Task-Level Robot Learning: Juggling a Tennis Ball More Accurately

Eric W. Aboaf, Steven M. Drucker, and Christopher G. Atkeson

The Artificial Intelligence Laboratory and Brain and Cognitive Sciences Department,
Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139

Abstract

We report on a preliminary investigation of task-level learning, an approach to learning from practice. We have programmed a robot to juggle a single ball in three dimensions by batting it upwards with a large paddle. The robot uses a real-time binary vision system to track the ball and measure its performance. Task-level learning consists of building a model of performance errors at the task level during practice, and using that model to refine task-level commands. A polynomial surface was fit to the errors in where the ball went after each hit, and this task model is used to refine how the ball is hit. This application of task-level learning dramatically increased the number of consecutive hits the robot could execute before the ball was hit out of range of the paddle.

1 Introduction

We are investigating how to program robots so that they learn from experience. Our premise is that robots can practice a task and use that experience to improve their performance. One way to improve robot performance with experience is with task-level learning algorithms. We initially explored task-level learning on a ball-throwing task [Aboaf et al. 1988] and now suggest that the performance of a more complex task, juggling a single ball in three dimensions, can similarly be improved. See [Aboaf 1988] for a more extensive discussion of the research described in this paper.

We base our task-level learning algorithms on a rather commonplace observation. A person throwing a ball at a target for the first time will often miss. If the ball does not reach the target, the person will aim a little farther and throw again. If the ball is thrown too far, the person will aim a little closer. From this simple example, we notice that people tend to vary the *aim*, a command specific to the throwing task, to compensate for the error in performance. An important part of this type of learning may be building an internal model of the task being performed. In the throwing task, the task model predicts the aim necessary to hit a particular target. We also observe that people are quite willing to practice a task until they finally succeed. Given the competence demonstrated by humans performing complex dynamic tasks, it seems reasonable to try to formalize this approach and assess its strengths and limitations.

We are developing task-level learning procedures that attempt to mimic the approach we think people are using. The procedures require that a robot system practice a task, monitor its own performance, and adjust its commands until the task is performed correctly. The learning procedures are called *task-level* because they directly refine task-level commands. In ball-throwing, the learning procedures improve the performance on the task by adjusting where the robot system aims when it throws the ball. The robot control system and the models of the robot used in planning and control need not be refined during practice. In juggling, we found we needed to develop a model of the juggling task. This model is used to adjust where the robot juggling system aims on each hit. By developing task models and adjusting the task-level command, the learning procedures compensate for inaccuracies in the models used in the robot control system and improve the robot's performance of a task.

Relation to Other Work

Task-level learning is complementary to robot calibration. In robot calibration, the component models such as the kinematic, dynamic, actuation, and sensing models that describe a robot system are made more accurate. With these more accurate models, the robot system can be controlled more precisely. The model calibration literature is extensive. Many researchers have calibrated the dynamics parameters of a robot manipulator based on the Newton-Euler model of rigid body motion (see [An, Atkeson and Hollerbach 1988] for references). Other researchers have accurately modeled the kinematics of robot manipulators based on the Denavit-Hartenberg model (see [Hollerbach 1988] for a survey). Other researchers have focused on the task of calibrating the sensing models of a robot system. For a robot equipped with a camera system, the research is referred to as hand-eye coordination [Inoue and Inaba 1984, Liang, Lee, and Hackwood 1988, Shiu and Ahmad 1987, Tsai and Lenz 1987].

In task-level learning, we assume that the models describing the robot system are not calibrated "well enough." In other words, when commanded to perform a specified task, the robot does not perform as expected. Task-level learning is a method of compensating for the modeling errors that cause performance errors. Task-level learning does not produce highly calibrated models of the robot behavior under many different conditions. Instead, it produces a command which will achieve a particular task or a task model which will produce appropriate commands for only one task. The major advantage of these learning algorithms is that less data is necessary to refine the task-level command or task model than to perform extensive general model calibration. Learning at the task level reduces the degrees of freedom of the models to be learned. Instead of adjusting all the parameters of the kinematics, dynamics, actuation, and sensing models of a robot, only the task-level commands of the system or the parameters that represent a particular task are modified. Robot calibration involves making a large number of motions to calibrate the system, Task-level learning uses motions that actually attempt the task to improve the performance. Task-level learning may provide a more concise method of achieving the task goal. The disadvantage of the task-level approach is that it only improves performance on a single task. Information on modeling errors gathered during the performance of the desired task is put in a form that makes it difficult to use for other tasks not similar to the desired task.

Ultimately, task-level learning and model calibration can be used simultaneously to improve performance. Task-level learning can also be combined with other proposed forms of robot learning (see [Atkeson et al. 1987] and [Aboaf 1988] for further references).

2 The Juggling Task

We have chosen a juggling task on which to explore task-level learning. Juggling is a dynamic multi-dimensional task. It is also complex, involving many elements of a robot system, such as real-time vision, kinematics, and dynamics.

The task is to bounce a platform tennis ball on a paddle attached to the end of the robot's arm. The robot repeatedly hits the ball with the paddle, bouncing it up into the air much as a person can do with a tennis racquet and ball. For each hit, the task goal is three dimensional: (1) to hit the ball to a specified height z, (2) for the ball to land at a position x on the paddle, and (3) for the ball to land at a position y on the paddle. This juggling task is similar to the two-dimensional task of juggling on an inclined plane that has been described by Bühler and Koditschek [1987].

Juggling begins when a ball is dropped from the ceiling and falls towards the robot paddle. The task involves monitoring the flight of the ball, and estimating its downward trajectory. Based on the estimated landing position of the ball, a desired upward ball trajectory is calculated that returns the ball to the center of the paddle after the hit. Using these upward and downward ball trajectories, the robot system computes the necessary velocity with which to hit the ball. The robot must then calculate a paddle motion that will hit the ball with the correct velocity and the correct angle at the correct time. After the robot moves the paddle with this desired motion and hits the ball upwards, the sequence of monitoring the ball, calculating a response, and hitting the ball is repeated.

The juggling system consists of the MIT Serial Link Direct Drive Arm [Asada and Youcef-Toumi 1987; An, Atkeson, and Hollerbach 1988], two electronically shuttered cameras, a Datacube vision system, four 68020 microprocessors running under the Condor operating system [Narasimhan and Siegel 1987], and a platform tennis ball. In addition, a square paddle (0.46 m by 0.46 m) is attached to the Direct Drive Arm. The experimental setup is shown in Figure 1.

The task begins when a platform tennis ball is released by a solenoid from above the paddle. Two video cameras, each mounted 5.0 m from the robot and perpendicular to one another, monitor the position of the ball at 30 Hertz. Centroids of the ball (which appears white on a black background) are obtained by thresholding the digitized image and averaging the locations of the bright pixel values. In addition, at each sample interval a time is associated with centroid information. The ball centroid information is transformed into robot coordinates by means of a *vision model*. The transformation assumes a simple orthographic projection model of image formation. Since the cameras are mounted perpendicular to one another, the x, y, z, t position of a tennis ball in paddle coordinates is easily inferred from a full data point.

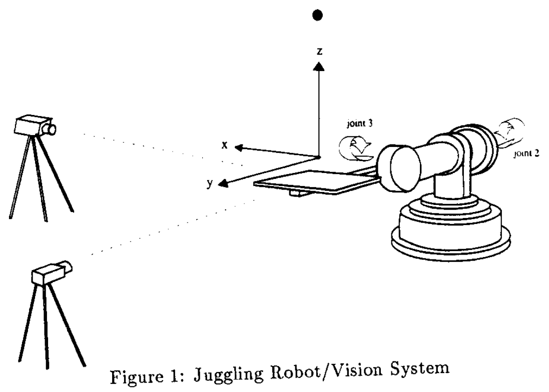

Figure 1: Juggling Robot/Vision System

Based on the centroids of the ball in flight, a *forward ballistics model* estimates the landing point of the ball on the paddle. The model performs a parabolic least-squares fit to obtain the landing time of the ball (the landing position is assumed to be when the ball crosses the plane $z = 0.0$, the height of the paddle). The model also performs linear least-squares fits to obtain the x and y position of the ball at impact. These fits presume a simple ballistic model for the flight of the ball:

$$x = x_i + \dot{x}_i t \quad (1)$$

$$y = y_i + \dot{y}_i t \quad (2)$$

$$z = z_i + \dot{z}_i t - 1/2 g t^2 \quad (3)$$

where the subscript i indicates the initial values of the positions and velocities, g is the gravitational constant, and t is time.

Based on the task goal and the estimated landing position of the ball, an *inverse ballistics model* computes the desired outgoing trajectory for the ball, using the above ballistic equations. The outgoing trajectory is fully determined by specifying the landing position of the ball and the task goal. The task goal is a three-dimensional vector that describes the height, z, that the ball should reach, and the x, y landing point on the next bounce.

Once estimates of the incoming and desired outgoing trajectories of the ball are made using the forward and inverse ballistics models, a *restitution model* predicts the angle and velocity with which to hit the ball. A simplified model [Beer and Johnston 1977] assumes that the angle of incidence and reflectance of the ball with respect to the paddle are equal. The angle at which to hit the ball is computed by averaging these two angles. The velocity with which to hit the ball can be computed using an estimated coefficient of restitution.

A *kinematics model* relates the desired angle and velocity of the paddle to the joint angles and velocities of the robot arm. Joints two and three of the Direct Drive Arm are used to hit the ball. The position of joint two is used to hit the ball in the x direction. The angular position of joint three is used to hit the ball in the y direction. The velocity of joint three is used to modulate the upward velocity of the ball.

A *trajectory model* computes the joint trajectories that bring the paddle to the desired angle and velocity for each hit. The model includes three phases—speed-up, contact, slow-down—for the trajectories of joints two and three. Both joints begin at rest in the speed-up phase. Midway through the contact phase, joint two reaches its prescribed hitting angle and joint three reaches its prescribed hitting angle and velocity. Both joints return to rest at the end of the slow-down phase. Planned joint motions follow fifth-order polynomial trajectories.

A *dynamics model* and feedback controller are used to drive the robot joints along the desired trajectory. A simplified dynamics model calculates the feedforward torques based on the acceleration profiles of the arm trajectories. A position–velocity feedback controller is used to improve trajectory following.

3 Task-Level Learning

When the robot uses the models described above to juggle a tennis ball, the system only averages about two hits before the ball is hit out of the robot's reach. The maximum number of successful hits (those that result in the ball landing somewhere on the paddle) is four. Even when four successful hits do occur, the ball rarely falls to the center of the paddle as desired.

Given a system with performance errors, we could try to further calibrate the component models described above, or use more accurate but also more complex models. We could also explore approaches based on some form of learning. We chose to explore task-level learning. Instead of calibrating models of the system, we modify the robot system's task-level command based on errors in task performance. The task-level command in this juggling task is the target location for the next bounce. We begin by describing task-level learning in the context of one-hit batting experiments and then demonstrate how the ideas can be extended to repeatedly bouncing the ball on the paddle.

3.1 The First Hit

The first hit involves hitting the ball upwards after it has been dropped from the solenoid towards the paddle. The aim for the first hit is three-fold: (1) to hit the ball upwards to a height of 1.0 m, (2) to hit the ball so that it falls to $x = 0.0$ m on the paddle, and (3) to hit the ball so that it falls to $y = 0.0$ m. This goal corresponds to hitting the center of a level paddle.

Our objective is to improve the robot's performance of the first hit. Without any learning, the ball lands forward and to the left of the center of the paddle after the first hit. The error is caused by the inaccuracy of the component models that describe the juggling system. After applying our task-level learning algorithms, the robot learns to hit the ball to the center of the paddle as desired.

The first hit can be performed by using task-level learning algorithms that have previously been developed for throwing a ball [Atkeson et al. 1987, Aboaf et al. 1988, Aboaf 1988]. In task-level learning, the system's task-level command is adjusted (using $\Delta\mathbf{aim}$) on each trial (indicated by the subscript n) based on errors in performance:

$$\mathbf{aim}_n = \mathbf{target} - \Delta\mathbf{aim}_n \tag{4}$$

$$\Delta\mathbf{aim}_0 = 0 \tag{5}$$

$$\Delta\mathbf{aim}_{n+1} = \Delta\mathbf{aim}_n + (\mathbf{result}_n - \mathbf{target}) \tag{6}$$

where **target** is the desired system performance or task goal and **result** is the actual system performance. At each learning iteration, the aim is adjusted based on the performance error on the previous try. This new aim is transformed through the seven inverse models to calculate the corrected low level signals that

Aim	Result	Error
$(0.00, 0.00, 1.00)$	$(-0.26, 0.46, 1.21)$	$(-0.26, 0.46, 0.21)$
$(0.26, -0.46, 0.79)$	$(0.15, -0.23, 1.12)$	$(0.15, -0.23, 0.12)$
$(0.11, -0.23, 0.67)$	$(-0.02, -0.07, 1.00)$	$(-0.02, -0.07, 0.00)$
$(0.13, -0.16, 0.67)$	$(0.00, 0.01, 0.99)$	$(0.00, 0.01, -0.01)$

Figure 2: Learning the First Hit

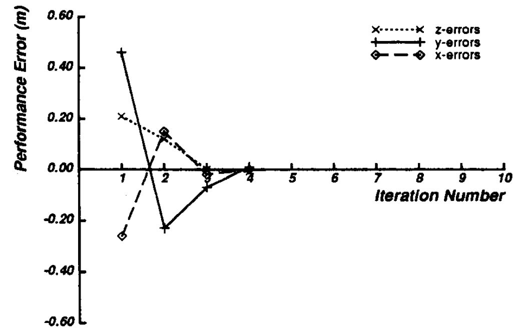

Figure 3: Learning the First Hit

drive the robot. A physical interpretation is helpful in understanding the above equations. In the case where the ball is hit forward, the performance error is positive, reducing the aim by that amount. This action corresponds to our intuition that we should aim slightly backwards if we hit too far forward.

Experiments

In our first experiments, we applied the task-level learning algorithms to the first juggling hit while suppressing the effect of random error. Noise is suppressed by performing the first hit five separate times and averaging the results. This averaging procedure allows us to focus on the learning process instead of the effects of noise. Later, experiments will be performed in which no averaging takes place and random error is not suppressed.

The first hit was successfully performed after four learning iterations. The target was to hit the ball to the point $(0.0, 0.0, 1.0)$. The learning sequence began with the juggling system aiming at $(0.0, 0.0, 1.0)$. Five trials were performed with this aim, and the resulting average hit was $(-0.26, 0.46, 1.21)$. The robot consistently hit the ball forward, to the left, and too high. Based on the average hit, a new aim of $(0.26, -0.46, 0.79)$ was calculated. Once again, five trials with this aim were performed, and the results averaged to $(0.15, -0.23, 1.12)$. A new aim was calculated, and five more hits were performed. This sequence of attempting the task, calculating the performance errors, and adjusting the aim was repeated four times. The sequence of aims, results, and errors are shown in Figure 2. Figure 3 shows a plot of the x, y, and z errors during learning. The task was considered complete once the error in each component was below the variability of a bounce. Variability levels were set at one standard deviation of the variability after the first bounce of a ball dropped from 1.5 m onto a paddle that was not moving ($\sigma_x = 0.07$ m, $\sigma_y = 0.07$ m, and $\sigma_z = 0.01$ m).

After learning, the aim that was used to perform the first juggling hit was $(0.13, -0.16, 0.67)$. The robot juggling system was aiming backward, to the right, and low in order to hit the ball to the desired target location. The difference between this aim vector and the target vector $(0.0, 0.0, 1.0)$ is an indication of the inaccuracy of the component models in the robot control system. Task-level learning compensated for this modeling inaccuracy using a small number of practice hits.

3.2 Learning the Second Hit

Hitting the ball the second time is much like hitting the ball the first time. As the ball is falling towards the paddle for the second time, the robot system computes an appropriate paddle motion that will hit the ball upwards. Once again, the goal is to hit the ball so that it will land at the center of the paddle, $x, y, z = (0.0, 0.0, 1.0)$.

To initially perform the second hit, the robot was programmed to use the aim of $(0.13, -0.16, 0.67)$, the aim that was learned for the first hit. The motivation is that the information learned on the first hit should generalize to the second hit. Unfortunately, the state of the system for these two hits is different, and the learned information does not generalize in a simple fashion. For the first hit, the ball is dropped from 1.5 m and reaches the paddle at approximately 1.5 m/s. For the second hit, the ball peaks at a height of 1.0 m, and reaches the paddle with a velocity of 1.0 m/s. The error in the juggling model is *different* in the two cases. As a result, learned compensations in the aim for the 1.5 m drop are different from learned compensations in the aim on the 1.0 m bounce.

Because the generalization was not good enough, we applied task-level learning to further adjust the robot's aim on the second hit. The initial aim was $(0.13, -0.16, 0.67)$. This aim was adjusted during task-level learning over nine learning iterations. No effort was made to "average out" the effects of noise on the learning process. As a result, seven iterations were necessary until the aim resulted in a hit that landed close enough to the target (one standard deviation of the bounce variability). To further improve the aim, we began performing three trials and averaging the results. Finally, on the ninth learning iteration, the error was again reduced below variability levels. The errors during learning are shown in Figure 4. It is difficult to assess the convergence rate or stability of the implemented algorithm due to the large amounts of noise. The final aim used was $(0.06, -0.12, 0.84)$.

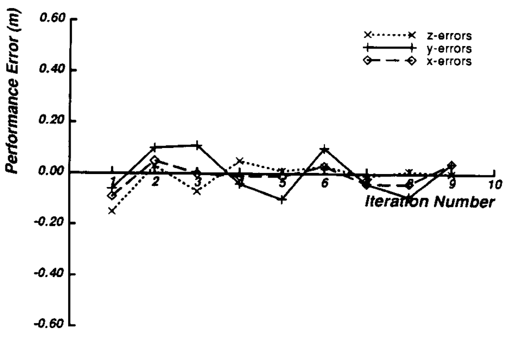

Figure 4: Learning the Second Hit

4 Juggling

After the robot improved its performance of the first two hits, we asked the question: how many hits can it do? The answer will tell us whether additional learning is necessary to make the robot juggle.

Initial Juggling Performance

We performed juggling experiments using the learned aim for the first hit and the learned aim for the second hit. The learned aim for the second hit was also generalized to all the successive hits. The robot juggling system performed 20 juggling runs in this configuration. The robot averaged 8 hits on each run, with a low of 3 hits and a high of 23 hits. The performance of the robot could be characterized as erratic. A visual analysis of the robot failures indicated that the ball oscillated back and forth in the y direction on successive bounces, eventually landing out of reach. The robot overcompensated along this direction. When the ball was hit forward of the center, the robot hit the ball too far back. If the ball was hit behind the center, the robot hit the ball too far forward. These oscillations eventually caused the ball to miss the paddle.

In order to improve on this performance, we applied additional forms of task-level learning. We analyzed the performance errors of the juggling robot. In order to interpret the errors that occurred during the 160 juggling hits from all the runs, we examined each performance error versus the x and y paddle location *from which* the ball was hit. A preliminary analysis indicated that these state variables accounted for much of the measured errors. We display the performance errors that occur before and after learning in Figure 5. The area of the robot paddle is divided into 16 square regions measuring 0.125 m on a side. The performance errors of the hits from within each region are averaged together and displayed. The shading of each region corresponds to the range in performance error, with white indicating a positive performance error and black indicating a negative one. The plots in the left column of Figure 5 describe the errors that occurred during the 160 hits before learning. Each plot provides an indication of the kind of errors that occur in the juggling system. The x plot suggests that the ball is hit too far to the left (a negative error), except when it is hit from the light-colored regions. The y plot suggests that the ball is consistently hit too far forward when it is hit from the near edge of the paddle. The z plot indicates that the ball is consistently hit too high when hit from the left side and too low when hit from the right side of the paddle.

Correcting for the Errors

The performance error at each paddle location can be used to correct the aim for a hit from that paddle location. When the first hit was learned, the performance error was used to modify the aim. Now that we have data for hits from different paddle locations, we can correct the aim for many different hits. The aim for a particular hit can be corrected by using the performance error that occurred when a ball was previously hit from that location. To include this notion of state, Equation 4 can be rewritten

$$\mathbf{aim} = \mathbf{target} - \Delta\mathbf{aim}(x, y) \qquad (7)$$

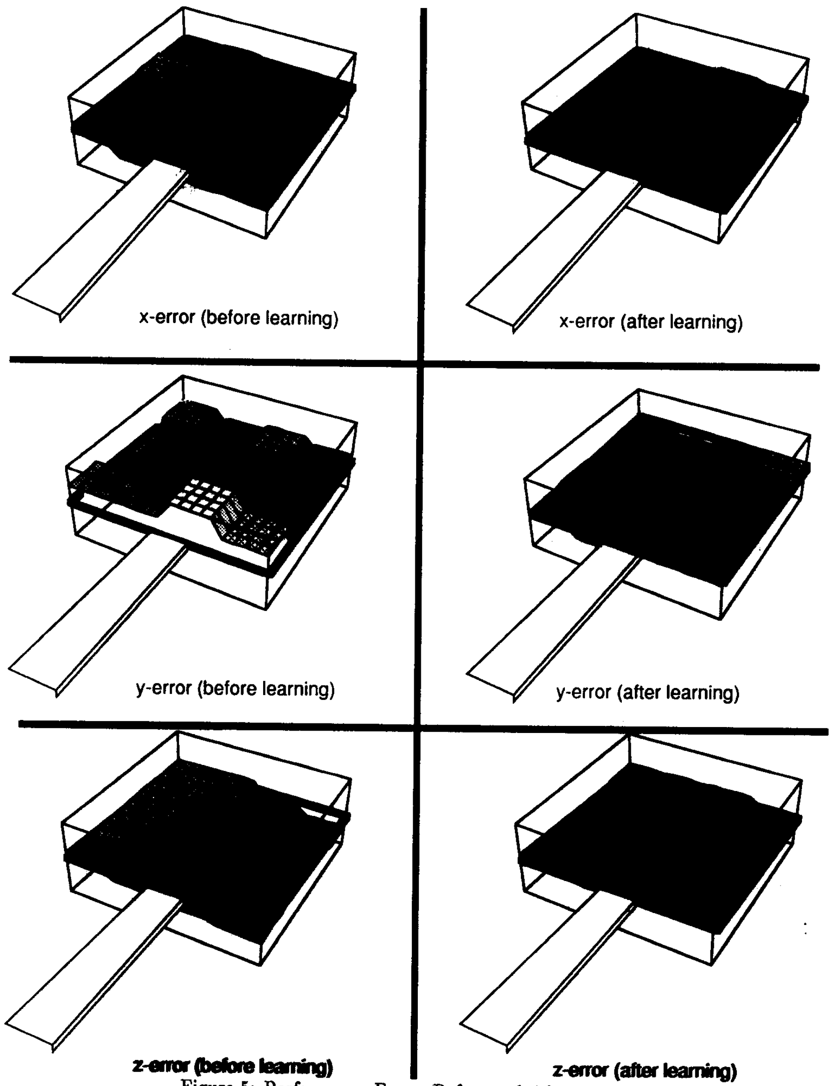

Figure 5: Performance Errors Before and After Learning.
Vertical side of outlined box is from -0.4 m to +0.4m.

The variables x and y denote part of the state of the system—the location on the paddle from which a ball is hit. Thus, a correction to the aim is calculated based on the state of the juggling system.

Implicit in Equation 7 is the suggestion that the state variables that best predict juggling errors are the x and y location from which the ball is hit. This is not necessarily the case. Another variable, such as velocity of the ball, could potentially be used to predict juggling errors. The choice to use x and y locations was made based on a qualitative assessment of which state variables best predict performance errors. A more quantitative study of the modeling errors has not yet been attempted. Choosing appropriate state variables to predict modeling errors is an important area for further research.

Once the state variables were chosen, we fit the performance errors using second order polynomials. This approach is similar to a method of approximating tabular data by polynomials that Raibert [1986] used to control a running machine. We could have used polynomials of another degree, or a different function altogether. Our rationalization is that second order polynomials seem to adequately describe the character of the errors. Once again, the choice of a function to fit the data is clearly an important part of the entire learning process and an important research issue.

State-Based, Task-Level Learning Experiments

State-based task-level learning was applied to eliminate all task errors—along the x, y, and z axes. The $\Delta\mathbf{aim}()$ functions are obtained by fitting second order polynomials to the x, y, and z performance errors shown at the left in Figure 5

$$\begin{aligned}
\Delta\text{aim}_x(x,y) &= -0.089 + 0.037\cdot x - 0.426\cdot y \\
&\quad - 0.772\cdot x\cdot y + 1.385\cdot x^2 - 0.603\cdot y^2 \\
\Delta\text{aim}_y(x,y) &= 0.092 + 0.188\cdot x - 1.020\cdot y \\
&\quad - 1.812\cdot x\cdot y + 0.037\cdot x^2 + 7.517\cdot y^2 \\
\Delta\text{aim}_z(x,y) &= 0.196 - 0.534\cdot x + 0.212\cdot y \\
&\quad - 1.648\cdot x\cdot y - 2.344\cdot x^2 - 2.886\cdot y^2
\end{aligned}$$

The fitting coefficients for these functions were 24%, 60%, and 76%, respectively. The small fitting coefficient for errors in the x direction indicate that the location from which the ball was hit is not a good predictor of the x errors.

With state-based task-level learning, the juggling performance of the robot dramatically improved. The juggling robot now averaged 21 hits on each run, with a low of 7 and a high of 70. The performance errors during these improved trials are shown in the right column of Figure 5. The errors after learning are more uniform and much closer to zero (grey coloring).

The approach described above is an attempt at *generalizing* task-level information. The method provides a correction to the aim for a ball that is hit from any location on the paddle. The basis for this correction is a series of hits, or experiences, that occurred at discrete points on the paddle. Once these points are fit to a polynomial, the experience is generalized to a hit from any location on the paddle.

5 Conclusion

We have demonstrated that task-level learning improves the performance of a juggling robot. The juggling system practiced the task, monitored its own performance, and adjusted its aim to better perform the task. The task-level algorithms were extended so that they could be applied to the multi-dimensional juggling task, and also take into account some of the state variables of the juggling system. This extended form of the task-level learning algorithms improved the performance of the juggling system from an average of 2 hits to an average of 21 consecutive hits. The robot system has learned what aim to use in order to hit the ball back to the center of the paddle. The required aim depends on where on the paddle the ball will be hit.

This preliminary study identified several issues involved in task-level learning of a dynamic task. A constant correction to the task-level command proved inadequate in compensating for modeling error. The modeling error depends on some or all of the state variables of the task. Identifying these variables is an unsolved research issue. We chose a part of the state of the robot system on an essentially ad hoc basis. Designing a robust method to generalize from past experience is also an important issue. Initially, look-up tables were used to generalize. Tabular methods proved sensitive to noise and the non-uniform distribution of previous experience. A polynomial surface fit to the data reduced these problems. It is not yet clear what limits the current performance of the robot juggling system. Further experimentation will aid in developing improved task-level learning algorithms. Analyzing the stability of task-level learning algorithms is an important research question.

Acknowledgments

This paper describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124, and the Office of Naval Research University Research Initiative Program under Office of Naval Research contract N00014-86-K-0685. Support for EWA was provided by an Office of Naval Research Graduate Student Fellowship. Support for SMD was provided by a Poitras Foundation Fellowship. Support for CGA was provided by a National Science Foundation Engineering Initiation Award and Presidential Young Investigator Award.

References

Aboaf, E.W., "Task-Level Robot Learning", M.S. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (September, 1988).

Aboaf, E.W., C.G. Atkeson and D.J. Reinkensmeyer, "Task-Level Robot Learning", IEEE Conf. on Robotics and Automation, (Philadelphia, PA, April 24-29, 1988).

An, C.H., C.G. Atkeson, and J.M. Hollerbach, *Model-Based Control of a Robot Manipulator* (MIT Press, Cambridge, MA, 1988).

Asada, H., and Youcef-Toumi, K., *Direct Drive Robots: Theory and Practice* (MIT Press, Cambridge, MA, 1987).

Atkeson, C.G., E.W. Aboaf, J. McIntyre, and D.J. Reinkensmeyer, "Model-Based Robot Learning", Fourth Intl. Symposium on Robotics Research, (Santa Cruz, CA, August 9-14, 1987).

Beer, F.P. and E.R. Johnston, Jr., *Vector Mechanics for Engineers* (McGraw-Hill Book Company, New York, NY, 1977).

Bühler, M. and D.E. Koditschek, "Robotics in an Intermittent Dynamical Environment: A Prelude to Juggling", Proc. IEEE Conf. on Decision and Control, (Los Angeles, CA, December 11, 1987).

Hollerbach, J.M., "A Survey of Kinematic Calibration", *Robotics Review* (MIT Press, edited by O. Khatib, J.J. Craig, T. Lozano-Perez, Cambridge, MA, 1988).

Inoue, H. and M. Inaba, "Hand-Eye Coordination in Rope Handling", First Intl. Symposium on Robotics Research, (Bretton Woods, NH, August 25 - September 2, 1983).

Liang, P., J.F. Lee, and S. Hackwood, "A General Framework for Robot Hand-Eye Coordination", IEEE Conf. on Robotics and Automation, (Philadelphia, PA, April 24-29, 1988).

Narasimhan, S. and D.M. Siegel, "The Condor Programmer's Manual: Version II", Artificial Intelligence Lab Working Paper 297, Massachusetts Institute of Technology, (July, 1987).

Raibert, M.H., *Legged Robots That Balance* (MIT Press, Cambridge, MA, 1986).

Shiu, Y.C. and S. Ahmad, "Finding the Mounting Position of a Sensor by Solving a Homogeneous Transform Equation of the Form AX=XB", IEEE Conf. on Robotics and Automation, (Raleigh, NC, March 31 - April 3, 1987).

Tsai, R.Y. and R. Lenz, "A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration", Fourth Intl. Symposium on Robotics Research, (Santa Cruz, CA, August 9-14, 1987).