous robot hand. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 711–716, Atlanta, Georgia, 1993.

[13] S. Narasimhan. Dexterous robot hands: Kinematics and control. Master's thesis, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, November 1988.

[14] R. C. Nelson, M. Jägersand, and O. Fuentes. Virtual tools: A framework for simplifying sensory-motor control in complex robotic systems. Technical Report 576, Computer Science Department, University of Rochester, Rochester, New York, March 1995.

[15] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, 1973.

[16] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Ltd., 1981.

[17] T. H. Speeter. Primitive Based Control of the Utah/MIT Dextrous Hand. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 866–877, Sacramento, California, 1991.