Repeatability of Real World Training Experiments: A Case Study

Dean F. Hougen, Paul E. Rybski, and maria Gini
Department of Computer Science and Engineering
University of Minnesota
200 Union St. S.E., Minneapolis, MN 55455-0159
hougen@@cs.umn.edu rybski@@cs.umn.edu gini@@cs.umn.edu

Abstract

We present a case study of reinforcement learning on a real robot that learns how to back up a trailer and discuss the lessons learned about the importance of proper experimental procedure and design. We identify areas of particular concern to the experimental robotics community at large. In particular, we address concerns pertinent to robotics simulation research, implementing learning algorithms on real robotic hardware, and the difficulties involved with transferring research between the two.

1 Introduction

The project we describe in this case study uses a connectionist reinforcement learning scheme to learn to solve a control problem. The specific control problem is learning how to back a cab and trailer rig to a target location by steering the front wheels of the cab. The inputs into the robot consist of the hitch angle as well as the angle of the rear trailer to the target. The output is the direction to turn the steering mechanism. This system uses laterally connected artificial neural networks to improve the efficiency of a basic reinforcement learning mechanism. This system is capable of rapid learning of output responses in temporal domains through the use of eligibility traces and data sharing within topologically defined neighborhoods.

The method has been tested extensively in simulation and on a real autonomous mini-robot that we have constructed for this task. Doing multiple runs in simulation was easy and we collected large sets of data using different values for the initial distance from the target, angle to the target, and hitch angle. For each trial new initial conditions were chosen randomly at the start of the trial with a uniform distribution over the entire range of values. We experimented with different ranges and with slightly different learning modalities, and obtained a very high performance after a very small number of trials (less than 100 in most cases). Results have been reported in [Hougen, 1998].

Once we started using our mini-robot extensively for multiple experiments, a new problem appeared. Because the robot was used to collect data on a series of runs, each of 100 trials, it became extremely tedious for a person to have to pick up the robot each time and place it in the new starting position. To solve the problem, we decided to have the robot move forward autonomously while steering randomly to a new starting position after each trial. We built a ring for the robot to stay in, so the robot would never get farther away from the target than we desired.

However, as we started collecting data yet another new problem appeared. Our learning algorithm was failing to learn to handle certain initial positionings of the robot as well as it had in simulation. To understand what was happening, we collected the initial values for the hitch angle and angle to the target for each trial. We discovered that despite using a random movement to position the robot to start a new trial, we were not generating a uniform distribution of initial starting values. This was sufficient to significantly decrease the number of large hitch and goal angles in the set of initial positions encountered during a run. This, in turn, decreased the system's ability to learn these rarely encountered initial states. Worse, it meant that, since learning and testing are done together in the system, the testing conditions did not include a large number of difficult starting positions — those in which the rig starts with large hitch and goal angles — instead they consisted mostly of easier initial positions. The average success rate of the system after learning was, therefore, quite good, but the testing conditions were not sufficient to thoroughly test the system. To solve this problem, we built a robot that could pick itself up and place itself in the required starting position (fed to it from the workstation used to record the results). The mechanical design of the lifting mechanism is quite interesting, considering the limited size of the robot.

In solving this problem, we identified several important issues regarding experimental methodology and how it pertains to robots in simulation and robots in the real world. We have developed a robotic control system under simulation and then transferred it to a real robot and observed the results. What we learned was that certain assumptions that were made under simulation could not be made on the real robot. We also learned that subtle details observed during robotic experiments can provide insights into complex interactions between the robot and its environment which may affect its performance. We believe that we have uncovered a potential problem with such learning systems not previously discussed in the literature: namely, that the influence of random events which affect the distribution of training samples in real-world experiments can be sufficiently different from distributions commonly used in simulation so as to cause significant deviations in the performance of a learning system. We observed this only after performing a *large* number of trials on the real robot, so limited experiments on real robots might not allow for the detection of the type of anomaly that we observed here.

2 The Learning Method

Control-learning for autonomous robotic systems is challenging because the learning system must be

1. robust enough to overcome the problems of noisy input data and uncertain interactions between motor commands and effects in the world,
2. able to give responses in real time, and
3. able to learn in a short period of time.

When using mini-robots, another challenge is to make the learning system compact enough to fit into available on-board memory.

Connectionist control-learning systems have recently received much attention, but most of the work has concentrated on simulated systems. Approaches such as the Cerebellar Model Articulated Controller [Shelton and Peterson, 1992], adaptive fuzzy systems [Kong and Kosko, 1992], backpropagation through time [Nguyen and Widrow, 1990], and "fuzzy BOXES" [Woodcock et al., 1991] have all been applied to versions of the trailer-backing problem. Our method generally learns faster or with less supervision than these other methods while remaining practical for implementation with minimal computing hardware.

Our method combines the self-organizing capabilities of Kohonen Maps [Kohonen, 1989] with the temporal sensitivity of eligibility traces. Learning requires constructing a mapping from the input space to output space. The input space is two-dimensional. The inputs are (1) the angle from the spine of the trailer to the goal, called the *goal angle*, and (2) the angle between the cab and the trailer, known as the *hitch angle*. The output space is one-dimensional but thresholded at zero, giving the system bang-bang steering (i.e. the steering wheels may only be moved to 30° left or right).

The system receives feedback from which rewards and punishments can be generated for reinforcement learning. If the rear of the trailer reaches the goal, success is signaled. If the angle to the target exceeds 90° or the angle of the hitch exceeds 90°, failure is signaled. No other sensory data or feedback is available.

2.1 Topology and Neighborhoods

The topology of the learning system is shown in Figure 1. The learning system used consists of two layers — an input layer to categorize the input angles and an output layer to determine the appropriate response for a given classification. Both layers consist of a number of simple, neuron-like computing elements arranged in networks with fixed topological layouts. In the input layer there are two separate eight-element networks, each

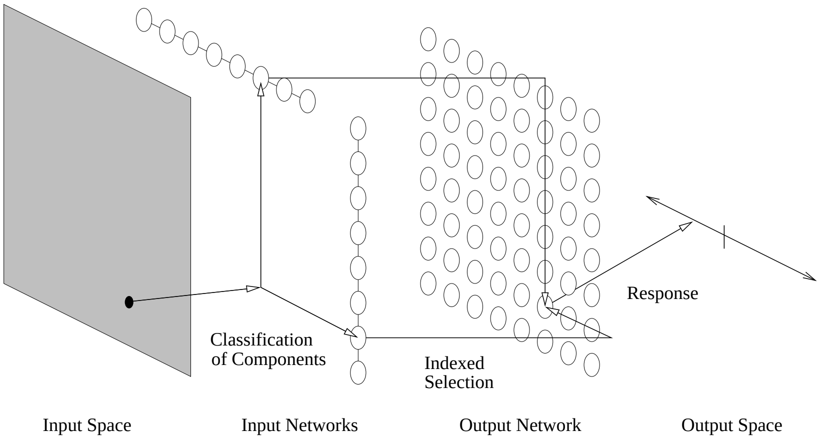

Figure 1: A mapping from input space to output space using component-wise classification and indexed partitioning.

with a one-dimensional, linear topological arrangement. In the output layer there is a single two-dimensional network with a rectangular topological arrangement.

The existence of a network topology allows for the definition of a *distance* function for the elements. In similar systems the distance function used is typically the Euclidean distance between coordinates in topology space. For the sake of computational efficiency, however, the maximum difference between coordinates of the elements was used.

The distance function is used indirectly through the definition of *neighborhoods.* The neighborhood N of neural element n on trial T is defined as

$$N_n(T) = \{u \in U \mid d(n,u) \leq W(T)\} \tag{1}$$

where U is the set of elements u in the network, d the distance function defined on topology space, and W a trial dependent function specifying the width of the neighborhood. W starts large and decreases with trial number.

The system is trained in a series of *trials.* A trial is divided into discrete time units, and lasts from an initial positioning of the cab and trailer rig until a success or failure signal is generated. During a *run* of multiple trials, the learning system progresses from random performance to competence.

2.2 The Input Layer

The two networks of the input layer separately handle the two input angles. Each element of each network has a sensitivity region defined for it. We call the variable used to store the value of an element's sensitivity region its *weight.* Each element has a single weight, and when, on each time step, a new pair of input angles is sensed, these are compared with the weights of the elements in the appropriate network. This comparison is done using a similarity measure S that is simply the absolute value of the difference between the values. The element s for which S is the smallest is *selected.* Formally,

$$\exists s[S(w_s,x) \leq S(w_u,x) \mid \forall u \in U, s \in U] \tag{2}$$

where w is a neuron's weight vector and x is the input value. If more than one neuron satisfies this equation, then the one with the lowest number is selected. In effect, the sensitivity regions partition each dimension of the input space into a number of discrete regions.

The sensitivity region weights are initially random within the range of expected input values. As new input values are given to the system, the elements change their sensitivity regions in a self-organizing manner, based on Kohonen Maps [Kohonen, 1989]. The weights of the selected neural element and of all other elements in its neighborhood are updated to match the input even more closely using

$$w^{new} = w^{old} + \alpha(t)(x - w^{old}) \tag{3}$$

where w is the weight being adjusted, x is the input, and α is a time-dependent function that determines the influence of the input ($0 \leq \alpha \leq 1$). α starts near 1 and decreases with time. In this way, there is competition amongst all the elements in a network to be the one selected, and cooperation within a neighborhood as the elements change their weights towards the same target. By repeated presentations of data to the network, the network self-organizes so that closely neighboring elements have similar values for their weights (and therefore respond to similar input) while elements that, according to the network topology, are far from one another have very different values for their weights (and therefore respond to very different input).

2.3 The Output Layer

The selection of one element from each of the two networks in the input layer allows for the selection of a single element in the two-dimensional output network. In effect, the input networks index into the output network. The selected element in the output network then *fires*, returning its output weight value as the control action for the system. If this value is positive the system turns the steering wheels to the right by 30°, otherwise the steering wheels are turned left by 30°.

Initially, all elements in the output network are given random weight values. The system learns good output responses for the neural elements through reinforcement. At the end of a trial, all neural elements that fired during that trial have their values adjusted. On failure they are punished, which pushes their values towards the negative of what they were during the trial. On success they are rewarded, which pulls their values in the direction of their sign.

The degree to which a neural element has its output values adjusted on trial completion is determined by its eligibility for adaptation. When a neural element fires it becomes eligible for change according to the equation

$$e^{new} = e^{old} + I \tag{4}$$

where e is the eligibility of the neuron for adaptation, and I is the initial eligibility value of a neuron that has just fired. This eligibility decreases over time but produces a trace that allows for adaptation, according to

$$e(t+1) = \delta e(t) \tag{5}$$

where δ is the rate of eligibility decay ($0 \leq \delta \leq 1$). The more frequently an element fires during a trial, and the more recently it fires before the trials ends, the greater its adaptation.

When a success or failure signal is received by the output network, all of the weights of all of the output elements are updated according to

$$v^{new} = \text{sign}(v^{old})(|v^{old}| + e\,\sigma(T)\,f) \tag{6}$$

where v is the weight, e is the eligibility for adaptation, σ is a scaling function that changes with the trial number T, and f is the feedback signal (+1 for success, -1 for failure). The direction of change is determined by the failure or success signal; the magnitude of change is determined by the eligibility. The scaling function σ is used to allow for large changes to the weights in early training trials and smaller changes in subsequent trials.

Because adjacent neural elements in the grid respond to adjacent partition regions in the input space, there is a good chance that similar responses for these elements may be helpful. By having neural elements communicate their output values to one another, elements may learn their behavior in part from the punishments and rewards received by their neighbors. Therefore, after the completion of a trial each element updates its weight a second time, this time based on the weight values of the other elements in its neighborhood, according to

$$v_i = (1 - \beta(T))v_i + \beta(T) \sum_{n \in N_i} \frac{v_n}{m} \tag{7}$$

where each v_i is a weight, N is the neighborhood of element i, m is the number of neural elements in that neighborhood, and β determines the degree to which an element's value is "smoothed" with that of its neighbors ($0 \leq \beta \leq 1$). This inter-neural cooperation allows the system to learn roughly ten times as fast as without this information sharing. β starts near one and decreases with time. This means that each element's value becomes more independent from those of its neighbors as time passes.

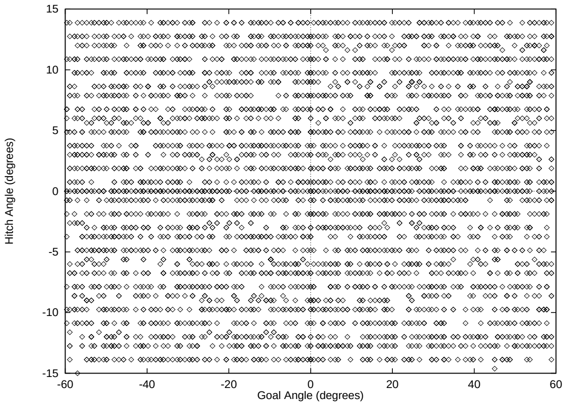

Figure 2: Uniform distribution of initial starting values.

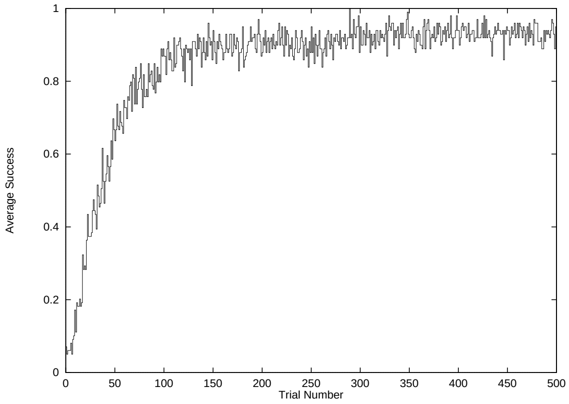

Figure 3: Average performance for 100 runs of 500 trials each using cooperative response learning with a learned eight by eight partitioning on the truck-backing task.

2.4 Experimental Results in Simulation

The system was tested extensively in simulation and, to a smaller extent, on the real robot.

For the simulation testing, the cab-trailer rig was started with the front of the cab six feet from the target, with an angle to the target between $-60°$ and $+60°$, and with a hitch angle between $-15°$ and $+15°$. New initial conditions were chosen randomly at the start of each trial with a uniform distribution over the entire range.

Figure 2 shows the distribution over 10,000 trials of the robot. The x-axis represents the allowed goal angles of $\pm 60°$ and the y-axis represents the allowed hitch angles of $\pm 15°$.

Using sixteen neural elements in the input network and sixty-four neural elements in the output network, the system is able to achieve remarkable success. Average results for 100 runs on this task, each starting from random performance and progressing to competence, are shown in Figure 3.

An average success rate of greater than 85% is achieved by the system after only 100 trials and an average success rate of nearly 95% is achieved by the system after roughly 300 trials.

As can be seen from the graph, the system quickly improved its performance, reaching its maximum performance within 300 trials. (While a realistic comparison of systems cannot be made it might be of interest to know that other authors have used as many as 20,000 trials to train their systems to control trailer-backing). In fact, the success rate of completely random control for the given initial conditions was determined experimentally to be zero, and the success rate using control by a random network without learning was determined to be approximately 6%, so it is obvious that significant learning has taken place within the first twenty trials. The steep initial slope of the graph indicates that significant learning occurred on failure as well as success.

3 Real World Experiments

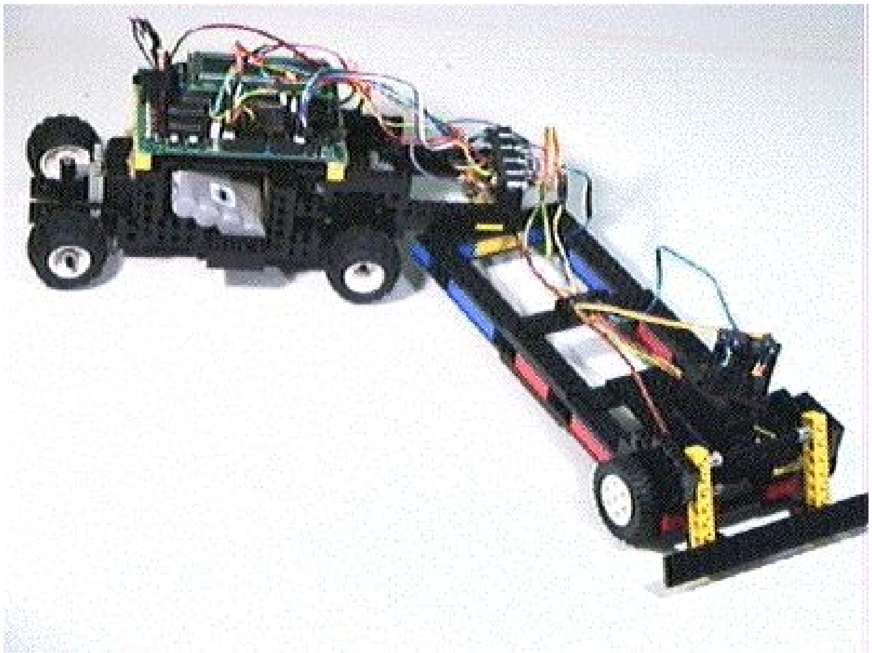

Figure 4: TBMin, the trailer backing mini-robot.

The Trailer-Backing Mini-robot (TBMin, seen in Figure 4) was built to extend the learning algorithm out of simulation and into the real world. The TBMin robot consists of two parts: a 4-wheeled Ackerman-steering cab unit and a trailer unit. The trailer is attached to the cab by a potentiometer hitch which lets the robot know the angle between the cab and the trailer. A light-tracking turret is mounted on the back of the trailer to keep track of the angle to the target. TBMin's on-board computer is a Motorola 68HC11-based Handyboard microcontroller [Martin, 1998]. The entire neural network architecture (data structures and supporting source code) easily operates within the on-board 32K of RAM.

3.1 Training the Robot

The environment in which the robot conducts its experiments consists of an enclosed circular arena 6' in radius with a target light bulb placed at the center. The robot starts each run at the perimeter of the arena and backs its trailer towards the target bulb.

A success condition occurs when the robot touches its rear bumper to the target. One of two types of failures occurs when the cab and trailer jackknife. This situation occurs when the measured hitch angle comes close to its physical limits of ±90° with respect to the cab. The second type of failure occurs when the light-tracking turret reaches its physical limits of ±90°, and the turret can no longer follow the target. Both of these failure conditions are identical from the perspective of the learning algorithm. When either a success or failure condition is reached, the robot stops its motion and applies the learning algorithm.

3.2 Initial Positioning Problems

Re-initializing the simulated trailer-backer is a simple matter of generating a new set of initial angles for the hitch and goal angles and re-drawing the robot on the screen. Re-initializing TBMin involves the robot pulling itself forward after a trial and randomly changing the position of its steering wheels. This random steering was intended to provide the robot with a new set of initial starting angles at the beginning of each trial. However, after a few hundred trials it was observed that the robot was failing on large initial angles that the learning system trained in simulation was able to solve. This pattern continued and the observation was made that some difference in the experimental setup between the robot and the simulation was causing this failure.

3.3 Back to the Simulation

Modifying the simulation to mimic the robot's automatic random positioning mechanism, a series of experiments was run. Figure 5 shows the distribution over 10,000 trials of the robot. The x-axis represents the initial goal angles of ±60° and the y-axis represents the initial hitch angles of ±15°. As can be seen, this distribution has a severe bias towards points near the origin.

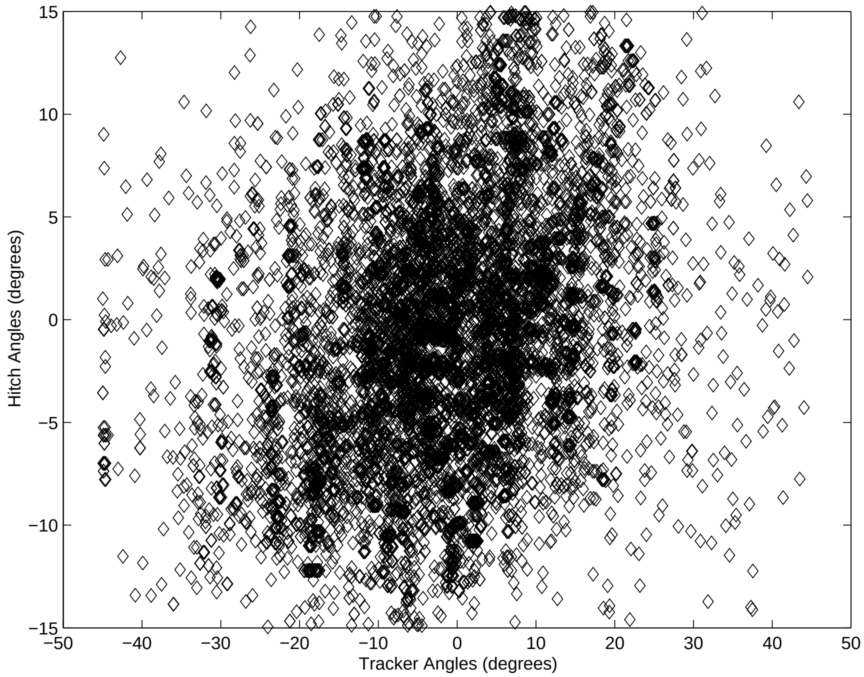

Figure 5: Non-uniform distribution of initial starting values.

After analyzing the convergence results from a series of simulation experiments, it was found that the training cases being generated by the robot's new initial configuration generation program were not providing the learning system with enough test cases to produce a general solution to the problem. Thus, the learning system was converging to a sub-optimal solution.

At first glance, the results of the uniform angle distribution runs (Figure 3) and the non-uniform angle distribution runs (Figure 6) do not appear to be very different. The graphs of the two learning curves seem to show that both methods are learning roughly equally well. What it doesn't show, however, is that the pull-forward method is generating "easier" starting positions.

To test this, two separate copies of the learning system were trained. The first was trained on a uniform distribution of starting angles and the other was trained on a non-uniform distribution obtained using the pull-forward method. After training, the systems were then "frozen" so that their input and output weights did not change. These were then tested using a uniform distribution of starting angles. The first column of Table 1 shows the results of this test. Clearly, the copy trained on the uniform distribution of initial angles has learned a better solution for this test set than the copy trained on the non-uniform distribution (93.65% success versus 79.29% success).

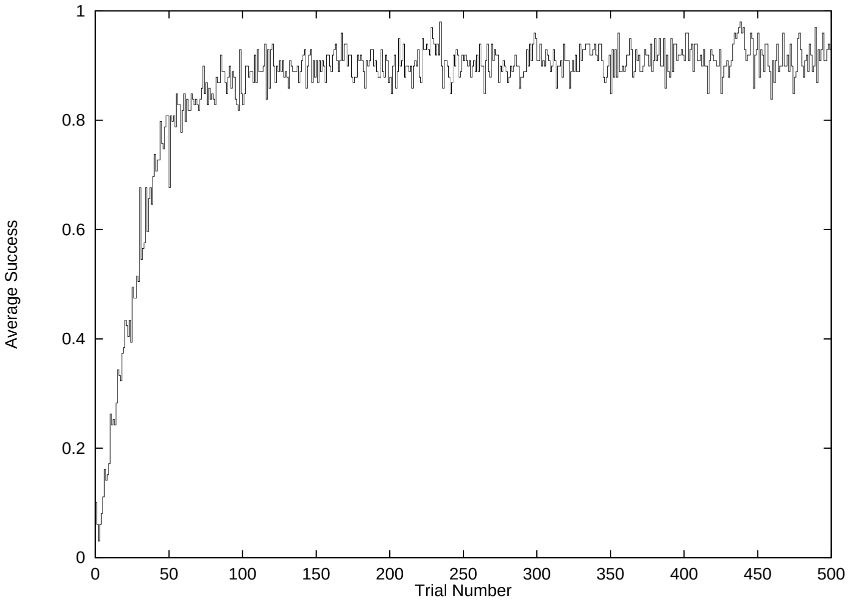

Figure 6: Average performance for 100 runs of 500 trials each using cooperative response learning with a learned eight by eight partitioning on the truck-backing task and a non-uniform distribution of the starting angles.

Then, to test the generalizing capabilities of the two different training methods, each copy was tested on a non-uniform distribution of starting angles obtained using the pull-forward method. The results of this test are found in the second column of Table 1. Again the copy trained on the uniform distribution of initial angles learned a better solution than the copy trained on the non-uniform distribution (98.19% success versus 95.19%).

Together these results plainly show that the copy of the learning system trained on the uniform distribution was able to generalize the problem better and produce a better solution than the copy trained on the non-uniform distribution. However, these results are masked in the typical reinforcement learning experimental procedure, in which learning and testing are combined and only the success rate of the system as it learns is observed.

	Uniform Test Set	Non-uniform Test Set
Uniform Training Distribution	93.65 %	98.19%
Non-uniform Training Distribution	79.29 %	95.19%

Table 1: Success Rate for Training and Testing on Uniform and Non-uniform Distributions

4 Robust Experimental Procedure

At this point we had several choices. Among them were to

1. modify the experimental procedure to separate the training and testing phases,
2. allow the set of initial positions on the real robot to be dominated by easy cases to the exclusion of difficult ones, or
3. modify the robot to allow us to select its initial positions at will.

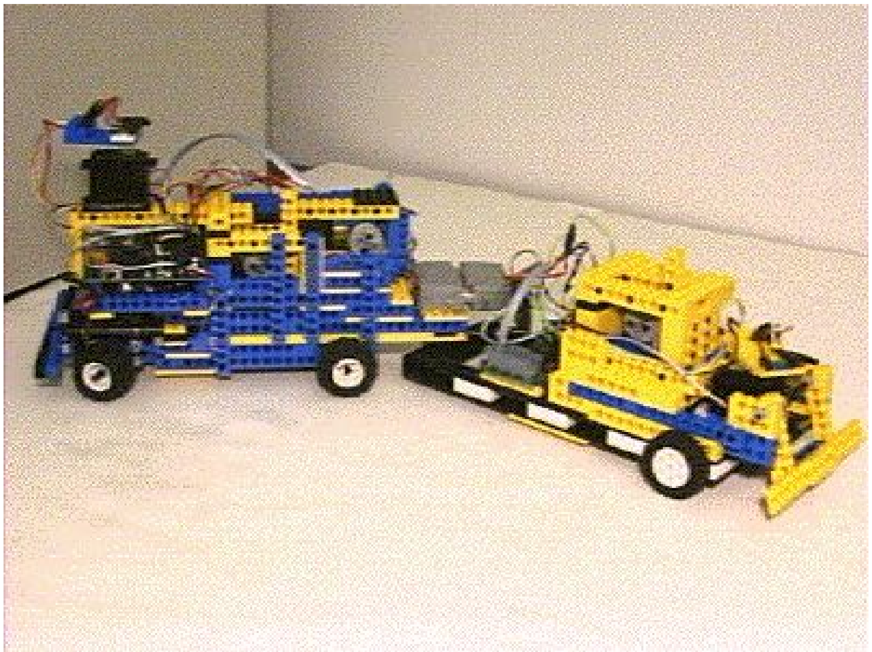

Figure 7: The Self-Positioning Trailer-Backing Mini-robot.

Separating training from testing was useful in discovering the effects of different training sets on the generalization capabilities of the learning system but contradicts the idea of reinforcement learning as a method by which a system may improve its performance while executing a task. Allowing the set of initial positions to be dominated by easy cases would not allow us to see if the robot was capable of achieving good performance under difficult test conditions. So, we chose the third option.

The method employed by TBMin of having the robot automatically set its position was not controllable enough to be made useful. A new mechanical system was required that would automatically reposition the robot accurately and repeatably. The solution to this problem was the design and construction of a new trailer-backer robot, shown in Figure 7. This robot, called the Self-Positioning Trailer-Backing Mini-robot (SPTBMin), had the same physical drive characteristics of the original TBMin (size, length and turn radius), but was fitted with a novel automatic re-positioning mechanism.

4.1 Hardware

Like TBMin, SPTBMin used a Handyboard to run the learning system software but used another Motorola 68HC11-based microcontroller, the Miniboard [Martin, 1995], to control its actuators. An additional light-tracking turret is placed on the cab for reasons described below.

Each segment of the robot is outfitted with a linear actuator which raises the segments' rear wheels off of the ground. At the base of this linear actuator is an independently-powered drive system which is mounted perpendicular to the other wheels of the segment. Figure 8 and Figure 9 illustrate the layouts of the segments as seen from the top and from the side, respectively. Using this linear actuator and drive system, each segment can pivot about a point at its front (the front wheels for the cab and the hitch for the trailer), allowing it to re-orient itself as necessary.

4.2 Auto-Positioning

Using the new robot, the learning experiments proceed as normal. The robot backs up, keeping its trailer-mounted light-tracking servo pointed at the target light. When it succeeds or fails and adjusts its weights, the robot pulls forward, using its cab-mounted light-tracking servo to steer it directly away from the light bulb. It continues driving away until its front bumper collides with the circular rim of the arena. At this

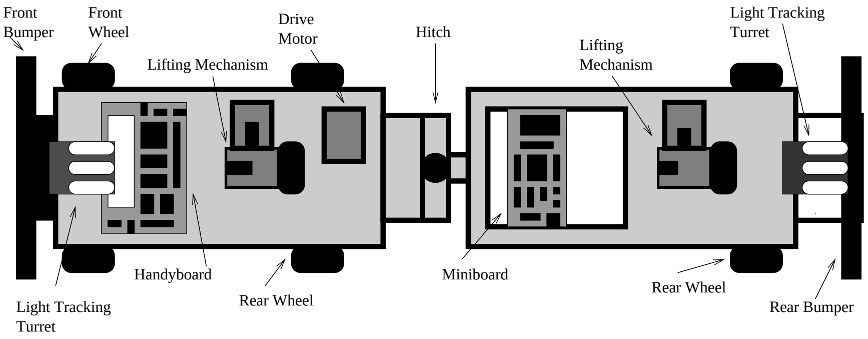

Figure 8: Top views of the cab and trailer segments.

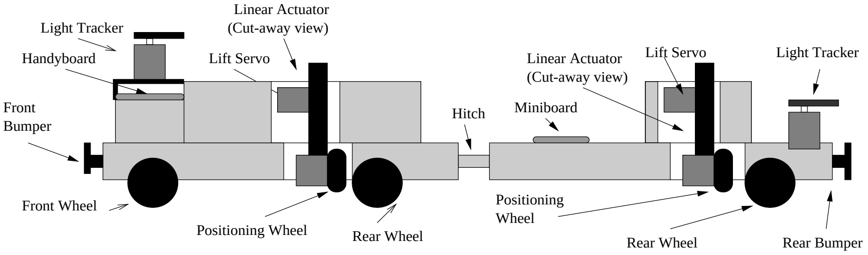

Figure 9: Cut-away side views of cab and trailer lifting mechanism.

point, a set of initial hitch and tracker angles is generated from a uniformly-distributed random number generator and downloaded to the robot.

Once the angles have been received, the robot uses them to determine how to position itself properly. When the robot re-positions itself for the next trial, it must first determine what position the cab should be in and put it there before it can position the trailer segment. This position is determined by the angle that the front of the cab makes to a line tangent with the rim of the arena. This particular angle is not part of the initial random angles that are given to the robot and thus must be derived from the hitch and tracker angles.

4.3 Modeling the Robot

The physical configuration of the system is modeled in the following way to solve for the appropriate angles. Figure 10 shows a diagram that represents the configuration of the robot with respect to the light:

θ_2 and θ_1 are measured quantities which are determined when the robot stops at the rim of the arena. D_c is simply the radius of the arena (6' for this set of experiments). L_c and L_t are known a priori and do not change. From these known values, the remaining values can be solved for in the following sequence. (It should be noted that the following equations describe the robot pose shown in Figure 10 and thus do not give proper treatment to the signs of the angles involved for the general case.)

$$\begin{aligned}
\gamma_L &= 270 - \theta_1 \\
L &= \sqrt{L_t^2 + L_c^2 - 2L_tL_c\cos(\gamma_L)} \\
\gamma_c &= \cos^{-1}\left(\frac{L_t^2 + L^2 - L_c^2}{2L_tL}\right) \\
\gamma_t &= 180 - (\gamma_c + \gamma_L) \\
\gamma_{D_c} &= 270 - (\theta_2 + \gamma_c) \\
temp_1 &= L\cos\gamma_{D_c} + \sqrt{L^2\cos^2\gamma_{D_c} - L^2 + D_c^2}
\end{aligned}$$

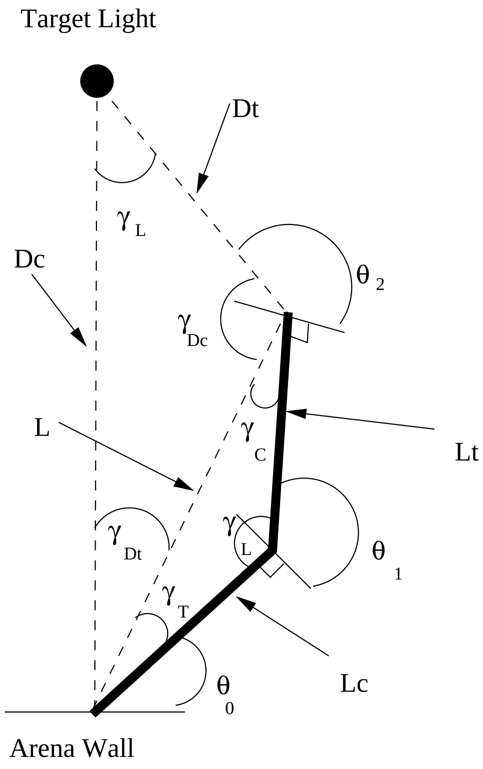

Figure 10: Determining angles for self-positioning.

$$
\begin{aligned}
temp_2 &= L\cos\gamma_{D_c} - \sqrt{L^2\cos^2\gamma_{D_c} - L^2 + D_c^2} \\
D_t &= \begin{cases} temp_1 & \text{if } D_c^2 = temp_1^2 + L^2 \\ temp_2 & \text{if } D_c^2 = temp_2^2 + L^2 \end{cases} \\
\gamma_{D_t} &= \cos^{-1}\left(\frac{L^2 + D_c^2 - D_t^2}{2LD_c}\right) \\
\theta_0 &= 90 - (\gamma_{D_t} + \gamma_t)
\end{aligned}
$$

The final value, θ_0, represents the angle to which the cab must move with respect to a line tangent to the rim of the arena. The cab's light-tracking servo head is mounted directly over its pivot point (directly between the front wheels), which lets the robot know its angle with respect to the target and move to it. When the cab self-positions, it drags the trailer along with it. If the trailer jackknifes during this procedure, it picks itself up and positions itself directly in line with the cab. Once the cab is in position, the trailer segment can use its own self-positioning mechanism to line up the hitch and rear light-tracker angles to the appropriate values of θ_1 and θ_2. If the error between the requested and measured values of θ_1 and θ_2 is greater than some given tolerance, then the robot knows that an error has taken place in its self-positioning procedure and can re-do it or summon human assistance.

5 Discussion

Several important points regarding developing learning algorithms for real robots were made very clear to us in this case study. First, there is the difficulty of trying to make correct assumptions when designing a simulation to be used for developing learning systems that are ultimately to be used on real robots. In particular, we believe we have uncovered a potential problem not previously discussed in the literature. This is the problem of sampling bias effecting the learning and testing of a reinforcement learning system. Second, there is the extreme importance of observation of the robot as it learns. Without direct observation of the robot, we would not have uncovered the problem that we did.

Both of these points emphasize the need for extreme care when running experiments on robots, simulated or real. If the research community is to fully benefit from such work, there are important issues that must be properly addressed. Primarily, care must be done to try to identify and isolate subtle environmental factors that may affect the behavior of a reinforcement learning algorithm in unexpected ways. In particular, without good experimental design and analysis, reported data and results may be inadvertently affected.

5.1 Learning in Simulation and on Real Robots

Simulation is an invaluable tool for the development of robotic systems. Like all powerful tools, however, it must be used carefully and wisely. The use of simulation in reinforcement control-learning systems is quite common and dates back to the earliest research in this area, from the original reinforcement-learning pole-balancer [Michie and Chambers, 1968], and many other reinforcement learning approaches to this same problem [Russell and Rees, 1975, Barto et al., 1983, Selfridge et al., 1985] through reinforcement learning for truck and trailer backing [Woodcock et al., 1991, Koza, 1992] to docking the space shuttle with a satellite [Jani, 1993].

More recently, however, there has been interest in reinforcement learning systems that go beyond simulation. In these cases simulation may be used to develop a learning system capable of learning on a real robot (e.g. [Benhrahim et al., 1992]) or the system may do its learning in simulation and use the learned solution to perform a task in the real world [Asada et al., 1996]. Because this emphasis is quite recent, potential pitfalls of these methods are largely unknown. Nonetheless, problem areas are beginning to be uncovered.

One such problem that has been previously uncovered is the effect that differences in the physics of the simulation and real world can have on the solutions learned. If the robot learns in simulation and acts in the real world, performance on the real robot may not be as proficient as it is in the simulated environment [Asada et al., 1996]. This is quite similar to the effect encountered in systems that do not learn but are programmed based on simulation models.

If the differences between the simulation and the real world are too great, the solution learned in simulation may not be worthwhile and learning must take place on board the real robot [Mahadevan and Connell, 1992].

When the learning system is developed in simulation but learning takes place in the real world, the problem becomes somewhat different. In this case, it is the learning itself that may be degraded, rather than the final performance of the system. (The final performance of the system may still be degraded, of course, if the learning system fails to arrive at as satisfactory a solution.) In some situations, differences between the simulation and the real world can be overcome by "tuning" the learning system's parameters to fit the real world. Unfortunately, there may be no known systematic way to make these adjustments [Jervis and Fallside, 1992].

Another problem that has been recognized, this one specific to systems where learning takes place on board a real robot, is that the large number of trials that many reinforcement learning systems require is simply not practicable in the real world [Jervis and Fallside, 1992]. This has motivated much of our prior research in producing faster reinforcement-learning systems [Hougen et al., 1994, Hougen et al., 1996, Hougen et al., 1997a, Hougen et al., 1997b] and has motivated similar research by other authors [Lin and Kim, 1991, Mahadevan and Connell, 1992, Kretchmar and Anderson, 1997]. It has also lead to research in learning single behaviors that can be learned separately and are not effected as much by initial conditions (e.g. [Matarić, 1997]). Even in these cases learning can take hundreds or thousands of trials and the performance may still not exceed 80% success [Colombetti et al., 1996].

The problem that we have uncovered in our present study is also unique to systems that learn in the real world. This is the problem of the distribution of training samples. In simulation it is very easy to choose the distribution of your training samples. A uniform, normal, or other simple distribution may be used as a matter of habit, since these are easily generated using pseudo-random number generators. However, the distributions encountered in the real world will often not be so simple, particularly in robot control situations in which the control signal may affect the next sample in some non-random way.

This is certainly similar to the problem of sampling bias often encountered in empirical sciences [Cohen, 1995], but with a new twist. In this case it is not a matter of sampling bias preventing a researcher from seeing an effect (or of making an effect appear where one is not really present). Instead, the difference in sample distribution between simulation and the real world may cause the learning system itself to learn a different solution. That is, the effect is one more step removed from the researcher — the different distribution is causing the learning system, and only indirectly the researcher, to see the world differently.

In fact, because the sampling distribution was biased for both learning and testing (as these phases are traditionally combined in reinforcement learning), while the learning system saw the world differently and learned a quantitatively different solution, the results appeared the same from the researcher's perspective, when viewed through the traditional lens of reinforcement learning — the success rate. It was only when the researchers looked beyond that measure to the movements of the robot in the real world that the anomaly was detected.

None of these problems is particularly surprising but we believe that they are important to bear in mind when doing research on learning in real robots.

5.2 Observation of Real Robots

Many robotics researchers believe that tests of their methods on real robots are vital to proving their correctness, as no simulation can capture all aspects of the real world. We wholeheartedly agree with this position and believe that it is just as important for robot learning. What we have discovered with this case, further, is that observation of the real robot as it learns may be vital as well. As pointed out previously (Subsection 3.3), if only the performance rates of the learning system given the two distributions were compared (Figures 3 and 6), very little difference would have been observed. In fact, the tested performance of each system on the distribution under which it was trained was also found to be very similar. Looking at Table 1, the performance of the network trained on a uniform distribution and tested on a uniform distribution (93.65% success) was actually slightly worse than the performance of the network trained on a non-uniform distribution and tested on a non-uniform distribution (95.19%). The only reason we detected that the system was learning a better solution with the uniform distribution was that, while watching the system learn on the real robot, it was noticed that for "difficult" starting positions the robot was rarely succeeding. However, we knew from previous experimentation on the real robot, when we had been hand-placing the robot at its starting positions (before we tried the pull-forward method), that the learning system had been able to learn to handle these "difficult" initial positionings. It was this direct observation of the robot as it learned that led us to investigate the possible interaction of the training sample distribution on the learning algorithm.

5.3 Experimental Methodology

We attempted to bring an algorithm that had been developed and tested under simulation into a different medium, namely the real world. Our simulated robot functioned under a particular set of assumptions which did not hold true after being transferred to a real robot. When designing a robotic control system it is very important to try to identify and isolate all the environmental factors that may affect the operation of the system. This is particularly important for learning systems as their learning performance may be dependent upon additional environmental factors.

The same kinds of issues have to be addressed when considering how to report one's own or use someone else's research. Have all of the specific hardware and environmental issues been properly reported and documented? Are there peculiarities within the original experimental setup which make that particular piece of research difficult to reproduce or to generalize from? These questions are not unique to robotics but are as important to our research as they are in any other discipline.

6 Conclusion

We have presented a case study of learning involving both a simulated and a real robot. We have described how we transferred the learning algorithm from the simulated robot to the real robot for the purposes of collecting data. In the process of doing so, a serious problem with the assumptions that were made was identified. These particular assumptions caused a severe distortion in the generation of good training sets for the robot. The discovery of this problem led us to design a novel experimental setup in the hopes of compensating for the unwanted environmental influences.

More importantly, the lessons learned from this experience can be applied to other robotics research. Good experimental design and execution cannot be ignored as subtle environmental factors can seriously alter the results of an experiment in a way that at first is imperceptible. Without proper identification of the factors that affect a system's performance, other researchers who may be interested in continuing the work or at least attempting to reproduce it may have serious difficulty in doing so. It is our hope that we can help improve the quality of the experiments and reported results from the robotics community as a whole.

References

[Asada et al., 1996] Asada, M., S. Noda, S. Tawaratsumida, and K. Hosada: 1996, 'Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning'. *Machine Learning* pp. 279–303.

[Barto et al., 1983] Barto, A. G., R. S. Sutton, and C. W. Anderson: 1983, 'Neuronlike adaptive elements that can solve difficult learning control problems'. *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-13**, 834–846.

[Benhrahim et al., 1992] Benhrahim, H., J. Doleac, J. Franklin, and O. Selfridge: 1992, 'Real-Time Learning: A Ball on a Beam'. In: *Proceedings of the International Joint Conference on Neural Networks*, Vol. 1. pp. 98–103.

[Cohen, 1995] Cohen, P. R.: 1995, *Emperical Methods for Artificial Intelligence.* Cambridge, MA: MIT Press.

[Colombetti et al., 1996] Colombetti, M., M. Dorigo, and G. Borghi: 1996, 'Behavior Analysis and Training: A Methodology for Behavior Engineering'. *IEEE Transactions on Systems, Man, and Cybernetics* **B, 26**(3), 365–380.

[Hougen, 1998] Hougen, D. F.: 1998, 'Connectionist Reinforcement Learning for Control of Robotic Systems'. Ph.D. thesis, University of Minnesota, Minneapolis, MN.

[Hougen et al., 1996] Hougen, D. F., J. Fischer, M. Gini, and J. Slagle: 1996, 'Fast Connectionist Learning for Trailer Backing using a Real Robot'. In: *Proceedings of the IEEE International Conference on Robotics and Automation.* pp. 1917–1922.

[Hougen et al., 1994] Hougen, D. F., J. Fischer, and D. Johnam: 1994, 'A neural network pole balancer that learns and operates on a real robot in real time'. In: *Proceedings of the MLC-COLT Workshop on Robot Learning.* pp. 73–80.

[Hougen et al., 1997a] Hougen, D. F., M. Gini, and J. Slagle: 1997a, 'Partitioning Input Space for Reinforcement Learning for Control'. In: *Proceedings of the IEEE International Conference on Neural Networks.* pp. 755–760.

[Hougen et al., 1997b] Hougen, D. F., M. Gini, and J. Slagle: 1997b, 'Rapid, Unsupervised Connectionist Learning for Backing a Robot with Two Trailers'. In: *Proceedings of the IEEE International Conference on Robotics and Automation.* pp. 2950–2955.

[Jani, 1993] Jani, Y.: 1993, 'Application of Fuzzy Logic-Neural Network Based Reinforcement Learning to Proximity and Docking Operations: Special Approach/Docking Testcase Results'. Technical Report NASA-CR-192294, Research Institute for Computing and Information Systems, University of Houston, Clear Lake, TX.

[Jervis and Fallside, 1992] Jervis, T. and F. Fallside: 1992, 'Pole balancing on a real rig using a reinforcement learning controller'. Technical Report CUED/F-INFENG/TR 115, Cambridge University Engineering Department, Cambridge, England.

[Kohonen, 1989] Kohonen, T. K.: 1989, *Self-organizing and associative memory.* Berlin: Springer-Verlag, 3rd edition.

[Kong and Kosko, 1992] Kong, S.-G. and B. Kosko: 1992, 'Adaptive Fuzzy Systems for Backing up a Truck-and-Trailer'. *IEEE Transactions on Neural Networks* **3**(2), 211–223.

[Koza, 1992] Koza, J. R.: 1992, 'A Genetic Approach to Finding a Controller to Back Up a Tractor-Trailer Truck'. In: *Automatic Control Conference.* pp. 2307–2311.

[Kretchmar and Anderson, 1997] Kretchmar, R. M. and C. W. Anderson: 1997, 'Comparison of CMACs and Radial Basis Functions for Local Function Approximators in Reinforcement Learning'. In: *Proceedings of the IEEE International Conference on Neural Networks.* Houston, TX, pp. 834–837.

[Lin and Kim, 1991] Lin, C.-S. and H. Kim: 1991, 'Use of CMAC Neural Networks in Reinforcement Self-Learning Control'. In: *Proceedings of the 1991 International Conference on Artificial Neural Networks.* Espoo, Finland, pp. 1285–1288.

[Mahadevan and Connell, 1992] Mahadevan, S. and J. Connell: 1992, 'Automatic Programming of Behavior-Based Robots Using Reinforcement Learning'. *Artificial Intelligence* **55**, 311–365.

[Martin, 1995] Martin, F. G.: 1995, 'The Mini Board Technical Reference'. MIT Media Laboratory, Cambridge, MA.

[Martin, 1998] Martin, F. G.: 1998, 'The Handy Board Technical Reference'. MIT Media Laboratory, Cambridge, MA.

[Matarić, 1997] Matarić, M. J.: 1997, 'Reinforcement Learning in the Multi-Robot Domain'. *Autonomous Robots* **4**(1), 73–83.

[Michie and Chambers, 1968] Michie, D. and R. Chambers: 1968, 'Boxes: an experiment in adaptive control'. In: E. Dale and D. Michie (eds.): *Machine Intelligence.* Edinburgh: Oliver and Boyd.

[Nguyen and Widrow, 1990] Nguyen, D. and B. Widrow: 1990, 'Neural networks for self-learning control systems'. *IEEE Control Systems Magazine* **10**(3), 18–23.

[Russell and Rees, 1975] Russell, D. W. and S. J. Rees: 1975, 'System Control — A Case Study of a Statistical Learning Automaton'. In: R. Trappl and F. de P. Hanika (eds.): *Progress in Cybernetics and Systems Research*, Vol. II. New York: John Wiley and Sons.

[Selfridge et al., 1985] Selfridge, O., R. Sutton, and A. Barto: 1985, 'Training and tracking in robotics'. In: *Proceedings of the International Joint Conference on Artificial Intelligence.* Los Angeles, CA, pp. 670–672.

[Shelton and Peterson, 1992] Shelton, R. O. and J. K. Peterson: 1992, 'Controlling a truck with an adaptive critic CMAC design'. *Simulation* **58**(5), 319–326.

[Woodcock et al., 1991] Woodcock, N., N. J. Hallam, and P. D.Picton: 1991, 'Fuzzy BOXES as an alternative to neural networks for difficult control problems'. *Artificial Intelligence in Engineering* pp. 903–919.