Pinball: Planning and Learning in a Dynamic Real-Time Environment

Nathaniel S. Winstead **Alan D. Christiansen**

Department of Computer Science
Tulane University
New Orleans, LA 70118

Abstract

We are interested in the automation of planning and learning in dynamic physical environments. Recently we have begun to consider the game of pinball as a robotics testbed. Our intention is to build an autonomous learning agent that controls the game and improves its performance over time. We believe that pinball will give us a structured environment in which to investigate issues of planning and learning, while providing the real-world constraints of uncertainty and real-time control. Our work to date has led to a simple, physically realistic pinball simulation, which we are now using to test learning agents. In the future we will have a physical game interfaced to our computer system.

Why Pinball?

In the past, we have studied learning problems in robot manipulation (Christiansen, Mason, & Mitchell 1991). In this past work, the dynamic behavior of the tasks was of short duration, and could be abstracted from the learned models. We are now interested in tasks with a significant temporal dependency, rather than the (apparently) more static tasks we have previously considered.

One can argue that a pinball machine (Figure 1), when interfaced to a computer and a vision system to track the ball, is in fact a robot. The system has sensors (vision and contact sensors for the game targets) and effectors (the flippers). The pinball task has a goal (score lots of points), and automated planning to achieve the goal is required for performance above the novice level. And the pinball task is very dynamic.

Pinball is also interesting as a control task in that the player has only intermittent opportunities to affect the system state, unlike other well-studied control tasks such as pole balancing. Only when the ball is close to the flippers is there any chance to change the ball's velocity. (We have not yet implemented "bumping" the table.) When the ball is away from the flippers, the player is "at the mercy" of the environment, which is complex and not completely deterministic. Thus, credit assignment for good and bad actions becomes difficult.

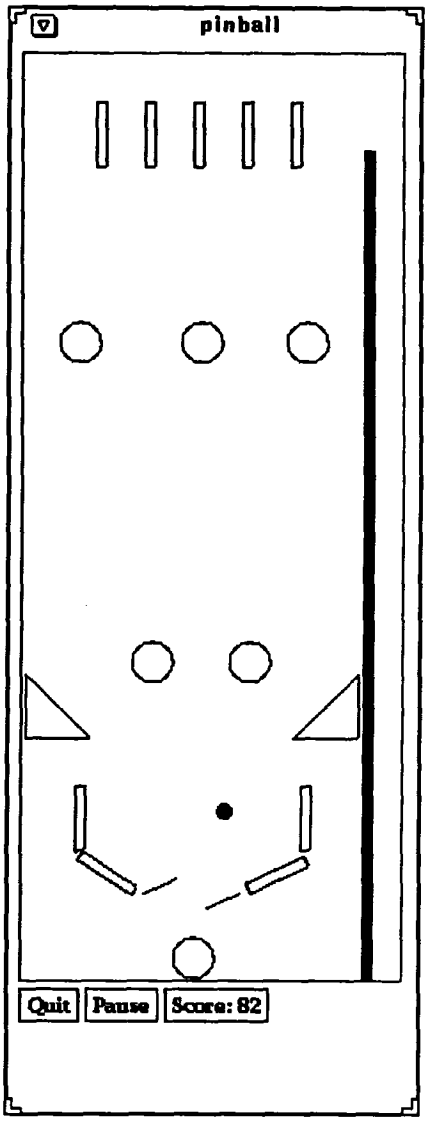

Figure 1: The pinball table layout.

Another advantage of the pinball task is that our system's performance can be compared directly to human performance. We can allow novice and expert players to play the game, and then compare those scores to the average score achieved by our pinball agent. Though it will be interesting to construct the best agent we can, based on our knowledge of the game, we remain interested in *learning* agents, which can improve their performance via practice in the task. We are hopeful that we can build an agent which initially exhibits rather poor performance, but eventually surpasses the ability of novice human players.

In approaching this problem, several questions concerning the organization/architecture of a player agent came to mind:

- Should the agent know Newton's Laws, or have some initial model of how the ball moves?
- Should the agent know the location and geometry of the obstacles?
- How much deliberation (vs. reaction) should the agent do?
- What sensors should the agent have?
- What effectors should the agent have?
- How should learned models be represented within the agent?

Our initially chosen responses are as follows (though we are not necessarily committed to these for the long term). The agent starts with no knowledge of how the ball moves, and no knowledge of the location or shape of obstacles. The agent tries to react to the ball quickly enough to keep it in play and score additional points. Deliberation/planning is allowed, as long as it does not impede the agent's ability to react to the environment. The agent senses the ball's instantaneous position and velocity, and its only actions are to flip (move the flippers up) or unflip (move the flippers down). With two flippers, this yields four distinct actions that the agent can take at any instant. Our initial approach is to use reinforcement learning (Q-learning, specifically), which means that the agent learns a map of the best action to take based on the current position and velocity of the ball.

The Simulator

Currently, we are working with a simulated pinball machine, which we created. The simulator software was written in C++ for a Unix/X-Windows environment. This simulator uses a simple, discretized time, physical simulation of the ball motion. Gravity acts on the ball at all times, and other than during collisions, the ball velocity and position are updated by integrating Newton's second law of motion ($F = ma$). Obstacles (bumpers) are either circular or convex polygons. Some bumpers add energy to the ball, causing the ball speed after a collision to be greater than the ball speed just before the collision.

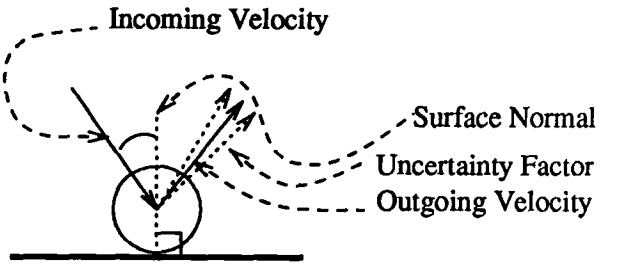

Figure 2: A simple model of how a collision occurs in our simulator.

Other simplifying assumptions were made for practical purposes in creating the simulator:

- Flippers move infinitely fast (i.e., at the time the flipper button is pressed, the flipper moves immediately to the position where it would contact the ball).
- All collisions in the simulator are modeled as elastic collisions, with various coefficients of restitution.
- Collisions are essentially deterministic, but include small added random factors.
- A ball is put into play by a fixed mechanism—the player does not have any control over this initial ball velocity. (A small random factor is introduced into the ball velocity.)

Collisions are modeled as shown in Figure 2. The incoming velocity is simply reflected about the surface normal of the obstacle, and the magnitude of the velocity is recomputed to allow for some energy having been absorbed (or added) by the obstacle, and a small random factor is injected into the velocity direction.

Planning and Learning

For the moment, our agents do no significant deliberation about how to attain future goals. We are currently focused on the problem of learning to score points, and learning to keep the ball in play. Of course, real pinball machines have targets which must be contacted in particular sequences in order to obtain "bonuses." It seems that a pinball playing agent must be able to shift its immediate goals over time, and to order those goals based upon the environment's "rules." We are very interested in the question of how to mix deliberation and reaction, though we have not yet addressed this issue.

The learning/planning technique adopted for our initial tests is called Q-learning (Watkins 1989). This technique uses a tabular representation of estimates of the reward that can be expected if a particular action is executed in a particular state. Over time, these estimates are refined based upon actual rewards received. The pseudocode for this algorithm can be seen in Figure 3. The "best action" to take from a particular state

$Q \leftarrow$ a set of values for the action-value function (e.g., all zeroes).

For each $x \in S : f(x) \leftarrow a$ such that $Q(x,a) = \max_{b \in A} Q(x,b)$.

Repeat forever:

1. $x \leftarrow$ the current state.
2. Select an action a to execute that is usually consistent with f but occasionally an alternate. For example, one might choose to follow f with probability p and choose a random action otherwise.
3. Execute action a and let y be the next state and r be the reward received.
4. Update $Q(x,a)$, the action-value estimate for the state-action pair (x,a):

$$Q(x,a) \leftarrow (1-\alpha)Q(x,a) + \alpha[r + \gamma U(y)]$$

where $U(y) = Q(y, f(y))$.

5. Update the policy f:

$$f(x) \leftarrow a \text{ such that } Q(x,a) = \max_{b \in A} Q(x,b).$$

Figure 3: Pseudocode for the Q-learning algorithm as presented in (Whitehead, Karlsson, & Tenenberg 1993).

is represented directly, so there is no search-based planning.

We initially had two ideas for providing rewards to our reinforcement agents:

- **Survival techniques** wherein the system's reward is based on the ball remaining in play between successive manipulations.
- **Scoring techniques** wherein the system's reward is simply based on the improvement in score between successive manipulations of the ball.

It should be clear that these two techniques reflect a response to two related and sometimes competing system goals. The two techniques reflect whether the agent defers its rewards (survival) or seeks to immediately achieve its goal (scoring).

Additionally, we believe that we will soon exceed the ability of current reinforcement learning algorithms to deal with multiple goals. We expect that the game will require our learning agent to pursue different goals at different times, and will therefore require an extension to standard methods along the lines of (Whitehead, Karlsson, & Tenenberg 1993).

Perhaps the most interesting aspect of this problem, from the point of view of artificial intelligence, is the creation of a learning method that will lead to improved playing capability by the agent. The demands

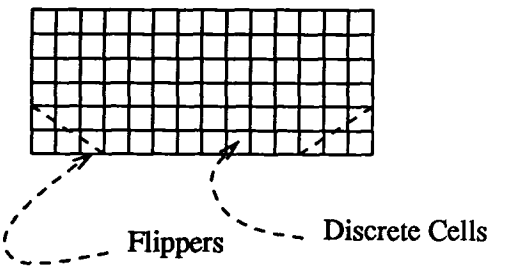

Figure 4: The area around the flippers is divided into discrete cells which hold action-value pairs for the reinforcement agent.

of the problem (real-time response, tolerance of uncertainty and errors) make it unclear if conventional learning methods (e.g., version space approaches (Mitchell 1982), decision tree induction (Quinlan 1986)) are appropriate. Promising methods include reinforcement learning (Whitehead, Karlsson, & Tenenberg 1993; Connell & Mahadevan 1993; Watkins 1989), neural networks (Barto 1989), stochastic automata (Simons *et al.* 1982), instance-based methods (Aha & Kibler 1989), and other statistical approaches (e.g., (Smith & Cheeseman 1986)).

Preliminary Results

We have implemented the Q-learning algorithm in C++ and interfaced it to our simulation. Currently the pinball agent flips both flippers simultaneously, and always returns the flippers to the down position (thus, "catching" the ball with the flippers is not currently possible). We represent state by a discretization of the ball position and velocity in the area of the flippers. Figure 4 illustrates the position discretization, which is actually 57 by 30 cells in the x and y dimensions, respectively.

To benchmark our learning agents, we created some agents implementing very simple strategies:

- **"Do-Nothing"** The agent never flips the flippers.
- **"Flail"** The agent moves the flippers up and down continuously.

In the environment depicted in Figure 1, the do-nothing and flail agents on average score approximately 60 and 102 points, respectively. The authors of this paper played the simulation interactively, and achieved on average 77 and 86 points. Several factors (e.g., computer network delays, delays in graphical rendering, limitations on human response time) make this not quite a fair comparison, though it is interesting that neither author did any better than the flail strategy.

In our initial experiments with reinforcement learning, we implemented agents with (1) a survival-based reward scheme and (2) a points-obtained reward scheme. The survival agent scored on average

102 points after training on 10,000 played balls. The points-reward agent scored on average 108 points after playing 10,000 balls. We are surprised that the difference in average score is as small as it is. (The non-learning flail strategy described above also scored about 102 points per ball.)

At this point, it is difficult for us to judge whether 108 points is a good score. It is good in the sense that it far exceeds the human players' scores, but it is not much different than the flail strategy, which we did not expect to be a very good strategy. Figure 5 shows a scatter plot of the scores obtained by the points-reward agent over 1000 balls (after learning for 10,000 balls). The scores vary a considerable amount, and though the agent exhibits some very good scores, the vast majority of the scores are low. This is partially explained by the nature of the pinball environment. On many occasions the ball goes out of play before the ball ever comes close to a flipper. Clearly, these lost balls are not the "fault" of the playing agent, and perhaps should not be counted when we measure the proficiency of the player. A redesign of the table layout could help to reduce the number of these "uncontrollable" balls. The high variance of scores obtained also makes it difficult to identify improvements due to the learning mechanism. When an average score rises, it may be the case that the improvement is a natural variation due to the random factors present in the simulation. More extensive experimentation will allow us to separate these factors.

Future Work

We anticipate the following activities in the coming months:

- Continued work on refinements to the simulator, with special attention to accurate collision models.
- Future agents are likely to use other established machine learning techniques so that we can benchmark any new techniques that we develop.
- Integrating goal-directed reasoning. Once we develop a model of the effects of agent actions, we must *invert* the model so that the agent can choose a *best* action. This process may be fully integrated with the learning, as is the case in reinforcement learning, or it may be completely separate from the learning process, such as the search-based method of (Christiansen 1992).
- Physical pinball. We will acquire a physical pinball machine, interface it to our computer, and use a commercial machine vision system to track the ball. The physical machine will allow us to better consider the problems of uncertainty and real-time control.

References

Aha, D. W., and Kibler, D. 1989. Noise-tolerant instance-based learning algorithms. In *International Joint Conference on Artificial Intelligence*, 794–799.

Barto, A. G. 1989. Connectionist learning for control: An overview. Technical Report COINS 89-89, University of Massachusetts-Amherst.

Christiansen, A. D.; Mason, M. T.; and Mitchell, T. M. 1991. Learning reliable manipulation strategies without initial physical models. *Robotics and Autonomous Systems* 8:7–18.

Christiansen, A. D. 1992. *Automatic Acquisition of Task Theories for Robotic Manipulation.* Ph.D. Dissertation, Carnegie Mellon University, School of Computer Science.

Connell, J. H., and Mahadevan, S. 1993. Rapid task learning for real robots. In Connell, J. H., and Mahadevan, S., eds., *Robot Learning.* Kluwer Academic Publishers. 105–139.

Mitchell, T. M. 1982. Generalization as search. *Artificial Intelligence* 18.

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

Simons, J.; Van Brussel, H.; De Schutter, J.; and Verhaert, J. 1982. A self-learning automaton with variable resolution for high precision assembly by industrial robots. *IEEE Transactions on Automatic Control* AC-27(5):1109–1113.

Smith, R. C., and Cheeseman, P. 1986. On the representation and estimation of spatial uncertainty. *IJRR* 5(4):56–68.

Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards.* Ph.D. Dissertation, King's College, Cambridge University.

Whitehead, S.; Karlsson, J.; and Tenenberg, J. 1993. Learning multiple goal behavior via task decomposition and dynamic policy merging. In Connell, J. H., and Mahadevan, S., eds., *Robot Learning.* Kluwer Academic Publishers. 45–78.

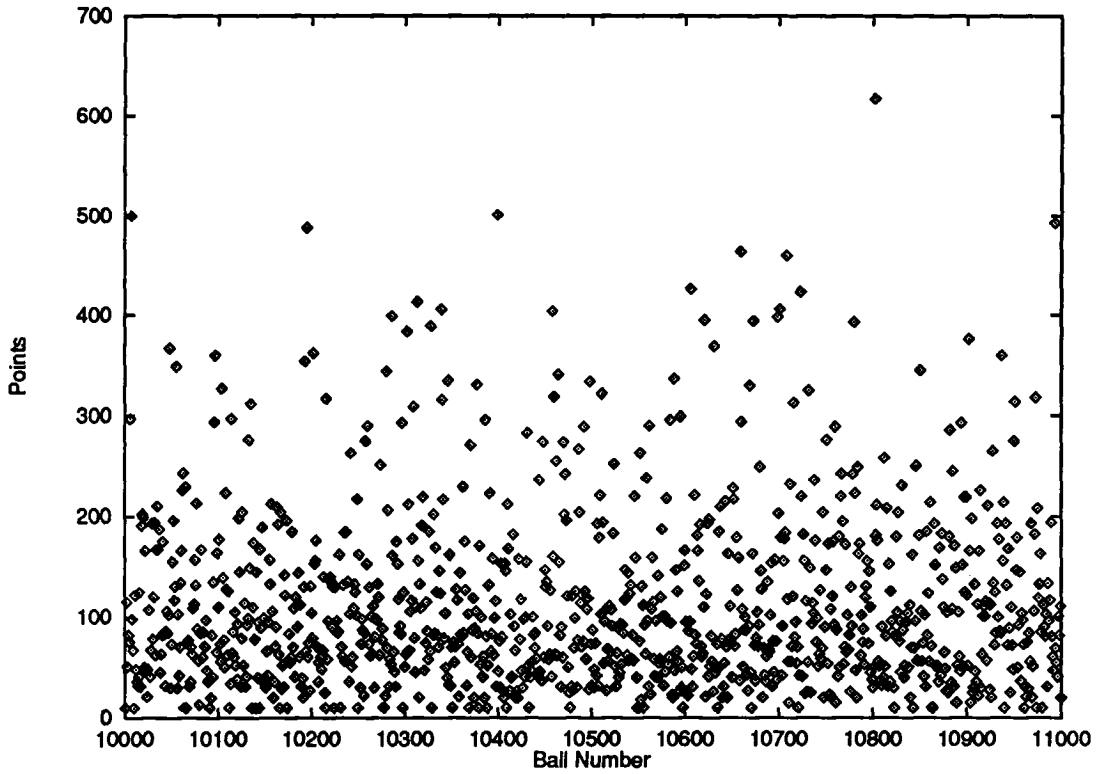

Figure 5: Scores obtained by the points-reward reinforcement learning agent (after learning for 10,000 balls).