Learning Dextrous Manipulation Skills Using Multisensory Information *

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

Abstract

In this paper we present a method for autonomous learning of dextrous manipulation skills with multifingered robot hands. We use heuristics derived from observations made on human hands to reduce the degrees of freedom of the task and make learning tractable. Our approach consists of learning and storing a few basic manipulation primitives for a few prototypical objects and then using an associative memory to obtain the required parameters for new objects and/or manipulations. During learning, sensory information from tactile sensors and a position measuring device is used to evaluate the quality of a candidate manipulation. The parameter space of the robot is searched using a modified version of the evolution strategy, which is robust to the noise normally present in real-world complex robotic tasks. Given the difficulty of modeling and simulating accurately the interactions of multiple fingers and an object, and to ensure that the learned skills are applicable in the real world, our system does not rely on simulation; all the experimentation is performed by a physical robot, in this case the 16-degree-of-freedom Utah/MIT hand. Experimental results show that accurate dextrous manipulation skills can be learned by the robot in a short period of time.

1 Introduction

Dextrous manipulators have potential applications in areas such as prosthetics and space and deep sea exploration where a single robot must perform a wide variety of tasks and thus versatility rather than precision is the main requirement. Programming these robots to operate intelligently in the world is a difficult and time-consuming task, therefore, 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.

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

Due to the large number of degrees of freedom of dextrous manipulators, machine learning approaches to this problem face the well-known "curse of dimensionality" [2], which 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 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, such as in dextrous manipulation, it is difficult or impossible to gather enough knowledge to build a realistic simulation. For this reason, 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. Using the virtual finger abstraction, the dimensionality of 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

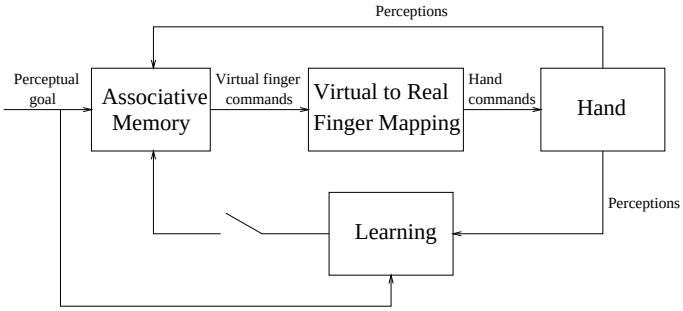

Figure 1: Schematic diagram of the manipulation system

from virtual to real fingers can be learned as a separate problem or be provided by the human programmer.

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 using an evolutionary algorithm 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 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.

2 Learning Dextrous Manipulation Skills

Figure 1 shows the general structure of our manipulation system, including its learning component. The system first learns a set of primitives for performing basic translations and rotations of the object [1]. New manipulations are obtained by combining the primitives using a nearest-neighbor method. The goal of the learning system is to build a table, indexed by goals and objects that gives as output the virtual finger commands that can later be converted to the robot commands that will achieve the goal. It is assumed that the mapping from virtual to real fingers, which is task and manipulator dependent, is provided by the human programmer.

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.

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

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

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

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

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 [14], an iterative probabilistic optimization algorithm 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, then the following two steps are 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 that 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 oc-

[1] These primitives are analogous to Speeter's *motion primitives* [15], but in his system the primitives were supplied by the programmer, while in ours they are learned automatically.

cur rarely, the search is likely to be near a minimum and the size of the neighborhood being searched should be decreased. If successful mutations occur too often, it means that convergence could be sped-up by increasing the step size.

2.1 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 used in the evolution strategy. 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. Similarly to 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 $\boldsymbol{p}(\boldsymbol{x}) \in \mathcal{R}^m, \boldsymbol{x} \in M$ be the perception obtained after executing the virtual finger command $\boldsymbol{x}$, where m is the dimensionality of the sensor space. The overall goal of the algorithm is to find a vector $\boldsymbol{x}^* \in M$ such that

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

The optimization procedure starts with a *parent* $u^0 = \langle \boldsymbol{x}^0, \sigma^0, \lambda^0, \boldsymbol{z}^0 \rangle$, where $\boldsymbol{x}^0 \in M$ is a candidate virtual finger command, $\sigma^0 > 0$ is the standard deviation, $\lambda^0 > 0$ is the extrapolation step length and $\boldsymbol{z}^0 \in \mathcal{R}^{k \times l}$ 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 $\boldsymbol{x}_m^i$ is obtained by mutating the parent

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

where $\boldsymbol{r}(0, \sigma^i)$ 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 $\boldsymbol{x}_e^i$ is then obtained by adding to the parent a vector with magnitude λ^i in the estimated direction of the gradient.

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

	Perceptual Goals			
Joint Angles	$\boldsymbol{g}_1$	$\boldsymbol{g}_2$	$\cdots$	$\boldsymbol{g}_m$
$\boldsymbol{j}_1$	$\boldsymbol{x}_{1,1}$	$\boldsymbol{x}_{1,2}$	$\cdots$	$\boldsymbol{x}_{1,m}$
$\boldsymbol{j}_2$	$\boldsymbol{x}_{2,1}$	$\boldsymbol{x}_{2,2}$	$\cdots$	$\boldsymbol{x}_{2,m}$
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
$\boldsymbol{j}_n$	$\boldsymbol{x}_{n,1}$	$\boldsymbol{x}_{n,2}$	$\cdots$	$\boldsymbol{x}_{n,m}$

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

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

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

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

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

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

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

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 iterations[2], 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* [12] and was proven to be optimal for a restricted kind of object functions and has been observed to work well in practice.

The learning algorithm is used to fill-up a table containing virtual finger commands and indexed by object and perceptual goal as shown in table 1. In the table, $\boldsymbol{j}_i$ is a vector encoding the joint angles of the hand after grasping the *ith* object in the prototype set. For each object the system stores the configuration of the hand $\boldsymbol{j}_i$ after the grasping operation and then it learns the virtual finger commands $\boldsymbol{x}_{i,1}, \ldots, \boldsymbol{x}_{i,m}$ required to attain perceptual goals $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_m$ for that object using the algorithm described earlier.

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

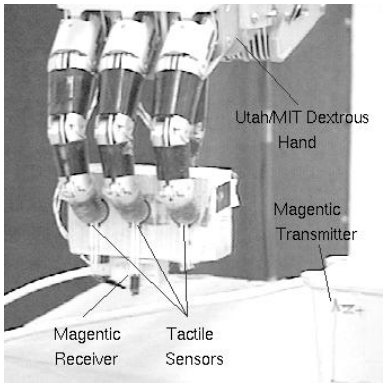

Figure 2: The experimental setup

In the look-up phase we use a nearest-neighbors approach to obtain the appropriate virtual finger commands for a given object and goal. Let $\boldsymbol{j}_{meas}$ be the configuration of the hand, obtained from the hand's joint angle sensors and let $\boldsymbol{g}^*$ be the perceptual goal. Let $\boldsymbol{j}_p$ and $\boldsymbol{j}_q$, $\boldsymbol{p} \neq \boldsymbol{q}$ be the two closest values to $\boldsymbol{j}_{meas}$ in the row index column. Let $\boldsymbol{g}_r$ and $\boldsymbol{g}_s$, $r \neq s$ be the two closest values to $\boldsymbol{g}^*$ in the column index row. The command $\boldsymbol{x}^*$ to attain the goal for the given object is given by:

$$\boldsymbol{x}^* = \sum_{i \in \{p,q\}} \sum_{j \in \{r,s\}} \left(\frac{|\boldsymbol{j}_i - \boldsymbol{j}_{meas}|}{|\boldsymbol{j}_p - \boldsymbol{j}_{meas}| + |\boldsymbol{j}_q - \boldsymbol{j}_{meas}|} \cdot \frac{|\boldsymbol{g}_j - \boldsymbol{g}^*|}{|\boldsymbol{g}_r - \boldsymbol{g}^*| + |\boldsymbol{g}_s - \boldsymbol{g}^*|} \boldsymbol{x}_{\{p,q\}-i,\{r,s\}-j} \right)$$

3 Experimental Results

The algorithms described in the previous section were implemented in our vision and robotics lab using the Utah/MIT dextrous hand [6]. 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 2.

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.

[3] This observation has been used in other systems (*e. g.* [10]) for computing the inverse kinematics of the Utah/MIT hand.

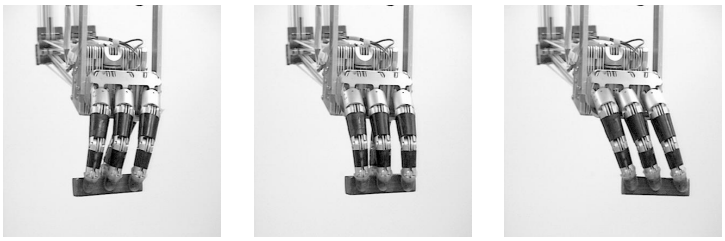

Figure 3: Translation along the x axis

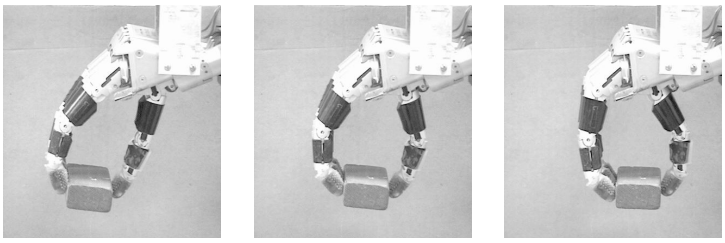

Figure 4: Translation along the y axis

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 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. They main bottleneck we faced was the hysteresis in the interlink tactile sensors, which forced us to wait for a few tens 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.

Figures 3 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 4 shows a translation along

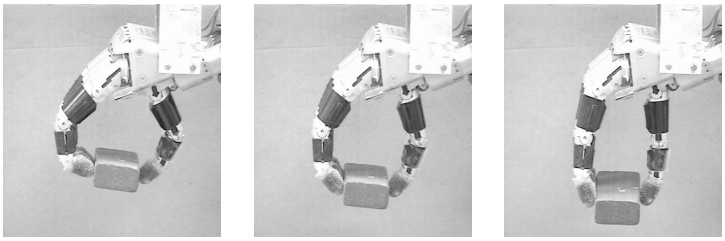

Figure 5: Translation along the z axis

the y axis using the sequence $[0, -25, 0], [0, 0, 0], [0, 25, 0]$. Figure 5 shows the sequence $[0, 0, 10], [0, 0, 0], [0, 0, -10]$ to perform a movement along the z axis. Intermediate positions (not shown) where 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 2.) 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 to the one obtained by other systems where the manipulations are programmed by hand, such as [15, 9, 7, 5].

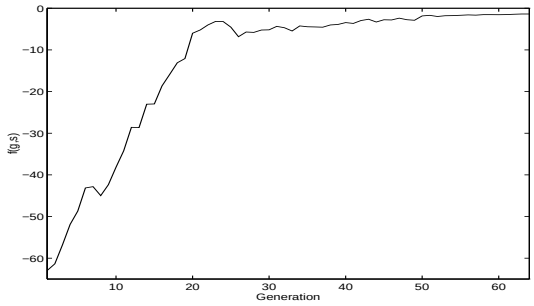

Figure 6: Evaluation of the quality of manipulation as a function of generation for perceptual goal $[25, 0, 0, 0, 0, 0]$, corresponding to a 25 mm translation along the x axis.

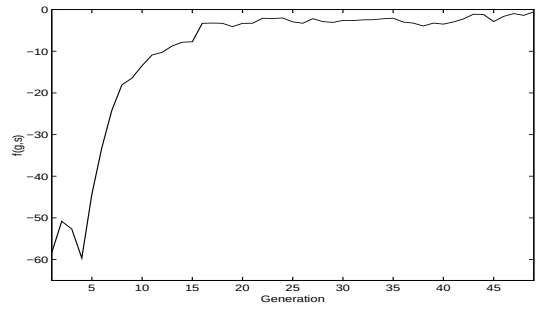

Figure 7: Evaluation of the quality of manipulation as a function of generation for perceptual goal $[0, -25, 0, 0, 0, 0]$.

Figure 6 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 forces 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 $\boldsymbol{p} = [24.940, 0.110, -1.208, 0.2406, -1.5355, -0.0688]$, yielding a final error of 1.21 mm in position and 1.56 degrees in orientation, which is remarkably accurate. Similarly, figure 7 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 a final perception of $\boldsymbol{p} = [-0.99, -26.28, 0.11, 0.243, -0.091, 0.066]$, an error of 1.6 mm in position and 0.26 degrees in orientation. Figure 8 shows the learning plot for $\boldsymbol{p} = [0, 0, -10, 0, 0, 0]$; the overall behavior of the optimization is similar to the previous two cases. 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.

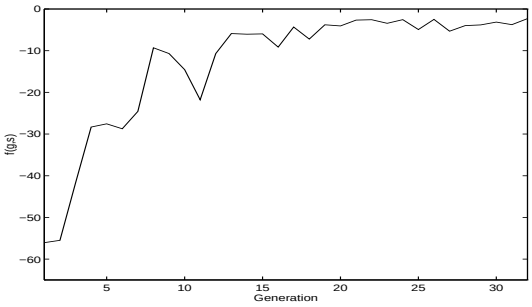

Figure 8: Evaluation of the quality of manipulation as a function of generation for perceptual goal $[0, 0, -10, 0, 0, 0]$.

Goal	Position Error	Orientation Error
[25, 0, 0, 0, 0, 0]	1.21	1.56
[-25, 0, 0, 0, 0, 0]	4.20	3.44
[0, 25, 0, 0, 0, 0]	2.08	6.64
[0, -25, 0, 0, 0, 0]	1.60	0.26
[0, 0, 10, 0, 0, 0]	1.79	2.99
[0, 0, -10, 0, 0, 0]	1.93	7.51

Table 2: Goal positions and errors for a few selected learned manipulations.

Figure 9 illustrates how the system can generalize to new objects. The figure shows the hand manipulating an object that was not in the original prototype set. The appropriate virtual finger commands were computed using the nearest neighbors method, as explained in section 2. It can be seen that the quality of the manipulation is good, even though the system did not receive any training for that particular type of object.

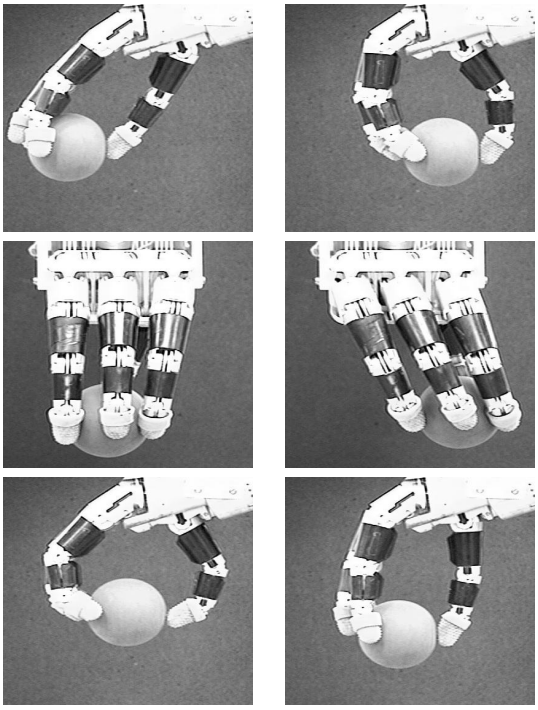

Figure 9: Generalization to a new object

4 Related Work

Some work has been done in robotic manipulation using machine learning techniques. In general, these approaches have dealt with simple tasks such as grasping with parallel jaw grippers, and simple manipulation strategies using robot arms.

Dunn and Segen [4] presented a robotic system that learns how to grasp objects. In their system, when an object is presented for the first time the robot experiments with it, seeking a way to grasp it by trial and error using visual information and input from the robot gripper. A discovered grasp is saved along with the object's shape. The system generalizes to different positions and orientations but not to sizes.

Kamon *et al.* [8] presented a robotic system that learned to grasp objects with a parallel-jaw gripper using visual information. Their system learns two separate subproblems: to choose grasping points, and to predict the quality of a given grasp. It incrementally improves its performance over the course of a training session. The system used very little information about the target object; in particular, no attempt was made to recover the object's shape.

Salganicoff *et al.* [13] used a modified version of the ID-3 inductive learning algorithm [11] in a robotic system that learned to grasp objects using visual information. Their system learned likely to succeed grasping strategies in the form of the azimuth and elevation approach angles of the gripper to the object given a superellipsoid fit of the object as input.

Christiansen, Mason and Mitchell [3] described a system that learned models of manipulation actions from observations of the effects of such actions. In their experimental implementation, a robot learned how to re-position and re-orient an object located on a tray, held by the robot from underneath, by a sequence of tray tilts. This work introduced the term *apparent non-determinism* to refer to the fact that executing the same action twice from the same starting state might give different results. They coped with apparent non-determinism by assuming probabilistic rather than deterministic transitions between states. This system uses a discretization of the perception and action spaces, which are both one-dimensional and scalability to more complex tasks may be a difficult.

The approaches to manipulation using machine learning described in this section deal with low-dimensional parameter spaces, and thus might not be suitable for use with a redundant, high-degree-of-freedom manipulator. Another potential inconvenience is the discretization of the action and state spaces.

5 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 form 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 successfully dealt with the noise in sensors and effectors and allowed the primitives to be learned in a period of a few minutes.
- We used an associative memory to select the appropriate set of primitives to use from the ones stored during the learning phase. The learned primitives could be combined to form general manipulations.

Present and future work includes application of the system to solve manipulation tasks such as assembly, 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 basic primitives 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] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[3] A. D. Christiansen, M. T. Mason, and T. M. Mitchell. Learning Reliable Manipulation Strategies without Initial Physical Models. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 1224–1230, Cincinnati, Ohio, 1990.

[4] G. B. Dunn and J. Segen. Automatic discovery of robotic grasping configurations. In *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, pages 396–401, 1988.

[5] 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.

[6] 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.

[7] 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.

[8] I. Kamon, T. Flash, and S. Edelman. Learning to grasp using visual information. Technical report, The Weizmann Institute of Science, Revhovot, Israel, March 1994.

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

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

[11] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

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

[13] M. Salganicoff, L. G. Kunin, and L. H. Ungar. Active exploration based ID-3 learning for robot grasping. In *1994 Workshop on Robot Learning*, New Brunswick, NJ, 1994.

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

[15] 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.