UNIVERSITY OF MINNESOTA

This is to certify that we have examined this copy of a doctoral thesis by

Dean Frederick Hougen

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Maria L. Gini James R. Slagle

Name of Faculty Advisors

Signature of Faculty Advisors

October 15, 1998

Date

GRADUATE SCHOOL

Connectionist Reinforcement Learning for Control of Robotic Systems

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Dean Frederick Hougen

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

October 1998

Acknowledgments

I owe a debt of gratitude to those who have made this thesis possible.

James Slagle for awakening my interest in neural networks and reinforcement learning, for acting as the instructor for the independent-study course in which the ideas in this thesis originally took shape, and for acting as an advisor throughout my graduate career.

Maria Gini for awakening my interest in intelligent robots and providing insight and expertise on this subject, for funding the development and construction of the mini-robots used, and for acting as an advisor throughout my graduate career.

Nikolaos Papanikolopoulos for supporting me financially through the final summer of my research and for moral support throughout my career.

Paul Rybski for devising the self-positioning algorithm and for designing, building, programming, and documenting the self-positioning mini-robot, SPTBMin, which was used to collect all of the real-world data in this thesis.

Rahul Singh for the enlightening conversations about connectionism and self-organization.

John Fischer for the idea of applying the learning system to the truck-backing problem and for designing, building, and programming PBMin and TBMin, the first mini-robots on which the learning system was tested.

Deva Johnam and Chris Smith for their contributions to the coding of PBMin and TBMin.

The other members, past and present, of the Artificial Intelligence, Robotics, and Vision Laboratory for making this an enjoyable place to live and learn.

DEDICATION

To my parents, Roy and Patricia Hougen.

For the things we have to learn before we can do them,

we learn by doing them.

— Aristotle

Abstract

Reinforcement learning for control may provide intelligent robotic agents with a means of improving their performance through interaction with the environment. Unfortunately, existing reinforcement-learning schemes require great computational power, a large memory, and/or thousands of training trials to achieve good performance. These resource requirements make implementation of reinforcement learning on real robotic systems impractical. This thesis presents a system for reinforcement learning for control that uses laterally connected artificial neural networks to improve the efficiency of a basic reinforcement-learning mechanism. This system uses a Kohonen-type Self-Organizing Map to efficiently partition the input space of continuous sensor data. It 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 system performance is demonstrated through extensive simulation results and implementation on a real mini-robot with low computing power and little memory. The system learns well despite noisy, low-grain sensory input and uncertain interactions between motor commands and effects in the world. The truck-backing task is used as an example of a difficult credit-assignment problem.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Reinforcement Learning for Robotic Control

Reinforcement learning methods provide great potential for intelligent robotic systems to improve their performance through "trial and error" interactions with their environments. The robot senses the state of the environment, responds by taking a course of action, and learns as it receives evaluations of the new environment state at which it arrives. Such a method requires a *critic* component to evaluate the results of actions, but does not require a *teacher* that knows *a priori* what the preferred system response would be in each circumstance. This makes reinforcement learning appropriate for application in domains in which the environment is available for exploration but no prior analysis has been conducted to determine the best course of action within this environment. This means that, unlike supervised learning methods which simply provide for generalization of known activity patterns, reinforcement learning methods are capable of novel discovery.

Unfortunately, reinforcement-learning methods run up against two obstacles when we try to apply them in the world of robotics. The first of these obstacles is the continuous nature of many robotic environments. The classical reinforcement model is one of discrete environment states; reinforcement-learning methods must be adapted in some way before they can be applied to continuous environments.

The second obstacle is many learning experiences, known as *trials*, that reinforcement-learning schemes require for performance to improve significantly. In simulated environments, or environments that are purely formal to begin with (such as games), it is possible to provide the number of trials required by conventional reinforcement-learning schemes. However, while simulation is widely and importantly used in robotics, no simulation can capture all aspects of the real world. For this reason it is generally felt that no system is proven until it is applied to a real robot. If a learning episode from initially random action to good performance, known as a *run*, takes thousands or tens of thousands of trials — as is typical of reinforcement learning schemes — application of such a method to a real robot is impractical.

1.2 An Integrated Approach

Rather than trying to surmount these obstacles separately, we take an integrated approach to both. To overcome the mismatch between the traditional reinforcement-learning model and the continuous nature of robotic environments we discretize the continuous input provided by the environment by having the system classify each input vector as belonging to one or another of a few distinct categories or classes. Our first step towards an integrated approach comes in how we determine the boundaries for these classes. To be consistent with our desire to use reinforcement learning where it is most appropriate — that is, without requiring a prior analysis of behaviors and circumstances — we have our system determine the class boundaries itself. This is done through a *self-organizing* process that extracts regularities from patterns of input data.

The partitioning of the continuous input space into a finite number of classes allows us to use mechanisms from traditional reinforcement learning in our approach. However, we still make use of the fact that the underlying input is continuous to reduce the number of trials needed for learning. This is our second step towards an integrated approach.

In traditional reinforcement-learning schemes, it is assumed that nothing is known about the relationships between the discrete environment states; all are taken to be independent and an action is learned for each state separately. In our approach, though, we know that the classes to which we have assigned the input have come from underlying continuous variables, and so we can determine which classes are "near" to one another in the continuous input space. Using this knowledge, and an assumption that often we will want to respond in similar ways in similar situations, we allow an experience which involves one class to aid our reinforcement learning system in determining responses to other classes that are "close by."

Of course, this assumption may not always hold. For this reason it is not strongly enforced. Rather, it is given sway early in the learning process but allowed to fade away as more experiences are gained by the system. Nonetheless, it provides a significant improvement in the overall learning speed of the method.

The third and final step in our integrated approach to the obstacles of continuous input and slow learning speed is the choice of method used to overcome them. For both the self-organizing process of input partitioning and for learning similar responses to similar input, we use laterally connected artificial neural networks. Different networks are used to solve each problem separately, but the overall working of the networks is similar. This parsimonious approach to both problems not only requires fewer mechanisms overall in the system but allows for changes to the entire system to be made more easily. For example, to reduce the memory requirements of the system, the networks for both input partitioning and response learning can be shrunk proportionally.

1.3 The Merits of the Parts

While the parsimony of the integrated approach gives the complete learning system a beautiful simplicity, this should not be taken to indicate that the parts of the system have no use outside of the whole as presented here. In fact, individual parts have proven worthy of study in their own right, bringing strong results to the problem areas they were constructed to address. The use of laterally-connected networks for learning similar responses to similar input has been shown to greatly improve the learning rate for cases in which a fixed input partitioning is given to the system [55, 58].

Likewise, having the system learn its own partitioning of the input through self-organization not only meshes well with reinforcement-learning domains but can improve the performance of the system over that of a traditional reinforcement-learning system given a fixed uniform partitioning (see 4.1.1).

This thesis, therefore, considers these parts and their individual contributions to reinforcement learning. But the whole is not just a sum of the parts. Taken as a whole, the system provides a single and consistent framework for overcoming two of the major obstacles to reinforcement learning for robot control. We have made a conscious effort to consider both obstacles at once, and have found that overcoming them together provides links from one that can be used as levers to make progress on the other. The result is a system that learns quickly and without requiring the implementors of the system to manually solve the classification problem. It is an organic whole that can be modified as such, because the part that solves one problem exhibits structures similar to those of the part that solves the other.

1.4 Application

As our application domain for the study of reinforcement learning in robotics, we have chosen to look at the problem of learning to back a truck and trailer rig to a goal. We are not concerned primarily with actually having trucks that can automatically back trailers to goals — rather we are interested in the study of reinforcement learning for robotic tasks.

We have chosen to study this particular problem for several reasons. First, this problem has the benefit of being simple to describe and understand. Many readers will already have experience in carrying out similar tasks themselves and, if not, the requirements of the task can nonetheless be easily imagined.

Second, the motions of a few rigid bodies simply joined is easy to model in simulation. This gives us a virtual test platform for algorithm development and result collection that can be used more readily than if we had to work solely on an actual physical robot.

Third, this task requires little in the way of strength or speed from the physical robotic test platform. Development and testing in simulation is all well and good, but we also want to be able to prove our system on a real robot, and this allows us to construct such an implementation platform at minimal cost.

Fourth, the backing motion of a truck and trailer rig is inherently unstable. If the spines of the truck and the trailer are not perfectly aligned, any backward motion of the truck will tend to cause the angle between the two bodies to increase. The steering of the truck must reflect this fact or a *jack-knife* condition will soon result, causing the robot to fail at its task. Since the robot is only allowed to move backwards during a trial, there is no way for the robot to recover from mistakes of this nature.

Fifth and finally, while controlling a truck and trailer rig as it moves backwards is not a difficult task, *learning* to control these movements certainly is. This is the most important criterion. If the task were too easy, then developing a system that learns to solve it quickly would be trivial.

To add to the difficulty of the learning task, we give our reinforcement-learning system exremely limited feedback. Rather than telling the learning system as it backs that it is doing well or poorly (or better or worse), we give it no judgments about its actions until it fails (e.g., jack-knifes, as mentioned above) or succeeds by reaching the goal. This means the robot has taken many different actions as it moved and before this feedback was given. If it fails, which actions were to blame? If it succeeds, were all of its actions correct or did some of its actions tend to send it off course and yet success occurred *despite* these bad moves? This is a central problem in reinforcement learning, known as the *credit assignment problem.*

1.5 Organization of the Thesis

This thesis is organized as follows:

Chapter 2 gives an overview of machine learning, focusing on reinforcement learning in particular, and an overview of connectionism, focusing on laterally-connected networks such as are used in our approach.

Chapter 3 presents the problem of applying reinforcement learning to robotic control tasks, describes why current reinforcement-learning methods cannot handle aspects of this task, and discusses the need for a solution to this problem.

Chapter 4 describes in detail the workings of our proposed learning system. The learning system is described incrementally from a traditional reinforcement learning system to the unified system that learns an input partitioning and rapidly learns appropriate responses. Some example results are shown to illustrate particular points.

Chapter 5 presents a comprehensive set of results that have been gathered, both in simulation and on the real robot. These results include tests of the baseline traditional reinforcement-learning system, intermediate systems embodying one or the other of our developments, and the full integrated system. For each of these systems, a range of test are conducted for systems with greater or fewer neural units, corresponding to different memory and processing requirements. Tests are made on versions of the truck-backing task with varying degrees of difficulty.

Chapter 6 discusses the results, showing the contributions of the individual parts of the system and how the system functions as a whole. The appropriateness of the application of the system or its parts under various circumstances is also discussed.

Chapter 7 discusses the contributions of this research, both for robotics and for machine learning. It presents conclusions and ideas for future research.

Appendix A gives details of the simulation model used, including all equations.

Appendix B presents the robotic platform used for carrying out the tests in the real world.

Chapter 2

REINFORCEMENT LEARNING AND SELF-ORGANIZATION

This thesis brings together two formerly disparate areas from the field of machine learning, *reinforcement learning* and *self-organization*. To understand these areas, it is helpful to take a brief look at the overall picture of the field. This is followed by discussions of the areas themselves, as they are relevant to this thesis. These include prior research on reinforcement learning for control and the use of laterally-connected networks for self-organization.

2.1 Overview of Machine Learning

Learning can be defined as *any process by which a system improves its performance* [25]. Machine learning, then, involves instantiating such a process in a machine. Given input I from an environment, a learning system should learn to produce progressively better output O, according to some measure.

2.1.1 Static versus Dynamic Environments

Machine-learning problems can be usefully divided into classes based on the way that the learning system interacts with its environment. If the flow is unidirectional — information flows from the environment to the learning system — then we say that the environment is *static*, at least from the perspective of the learning system. This is not to say that the environment is necessarily *stationary* from a broader perspective (it may, for example, change with time) but only that the output of the learning system does not directly affect the state of the environment from which the input originates. (See Figure 2.1.) An example of this class of machine-learning problem is learning to recognize (classify) printed characters.

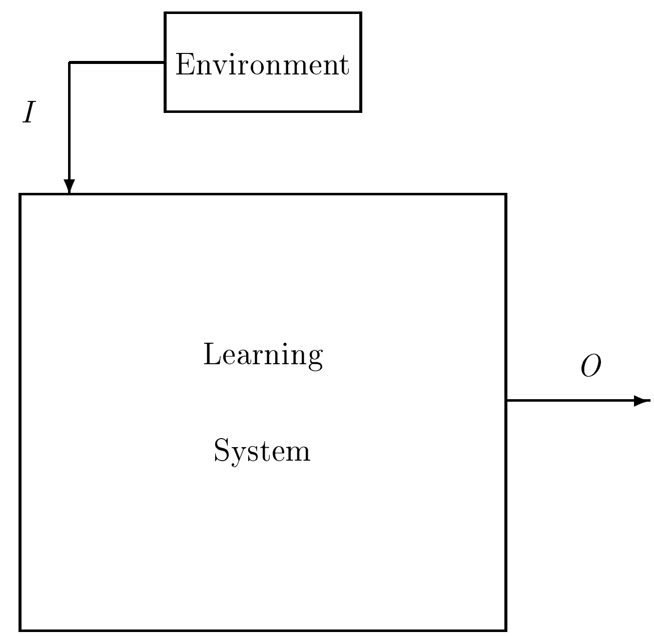


Figure 2.1: Learning in a static environment.

If the flow is bidirectional — information from the environment flows to the learning system and output or *responses* from the learning system are *actions* that affect the environment — then we say that the environment is *dynamic*. (See Figure 2.2.) An example of this class of problem is learning to control a robot.

(The third possibility, unidirectional flow into the environment, does not permit learning.)

2.1.2 The Role of Supervision in Learning

A second division in machine-learning problems comes in terms of the level of supervision provided during the learning process. A learning system may be decomposed into a learning/acting element or *agent* and a supervisory element. If a desired output

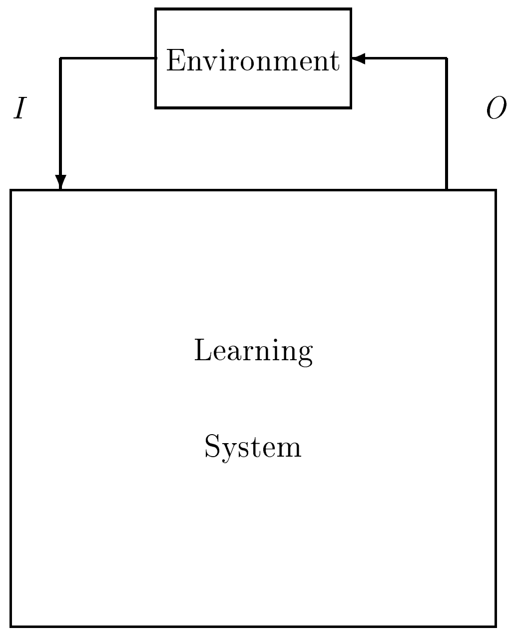


Figure 2.2: Learning in a dynamic environment.

is known for each input provided by the environment, this can be given to the supervisory element, which then acts as a *teacher* for the learning element. The teacher receives the same input pattern as the agent and provides the desired output as a training signal T to the agent. (See Figure 2.3.) This is known as *supervised learning* or *learning with a teacher*.

In the character recognition task mentioned above, for example, one might have a set of character images and desired classifications. The environment would be the mechanism that presents the images, while the teacher would simply relay the desired classification for each. The usual use of supervised learning is to replace a costly or slow method for accomplishing some task with a less expensive or faster agent capable of performing the same task (e.g. replacing a person reading addresses with a machine) or to produce generalization capabilities not found in the original teacher.

For supervised learning in dynamic environments, the desired output as provided

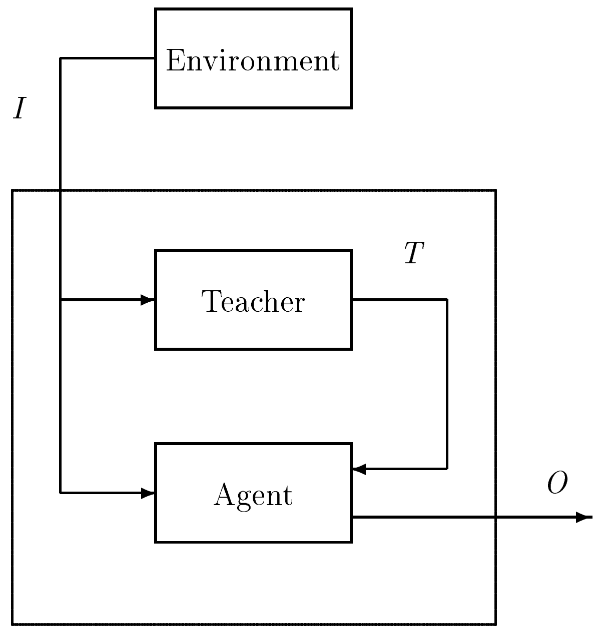


Figure 2.3: Learning with a teacher in a static environment.

by the teacher is typically used as the action taken in the environment. (See Figure 2.4.) For example, to teach a robot to navigate through some environment, the teacher would use the current input from the environment to decide what movement to make. This decision would be fed to the learning element as the desired movement given the current input, and also as the action taken to bring the robot to a new location, so that navigation learning may proceed from there.

In many cases, however, a desired output is not known. In fact, this is often the reason that machine learning is desired — to learn some function not already known. For a static environment, so-called unsupervised-learning methods have been developed. As Barto [11] points out, however, these methods are not truly unsupervised. Instead, they attempt to extract regularities or statistical properties from the input by recoding it according to a built-in coding principle, such as principal component analysis [11, 156]. This is akin to having a principle-specific teacher, rather than an environment-specific teacher. For this reason, we will prefer the term *self-*

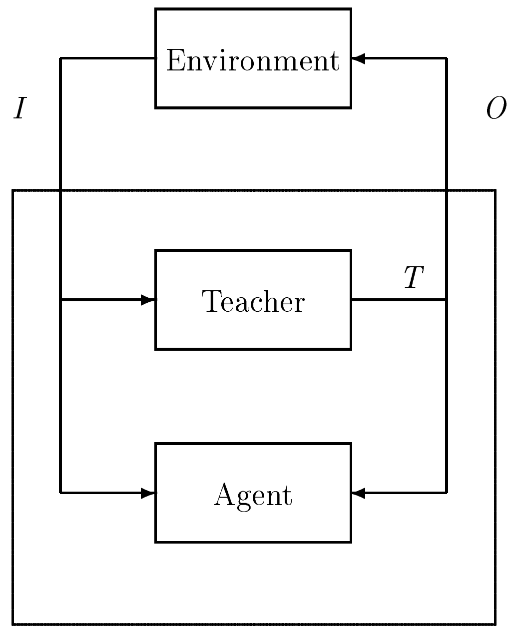


Figure 2.4: Learning with a teacher in a dynamic environment.

organizing system, rather than unsupervised-learning system. The principle-specific teacher takes the current function f used by the agent to recode the input, as well as the current input and output vectors, and determines a new coding function $\hat{f}$. (See Figure 2.5.) This produces a much tighter coupling between agent and teacher than in the case of the environment-specific teacher of supervised learning.

A recent example of this type of learning is the use of a self-organizing map (SOM, see Section 2.3), to classify over one million documents (Usenet newsgroup postings) to produce a content-addressable memory for the database [72]. There was no desired classification for any particular document; the goal was to group similar documents together.

For the dynamic environment, however, it is necessary for the agent to do more than simply recode the input. The response given by the agent must be an *appropriate* action to take, given the input from the environment. This notion of appropriateness means that there must be some method to distinguish between "good" and "bad"

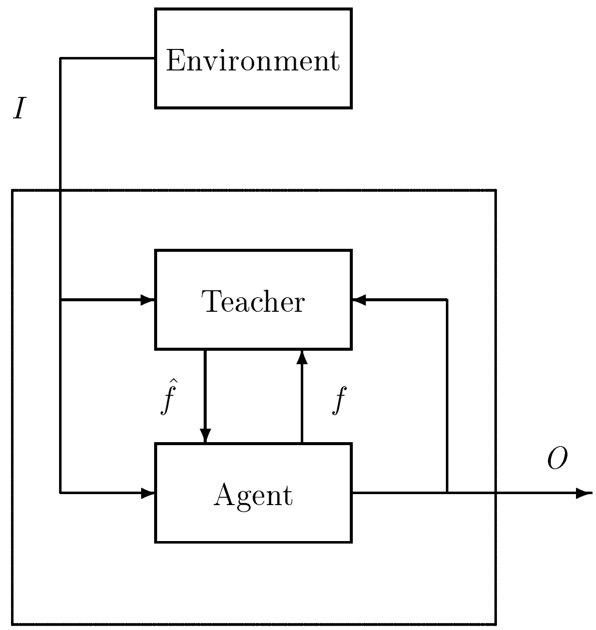


Figure 2.5: Learning by self-organization in a static environment.

actions. If a teacher is not available to provide the desired control signal, there must at least be a *critic* present that can provide a qualitative judgment of performance. In general, this judgment comes from evaluation of subsequent input from the environment, although a critic may also base its judgments on agent responses. Because the critic does not know the desired action, the response of the agent is typically used as the action to be taken in the environment, even during learning. (See Figure 2.6.) This method of learning has been called *bootstrap adaptation* [159] and *learning with a critic*, although the most common term for it is *reinforcement learning*. The critic's evaluation, then, is termed the *reinforcement signal* r.

A classic example of this type of learning is learning to balance a pole hinged to a cart that is running on a track of finite length. In this problem, known as the inverted-pendulum problem, the agent is provided with input approximating the state of the cart-pole system and must learn to apply forces to the cart in such a way that the pole does not fall over and the cart does not hit either end of the track. Normally,

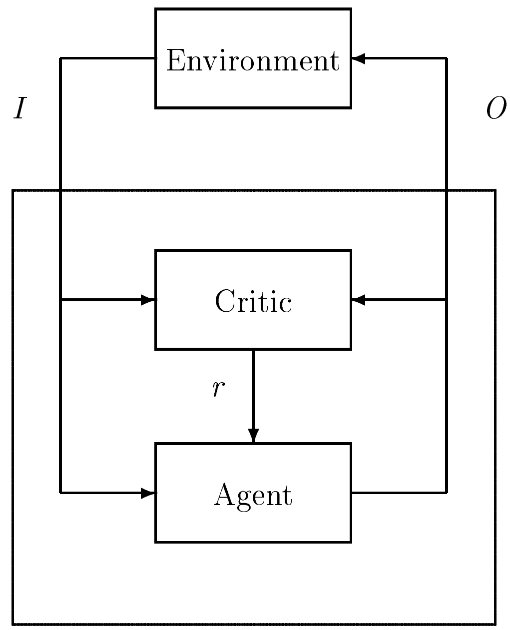


Figure 2.6: Reinforcement learning in a dynamic environment.

a simple critic is provided that signals failure when either of these conditions occurs.

In general, supervised-learning methods allow agents to learn more quickly, since a desired output is always provided, and may be preferred if they are available. However, self-organizing and reinforcement methods are applicable to a wider range of problems, since they do not require a teacher with specific knowledge of the environment. Self-organizing systems can discover patterns to data not known *a priori*, the teacher needs only to embody a coding principle. Reinforcement-learning systems can arrive at good action policies through trial and error, they only require a critic that can evaluate the results of an action — it need not know what particular action would be preferred.

This breakdown of machine-learning types should not be seen as putting limits on the components of particular learning systems, since hybrid systems are certainly possible. It has been shown that it is possible to combine self-organizing methods with supervised methods, for instance. The Counterpropagation Networks of Hecht-

Nielsen have a self-organizing layer that extracts regularities from the input permitting classification and a supervised layer that learns associations between classes and desired output [51]. Likewise, it will be shown in this thesis that self-organizing methods can be combined with reinforcement learning.

2.1.3 The Quality of Feedback

The distinction between learning with a teacher and learning with a critic can be thought of as a distinction between grades of feedback [159]. The feedback from a teacher is specific; it tells the agent what the answer should be. The feedback from a critic is qualitative; it tells the agent how good the agent's own answer is.

Another consideration in learning is the immediacy of feedback from the supervisory element, either teacher or critic. In many machine learning systems it is assumed that feedback is given for each input-output pair and that the pairs are independent. These assumptions are often false in real-world situations, particularly in control problems. Instead, a long sequence of control actions may be taken before feedback is given. In the inverted-pendulum problem, for example, the usual formulation is for the critic to only provide feedback (signal failure) when the pole falls or the cart hits the end of the track. Before that time, the agent will have given many control signals (perhaps thousands [12]) and the cart-pole rig may have been moving around for an extended period of time. This feedback delay produces the *temporal credit assignment problem*. In supervised cases, this problem may be handled by Backpropagation Through Time [99, 100, 101, 160, 162]. In reinforcement learning, however, this remains the "biggest problem facing a reinforcement-learning agent" [65].

For the temporal credit assignment problem in reinforcement learning, two basic mechanisms have been proposed [136]. These are (1) to have the agent learn an internal reward estimation that is more immediate than the external reward signal provided by the critic, and (2) to have the agent use *eligibility traces* to determine the degree to which to reward or punish particular state-action pairs.

The first mechanism essentially places an adaptive critic element within the agent that learns, by trial and error, which regions of the input space are likely to lead to reward and which to punishment. This internal critic then provides heuristic reward or punishment signals of its own, depending on which region of the input space an action leads to. A basic form of this mechanism is known as the Adaptive Heuristic Critic [65].

The second mechanism, known as the eligibility trace, makes use of the credit assignment heuristics of recency and frequency [136]. The recency heuristic says that the more recently an event occurred prior to reward or punishment, the more deserving it is likely to be of this feedback. The frequency heuristic says the same thing with regard to how frequently an event occurred prior to feedback. Both heuristics are used in accumulating traces but only the former is used in the replacing trace [136]. (See Subsection 4.1.3.)

The mechanisms of internal reward estimation and eligibility traces are combined in the popular temporal difference methods of TD(λ) and Q-Learning [65]. The method used in this thesis only makes use of the mechanism of the eligibility trace.

Finally, within reinforcement signals themselves there are levels of quality. Some critics may provide signals that are simple failure or success indications (e.g. 0 and 1) while others may provide more gradations. In general, construction of more complex critics requires more detailed knowledge of the environment in which it will be used, but may provide for more rapid learning.

2.2 Reinforcement Learning for Control

The term reinforcement learning, first used for this branch of machine learning by Minsky [93] was borrowed from psychology [67]. Its theoretical basis, however, is to be found in statistics and dynamic programming [18]. Research in reinforcement learning has been conducted for nearly forty years [128] but there has been great interest in the field recently [37, 65, 36, 83]. For a survey of the current state of the art, see Kaelbling, Littman, and Moore [65].

The "standard reinforcement-learning model" consists of discrete sets of environment states and agent actions and a scalar reinforcement signal [65]. This model has lent itself well to game-learning. Samuel's famous checkers playing system contains an early application of reinforcement learning, although many other components were included in this system as well [128, 129]. One of the earliest systems to learn soley by reinforcement was developed by Michie and Chambers [90]. Known as GLEE, the Game Learning Expectimaxing Engine learned to play "Naughts and Crosses" (tic-tac-toe) by trial and error. Another early system, this one by Widrow, Gupta, and Maitra, learned by reinforcement to play the card game blackjack, although these authors credit James S. Koford with being "the first to make bootstrap learning work" [159]. One of the most successful reinforcement-learning systems was developed more recently to play the game of backgammon [65]. It has competed strongly at the top levels of tournament competition.

While game-playing could be thought of as a type of control problem, traditionally control has been concerned with physical systems, such as manufacturing equipment, robots, or chemical processing facilities. In the terminology of control theory, all of these systems are known as *plants*. Plants do not fit as nicely into the standard reinforcement-learning model as do games. One major difficulty is that rather than discrete sets of environment states, as are found in games and called for in the standard model, physical systems are characterized by continuity, at least at the macroscopic level at which control is to take place. Another major difficulty is that the trial and error method of reinforcement-learning schemes can take thousands of trials and thousands of errors before successful behavior is learned. While game rules are purely formal and can be completely internalized within the computer for rapid play — allowing for sufficient training trials — plants exist outside the computer. For this reason, most reinforcement-learning systems have been applied soley to computer simulations of real physical systems. Nonetheless, there has been a long and ongoing interest in finding ways to adapt the reinforcement-learning model to control.

Probably the original reinforcement control-learning problem, already mentioned above, is the inverted-pendulum problem. This problem is also a classic problem in traditional control theory [130, 23, 103, 13, 104] and *fuzzy* control [31] and has been studied extensively in supervised control-learning as well [161, 158, 47, 46, 148, 108, 155, 150, 163, 113, 48, 49, 105]. The first attempt at reinforcement learning for this problem appears to have been done by E. C. Fraser, although few details of the learning scheme are given other than the critic was a person who would evaluate "long chains of decisions" as acceptable, unacceptable, or uncertain by watching the system control the movements of an actual cart-pole rig [161]. The first reinforcement control-learning system to be described in any detail was Michie and Chamber's BOXES system [90, 91]. Utilizing the same learning method they applied to tic-tac-toe, Michie and Chambers created a system to learn to balance a pole in simulation. Since then, many other authors have explored reinforcement-learning methods for this problem [124, 12, 132, 2, 28, 117, 116, 127, 3, 4, 78, 60, 29, 33, 80, 79, 84, 164, 63, 113, 118, 42, 123, 38, 97]. Evaluations of the inverted-pendulum problem as a benchmark machine-learning problem have also been published [41, 44, 43].

Another control problem that has received much attention is the truck-backing problem. This is the problem of backing a non-holonomic vehicle (the truck) with zero or more trailers to some target or goal position. In most versions of the problem, the only steering comes from the front wheels of the truck and the only power comes from the truck's rear wheels. This problem has received extensive recent attention in traditional control theory [9, 146, 10, 77, 98, 137, 154, 140, 147, 20, 131]. It has also been investigated from the perspectives of fuzzy control [141, 142, 143] and supervised learning [99, 100, 101, 160, 73, 162] and it has been evaluated as a machine-learning benchmark [45]. The exploration of this problem as a reinforcement-learning problem has been less extensive but present nonetheless [164, 74].

Many other problems have also been investigated as reinforcement control-learning problems, including inserting a peg in a hole [135], controlling a trash to steam plant [121], balancing a ball on a beam [19], pushing boxes [81], docking the space shuttle with a satellite [61], kicking a ball into a goal [6], vision-based grasping [126], and pushing an object up a steep hill [96, 75]

2.2.1 Input Partitioning

To handle the discrepancy between continuous input from the plant and the use of discrete environment states in the standard reinforcement-learning model, two basic approaches have been used. The first is to discretize the input and use a standard model. The second is to modify the model to accept continuous input.

Discretizing Continuous Input

BOXES, the original Michie and Chambers system for reinforcement learning for the inverted-pendulum problem, used the first method. To discretize the four-dimensional input space (cart position, cart velocity, pole angle, and pole angular velocity), these authors partitioned the space into four-dimensional hyper-rectangles (225 in one study [90] and 162 in another [91]). All input that fell within a given partition region was treated as identical by the reinforcement-learning agent.

The choice of quantization values delineating the regions was made by the authors to reflect the task to be learned. The partition regions were chosen to be small near the "center" of the input space (pole vertical, cart centered on track, velocities zero) and larger away from this point. This reflected the authors' belief that fine control was needed in the central regions but more approximate control would suffice towards the outside.

This method has subsequently been followed by other authors working on this problem [12, 132, 127, 63, 42] and other problems using reinforcement learning for control [19, 81, 6]. A minor variation on this approach has been to allow for fuzzy decision boundaries between the regions [78, 61].

The simplest alternative method, and one that requires less intimate knowledge of the task to be learned, is simply to use a fine uniform partitioning [122].

Finally, there has been recent interest in systems that can learn their own partitionings [135, 113, 118, 38, 96, 126, 75]. Learned partitionings have also been considered for finely-grained environments in non-control areas [24, 96]. Some learned-partitioning procedures have worked by successively dividing existing partitions regions, based on some criteria [135, 24, 96, 126] while others use a fixed number of partitioning regions but adjust the borders between them [113, 118, 38, 75]. The system presented in this thesis uses a border-adjusting procedure.

Accepting Continuous Input

The other approach to dealing with the continuous nature of plants is to create a reinforcement-learning scheme that can use continuous input directly. This has been accomplished through the use of multi-layer feed-forward neural networks [2, 3, 4, 60], genetic algorithms [33, 84, 74, 97], more traditional gradient search methods [29], and other methods [28].

2.2.2 Rapid Response Learning

One widely heard complaint about reinforcement learning is that it takes too long to achieve good results. This criticism is especially relevant if one wishes to have a system that learns on an actual, rather than a simulated, plant. A primary reason for these long learning times comes from learning responses individually for each of the discrete environment states found in the standard reinforcement-learning model. This is how traditional reinforcement-learning algorithms, such as TD(λ) and Q-Learning, have worked. This allows these algorithms to be general; they make no assumptions about the characteristics of the input codings they receive. However, if the state space is large, the *structural credit assignment problem* — learning actions for states that may be encountered rarely — becomes paramount.

Fortunately, in some cases we may be able to make some assumptions that allow for more rapid learning. If we can find relevant similarities between environment states, this should allow us to treat them the same. It is not unreasonable, when input codings are simple partitionings of continuous state spaces found in physical plants, for example, to guess that nearby partition regions require similar responses for control. We should not enforce this assumption too strongly, as it may be necessary for nearby regions to have highly dissimilar responses for good performance, but some degree of generalization across states may "jump start" the learning process.

Generalization schemes have generally borrowed from methods used previously in supervised learning, including use of the Cerebellar Model Articulation Controller (CMAC) [80, 79, 75], use of a neighborhood relation [113], and use of Radial Basis Functions or variants [75]. Some original generalization methods have also been suggested, such as the use of the Hamming distance between bit encodings of binary input data [81] and statistical clustering of state-action pairs [81].

2.3 Self-Organization and Laterally-Connected Networks

The use of the term self-organization has not always been consistent (cf. [50] and [70]). In this thesis the term is defined to correspond with what we believe to be currently the most widely accepted use: as a general class of unsupervised pattern-recognition or classification-learning methods (see Subsection 2.1.2). Self-organization can be accomplished by many procedures, such as Classification and Regression Trees (CART) [21], ID-3 [110], or C4.5 [111]. In connectionist research, self-organization is most often accomplished through the use of laterally-connected networks.

2.3.1 Maximum-Likelihood Estimator

One simple use of lateral connections is to form a "winner-take-all" network for use as a maximum-likelihood estimator [106]. In such a network, each individual neural unit has an encoding for a class exemplar, such that when input is presented to it, it becomes more or less active based on some measure of similarity between the input and the exemplar. Units further have lateral connections to each other such that when a unit becomes active it tends to inhibit the activation of other units to which it is connected. Finally, each unit has a circular excitatory connection to itself. If the network is fully connected, all excitation connections are of equal strength, all inhibition connections have equal strength that is less than e/n where e is the excitation strength of the circular connections and n is the number of units in the network, and the input is presented to all units at once, then the output of the network will converge to a state in which only a single unit is active. That unit will be the one with the exemplar most similar to the input and will be considered the "winner" — the input will be classified as belonging to the class represented by that unit's exemplar. This competition between units gives a biologically-plausible mechanism for a maximum-likelihood estimator. The functioning of this network is similar to the "on-center, off-surround" phenomenon found in some biological networks [39].

2.3.2 Cooperative Learning

A second simple use of lateral connections is to allow for cooperation during learning. If a unit undergoes some weight updates due to learning, this information may be shared with other units that then undergo similar updates. If the updates of all weights were uniform across the network, this would accomplish little, if anything. If, instead, a unit more strongly influences the weight updates of some units than others, then more interesting results are possible. For example, defining a distance function over the units in the network, such that some units are "closer" to one another than others gives us a structure that we may overlay with an interesting weight-update rule. Under this rule, units are updated to in relation to their lateral distance from the unit originally undergoing weight updates from its own learning experience. One commonly seen rule for this update is a “Sombrero function” (formed as a sum of Gaussians) as shown in Figure 2.7, although variations are certainly used as well [1]. Using this rule, units close to one another will tend to develop similar weights, while those more distant will tend to develop different weights. This provides a cooperative local generalization tendency in the network.

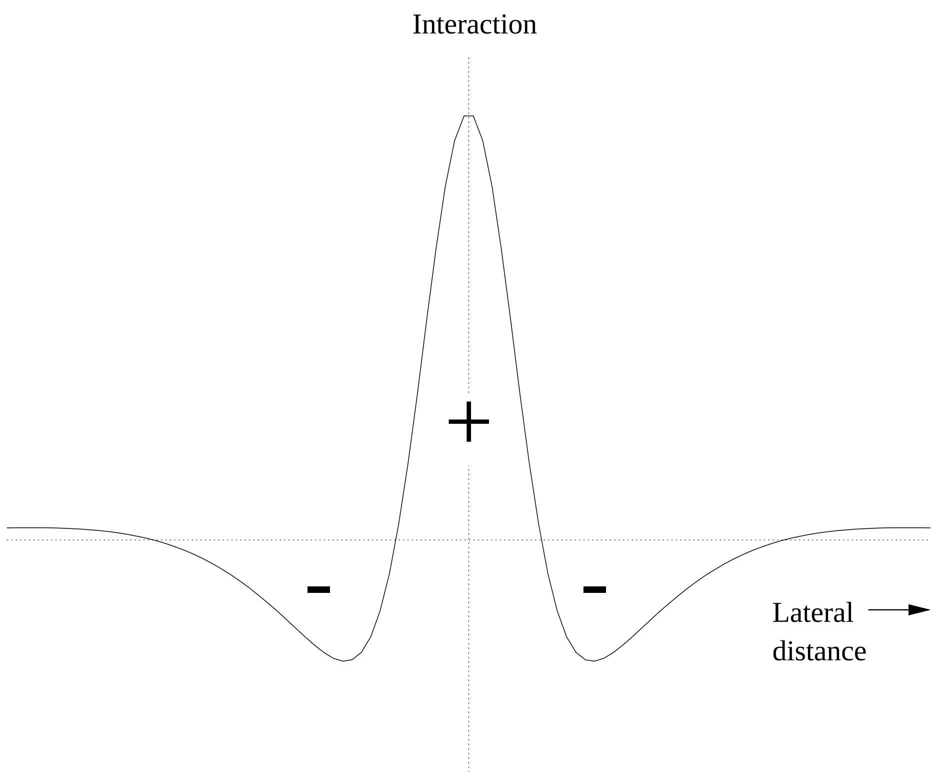


Figure 2.7: Lateral interaction (after Kohonen).

2.3.3 Self-Organizing Maps

These two uses of lateral connections are most often combined into a single learning system, as in the Self-Organizing Map (SOM) developed by Kohonen [69, 70, 71]. In this system, the network is defined in terms of a simple topology, such as planar. (In a planar network, the units are given Cartesian grid coordinates and the distance function used is the Euclidean distance between coordinates.) When an input vector is presented to the network, a three step learning process occurs. First, a competitive winner-take-all procedure determines the unit with a weight vector most similar to the input. Second, the winning unit updates its weight vector to more closely approximate the given input vector. Third, the units surrounding the winning unit update their weight vectors towards or away from the input vector, depending on their lateral distance from the winning unit. In this way, the weight vectors of the units of the network come to approximate the distribution of the input vectors while tending to apply a topological ordering to it.

In this thesis, this type of competitive and cooperative self-organization is used for input space partitioning, and a second cooperative-learning phase is used to aid in generalization for reinforcement control-learning. This is similar to a system previously used for supervised control-learning [87, 112, 113, 115, 88].

Chapter 3

THE PROBLEM: AUTONOMOUS ROBOT LEARNING

Robots have become well established in industry and forecasts predict a continued rapid increase in industrial robotics at least through the year 2000 [119]. The purpose of these robots is to increase economic competitiveness through improvements in workplace efficiency. These robots accomplish this by performing repetitive tasks in highly controlled environments. For these tasks, each robot is given the sequence (and often timing) of all movements that the robot should make. The robots then automatically follow these action sequences with, perhaps, some variations based on feedback from the environment [7].

One of the primary goals for robotics research today, however, is to create robots that can function outside of such highly controlled environments [30, 14, 15]. This is the field of *intelligent robotics*.

3.1 The Need for Autonomous Learning

Much early research on intelligent robotics focused on creating robots that could "reason" about simple situations involving novel arrangements of known objects in an environment. This could be done, for example, through the processing of predicate calculus formulae [102]. Once a plan is created it can be carried out by performing a sequence of basic robot actions. One difficulty with this approach is that it requires the programmer to provide the system with a rich collection of both basic actions and appropriate formulae if the robot is to be used in a complex environment.

In a second approach, some researchers have worked instead to give robots a basic set of "low-level" behaviors such as wandering or object avoidance that require much less detailed knowledge of the environment in order to be carried out [22]. The plan of this research is to add higher level behaviors until the robot is capable of achieving complex goals. Unfortunately, the design of these higher level behavior modules has proven to be difficult [81].

The difficulties in these approaches, and the concern that increasing complexity in robotic systems will only exacerbate these difficulties, has lead to research into robot learning as a replacement for directly programming all parts of these systems [81, 26, 64].

Other needs for learning in intelligent robotics include the need to program systems for which a model is not tractable or not available, the need to have robots act in environments that are not known sufficiently at the time the robot is programmed, and the need to have robots act in changing environments without constant human supervision [17, 26]. Further, research on learning in robotics can provide insights into other aspects of robotics (such as sensing) and insights into artificial intelligence in general, and intelligent embedded systems (or intelligent situated agents) in particular [26, 52].

For some of these needs, various supervised learning methods are appropriate [17, 26]. For others, such as the need to have robots act in changing environments without constant human supervision, supervisory techniques are antithetical.

To fully develop intelligent robots, then, requires much more than automation of a process, it requires *autonomy* of the robotic agent. This is autonomy in the sense of Russell and Norvig. "A system is autonomous to the extent that its behavior is determined by its own experience" [125]. This is a stronger sense than that in which autonomy means merely "not directly controlled by an operator." That is, to be autonomous robots need to be capable, not only of independent action, but of independent learning.

3.2 Autonomous Learning in Real Robots

Reinforcement learning seems like the obvious solution to the problem of autonomous control learning in robotic systems. Unlike supervised methods (in which a teacher is required that knows the correct action to take in each situation), all that reinforcement learning methods require is a way to measure the degree to which an action is successful. If the evaluation mechanism — the critic element — can be provided to the robot, the robot can learn from its own experiences. In fact, the idea of autonomous robot learning has been used to provide the motivation for the study of reinforcement learning. In Mitchell's text on machine learning [94], the chapter on reinforcement learning begins:

> Consider building a learning robot. The robot, or *agent*, has a set of sensors to observe the *state* of its environment, and a set of *actions* it can perform to alter this state. For example, a mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn." Its task is to learn a control strategy, or *policy*, for choosing actions that achieve its goals. For example, the robot may have a goal of docking onto its battery charger whenever its battery level is low.

Unfortunately, there are difficulties in actually applying reinforcement learning to robots.

3.3 Problem 1: Continuous Input

The first problem in actually applying reinforcement learning to robotic systems is the discrepancy between the continuous input data often generated by robot sensors and the discrete environment states in the standard reinforcement-learning model. Two basic approaches have been used to overcome this problem. One is to discretize the input and use a standard model; the other is to modify the model to accept continuous input.

3.3.1 Discretization and the Standard Model

The original method used to discretize the input space was to simply have the system designer partition the regions according to his or her understanding of the task to be learned. The difficulty with this approach is that it requires the designer to know a great deal about the potential interactions between the robot and its environment. This is both a practical and a theoretical problem. From a theoretical standpoint, reinforcement learning is most appropriate when these details are poorly known. If the relevant environment parameters are measurable and the interaction dynamics are well understood, then it may well be possible to design a teacher rather than just a critic, or to directly code the control procedure without the need for a learning system. From a practical standpoint, it may be difficult to find a workable variable-resolution partitioning even if complete knowledge of the environment is available, since there is no algorithm for carrying out this task [63].

The alternative of a uniform fine partitioning has the shortcoming that in many cases it will involve many unnecessary partitions resulting in slower learning within what could otherwise be a single control region. That is, finely partitioned systems are more subject to the structural credit assignment problem. This only serves to exacerbate the problem of learning time in real robots (see Section 3.4).

Recent investigations into systems that can learn their own partitionings provide methods by which more autonomous learning is done by the robot. Besides reducing the need for over-compensation in the number of partition regions, this should save the designer from needing to understand the task as completely prior to deployment of the robot. Unfortunately, however, the successive-dividing procedures have relied on assumptions about the problem definition and run the risk of exhausting available memory by excessively partitioning unimportant regions of the input space if parameters are not tuned properly [135, 24, 96, 126].

Procedures that learn partitioning regions by adjusting their borders, then, seem to be the best choice from the discretization methods. Like successive-dividing methods, however, border-adjusting methods do increase learning time for the system, and add to the computational burden on the system, when compared to those in which the region borders are assigned by the system designer. This is to be expected — as more of the burden is lifted from the system designer, more of the burden must be placed on the learning system. The indexed-partitioning method developed in this thesis (see Section 4.2.3) has advantages over previous border-adjusting methods [113, 118, 38] in terms of learning time and computational power requirements.

3.3.2 Continuous Input with a Non-standard Model

The more radical approach to dealing with continuous input in reinforcement learning is to add to or modify the standard reinforcement-learning model so that the system can directly accept continuous input. While the continuous input methods should be more generally applicable than discrete partitioning methods, they have suffered from several difficulties, including significantly longer learning times [2, 3, 4, 60, 33, 84, 74, 97], poorer performance [29, 84], much greater computational complexity [28], and/or a need to give the system task-specific assistance [28].

3.4 Problem 2: Long Learning Times

The second problem with applying reinforcement learning to robotic systems is the large number of trials typically required for the system to acquire competent behavior. This is certainly among the reasons that so many studies of reinforcement learning for control have used simulations exclusively (e.g., [90, 91, 124, 12, 132, 2, 28, 117, 116, 127, 3, 4, 78, 60, 29, 33, 80, 79, 84, 121, 164, 74, 113, 118, 42, 61, 123, 38, 97, 75]).

Since it is the learning that takes so much time, it is tempting to have the system learn in simulation and apply the resulting policy to a physical system. This approach may work well for systems in which the simulation closely approximates the relevant aspects of the real world, although degradation of performance may be observed [6]. However, if the simulation is not sufficiently accurate, policies that work well in simulation will not be appropriate for the robot to follow in the real world [81]. In this case, there are two options. The first is to use a more accurate simulation. The second is to have the learning carried out on the real robot.

In cases where sufficient relevant information about the potential interactions of the robot and its environment are known it may be possible to construct a more accurate simulation. However, as with partitioning the input space, the job of creating a more accurate simulation has both theoretical and practical difficulties. Again, reinforcement learning is most appropriate when such information is at a minimum. So, from a theoretical standpoint, if enough is known to construct a more accurate simulation, then perhaps supervised learning would be more appropriate or perhaps the control policy could be determined directly by the system designer without the need for robot learning at all. From a practical standpoint, there is no algorithm for determining what information is relevant to a simulation environment for a reinforcement-learning agent.

This leaves us with the option of having the learning carried out on a real robot and returns us to the problem of long learning times required for reinforcement learning. Addressing the structural credit assignment problem would go a long way to reducing these long learning times. The input generalization schemes — based on the fact that the underlying continuous input space has been partitioned to adapt it to the reinforcement learning model — are attempts to address this problem.

The use of the Cerebellar Model Articulation Controller (CMAC) has proven effective in some cases, but rests on the additional assumption that the set of regions that are critical to task completion are a small subset of the total set of partition regions [80, 79].

The use of Radial Basis Functions and their variants may result in better performance on some learning tasks than using CMAC. The primary difficulty with this approach appears to be in adapting it to allow for the system to learn appropriate centers and widths to use for the functions on a given task [75].

The use of neighborhood relations on output regions has proven effective [113], although this system was limited to learning only when provided with immediate feedback.

Other input generalization methods are too specific to a particular task to be readily used in other areas [81].

The mechanisms we give for cooperative response learning (presented in Section 4.3) are neighborhood-based and therefore most similar to those of Ritter et al [113]. However, our system is capable of learning in the more general reinforcement-learning case that allows for delayed feedback.

3.5 A Connectionist Approach

In this thesis we address the need for autonomous learning in real robotic systems. We provide solutions to the two primary problems of using reinforcement learning in this domain — continuous input spaces and long learning times. We present a system for learning that uses a self-organizing process to learn a reasonable partitioning of the input space and a reinforcement-based scheme with cooperative response learning to more rapidly arrive at an appropriate action policy. The mechanisms given may be used independently yet together form an integrated whole. Both of these mechanisms are implemented using a connectionist approach that brings unity to the system, allowing for ease of modification of the entire system.

Chapter 4

THE LEARNING SYSTEM

The proposed learning system is presented in four parts. First, we describe a basic reinforcement-learning scheme that uses a discrete partitioning of the input space. This system uses eligibility traces as the mechanism for dealing with the temporal credit assignment problem. Results for this system applied to the truck-backing problem are presented as a baseline so that improvements to the system performance from input partition learning and cooperative response learning, both individually and in conjunction, can be demonstrated. Second, we introduce our system for learning input partitionings. This system has two alternate configurations — one more general, the other more efficient — and each is presented in turn. Third, the method of cooperative response learning is presented. Fourth, the unified system is presented as a whole.

In order to accommodate the limited memory and computing power of the mini-robot on which this system is implemented, all learning-system calculations are carried out using integer values. This fact should be kept in mind when viewing the equations given below. Further details of algorithm implementation are also given within the text below.

4.1 The Basic Learning System

The learning system consists of two primary components. Each component can be thought of as a network of neural units. The first component is the input network and the second is the output network. Through competition, the units of the input network classify each input vector. This classification is then used to select the responding unit in the output network. Through reinforcement learning, the units of the output network learn appropriate control responses. In this way, the system can be seen as forming a mapping from input space, via the input and output networks, to responses in the output space of the system (see Figure 4.1). In the basic learning system, the classification regions of the input network and the selection of responding units from the output network are hard-coded; all learning in the basic system takes place in determining the responses of the output network.

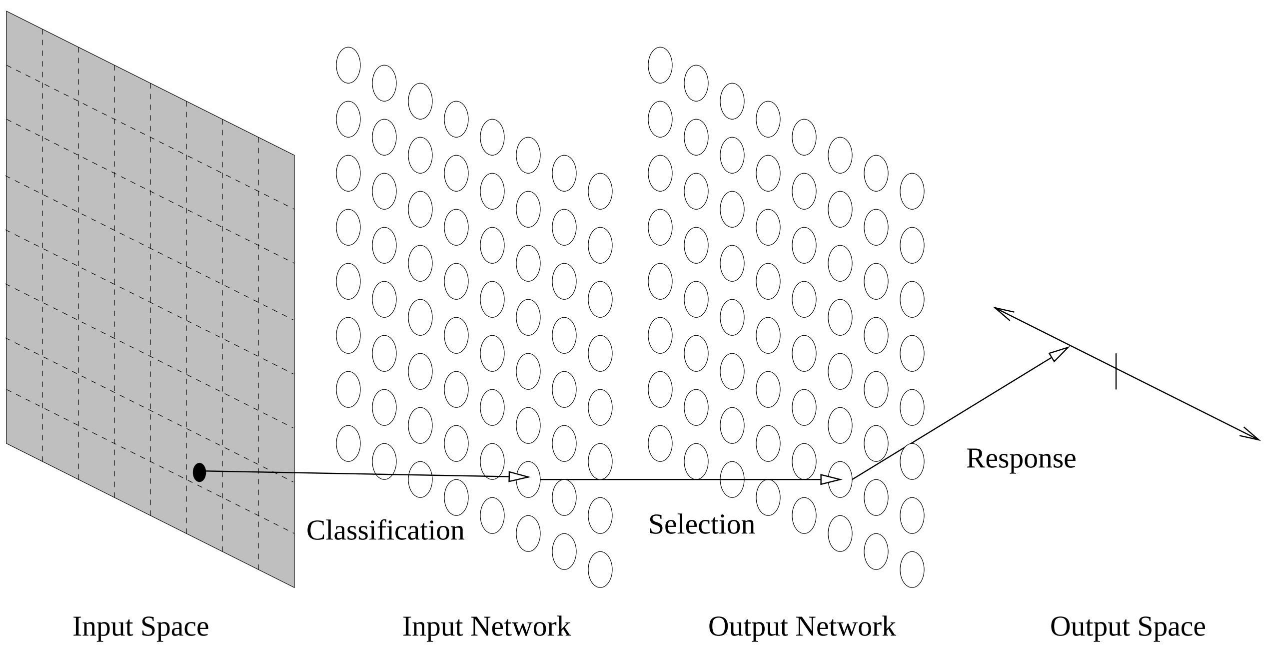


Figure 4.1: The system forms a mapping from input space to output space.

Figure 4.1 shows a two-dimensional input space and a thresholded one-dimensional output space. These dimensions were chosen because they are easily figured and understood and because they correspond to the formulation of the truck-backing problem that we have used the most. However, the learning system can be applied to input spaces of arbitrary dimensions. All that is required for the basic learning system is that the number of units in the input network and the number of units in the output network both equal the number of input partitioning regions (see Subsection 4.1.1). However, the output space needs to be one-dimensional and thresholded. (Possible extensions to the learning system to allow for multi-dimensional output spaces are discussed in Chapter 6.)

The system is trained in a series of *trials*. A trial is divided into discrete time units, and lasts from an initial configuration of the controlled system until a success or failure signal is generated. For the truck-backing problem, a trial begins with the rig at some distance and orientation to a goal. The inputs to the learning system are the angle between the spine of the truck and the spine of the trailer, known as the *hitch angle*, and the angle between the spine of the trailer and the goal, known as the *goal angle* (see Figure 4.2). This is the learning system's two-dimensional input space — note that distance is not provided to the system. On each time step the learning system gets a new reading of the angles as input and selects a right or left turn for the steering wheels. This is the learning system's thresholded one-dimensional output space. The truck then rotates its drive wheels through some angle, driving the vehicle backwards. This is repeated on each time step until the trial ends. The trial ends in success if the rear of the trailer reaches the goal. The trial ends in failure if the magnitude of either the hitch angle or the goal angle exceeds 90°. During a *run* of multiple trials, the learning system progresses from random performance to competence.

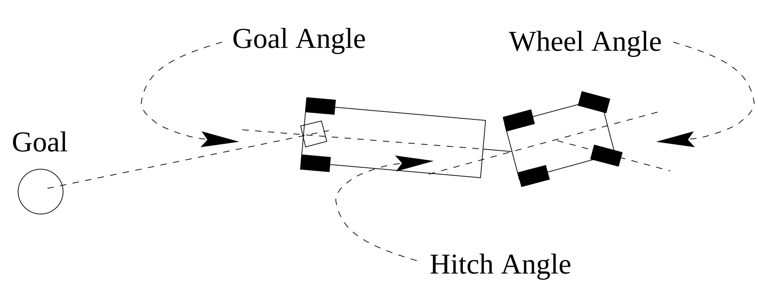


Figure 4.2: The basic truck-backing problem.

4.1.1 Input Classification

Since we wish to apply the learning system to situations about which little is known, we make no attempt to partition the input space according to our understanding of the problem under consideration, as has been done in some previous systems (e.g. [90, 91, 12, 132, 127, 164]). Instead, the input space is partitioned uniformly in each dimension.

While we could think of this partitioning as boxes into which input vectors fall, or as a coding of the vectors, as has been suggested by previous authors ([90, 91] and [12] respectively), we prefer to think of this as a neural network-based maximum-likelihood estimator that works by lateral inhibition. The reason for this conceptual preference will become more apparent when we describe input partition learning (see Section 4.2). For our actual implementation, however, there is no practical difference in how this classification takes place for the basic system.

Each unit of the input network has associated with it a weight vector $\bar{v} = (v_1, v_2, v_3, ...v_{d_I})$ where d_I is the dimensionality of the input space. One unit's weight vector is set equal to the middle point of each input partition region. On each time step a new input vector $\bar{x}$ is given to the input network and is compared to the weight vectors using an appropriate similarity measure S. We use the sum of the absolute values of the differences between vector components. Formally,

$$S(\bar{v}, \bar{x}) = \sum_{i=1}^{d_I} |v_i - x_i| \tag{4.1}$$

This is the well-known *Manhattan* or *city-block distance*. The Euclidean distance could be used instead, but at greater computational expense. For rectangular (or hyper-rectangular) partition regions, which we have due to the uniform partitioning in each dimension of the input space, these distance measures are functionally equivalent.

The unit s that has the weight vector most closely matching the input is said to be the "winner." Formally,

$$\exists s[S(\bar{v}_s, \bar{x}) \leq S(\bar{v}_u, \bar{x}) \mid \forall u \in U, s \in U] \tag{4.2}$$

where $\bar{v}_u$ is the weight vector of unit u and U is the set of all units in the network. If more than one unit satisfies this equation, then one is chosen by any arbitrary method. In the present study, the units are examined in order according to their numerical labels (see Subsection 4.1.2) and the one found first is chosen.

The winning input unit s is used to select the output unit for response on this time step.

4.1.2 Selection of a Responding Unit

In the basic learning system, the units in both the input and output networks are assigned numerical labels for use in selection of a responding unit. With d_I as the dimensionality of the input space, let p be the number of partition regions in each dimension. Each unit is given a d_I-tuple that uniquely labels that unit within its respective network. The values of the components of each d_I-tuple range from 1 to p.

This labeling means that each unit in the input network corresponds to exactly one identically labeled unit in the output network. This correspondence is used for the direct selection of a responding unit by the winning unit s from the input network.

4.1.3 Response Learning

The neurons which comprise animal brains are complex elements and only a few of their functions are approximated by today's artificial neural networks. One function of biological neurons which has not been approximated in the more standard connectionist systems is what is referred to as the eligibility trace [136]. It is known that many neurons become more amenable to change when they fire (see, e.g. [68]). This plasticity reduces with time, but provides an opportunity for learning based on feedback received by the neuron after its activity.

The eligibility trace has been used in reinforcement learning as one of two basic mechanisms used to deal with the temporal credit assignment problem [136]. This is the problem of determining which actions deserve credit for successes and which deserve blame for failures when reward and punishment signals are dependent on multiple actions taken over some period of time. While it is often used in conjunction with *adaptive heuristic critic* mechanisms, it is used here alone, to simplify the algorithm as much as possible (see Chapter 6).

All units in the output network of the proposed system have an eligibility value associated with them. At the start of each trial, all units are given an eligibility value of zero. When an output unit first *fires* (is selected by the winning unit from the input network to give an output response), its eligibility is increased by a preset amount, the *base eligibility value*, which is uniform for all units in the network. The eligibility value for each unit decays exponentially with time, providing a trace of eligibility that allows for adaptation. (See Figure 4.3.) The greater the eligibility value of a unit when a reward or punishment event occurs, the greater its adaptation.

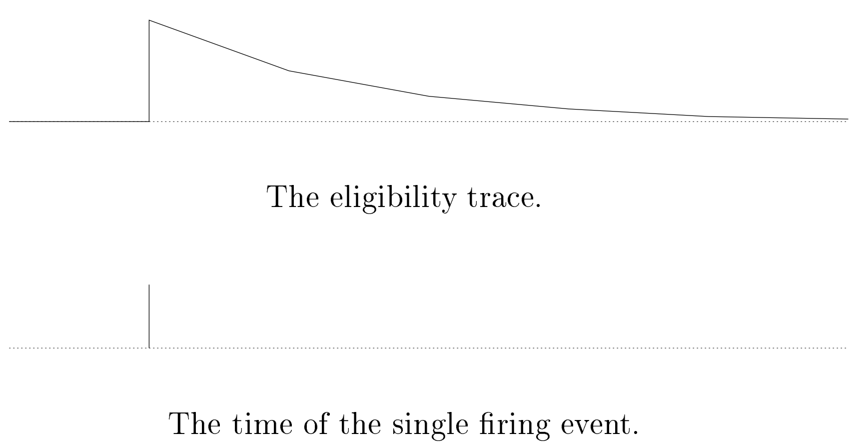


Figure 4.3: An exponentially decaying eligibility trace due to a single firing event.

For units that fire more than once during a trial, the trace becomes a *saturating* *trace*. This is in distinction from the more widely known *accumulating trace* and the *replacing trace* [136]. In a replacing trace, the total eligibility value of a unit that has just fired is simply the base eligibility value, regardless of that unit's previous eligibility value. In an accumulating trace, the total eligibility value of such a unit is its previous value plus the base eligibility value. In a saturating trace, the total eligibility value of a unit that has just fired is its previous value plus the base eligibility value, unless that total would exceed the saturation value, in which case the total is set equal to the saturation value. (See Figure 4.4.)

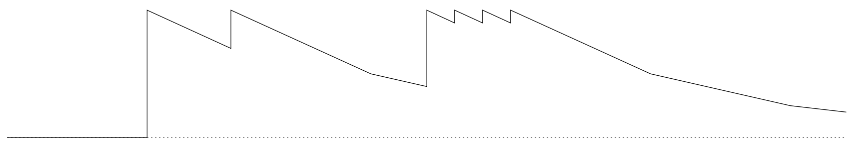
Replacing traces can be seen as embodying a recency heuristic — more credit is given to more recent events. Accumulating traces embody an additional frequency heuristic — more credit is given to events that happened more often. With a saturating trace, the total eligibility of a unit for adaptation depends on both the frequency and recency of its firing, however, the frequency contribution to the total eligibility is bounded. A saturating trace was chosen over an accumulating trace to prevent overflow errors with our integer calculations.

In addition to an eligibility value, each unit in the output network also has an output weight. This weight vector is initially given a random value. The output weights are used to determine the system's response to an input vector. For the truck-backing application, the weight value of the firing unit is examined for its sign alone. If the weight is positive, the wheels of the truck are turned to the right. Otherwise, the wheels are turned to the left.

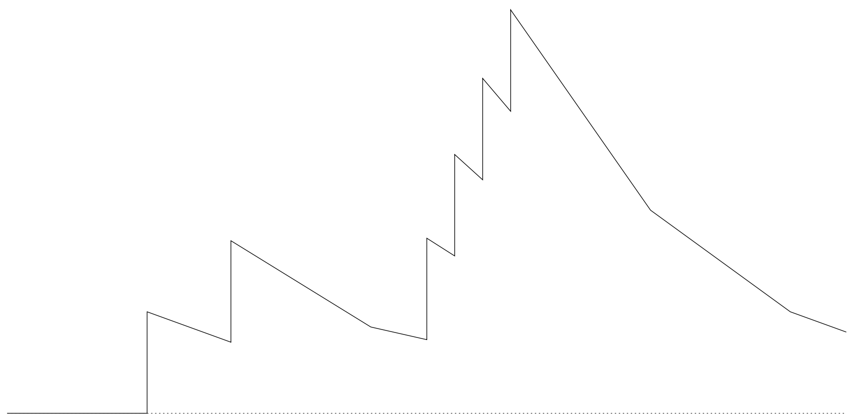
When reward or punishment occurs, all output weights are updated according to

$$w^{new} = \operatorname{sign}(w^{old})(|w^{old}| + e\,\sigma(T)\,f) \tag{4.3}$$

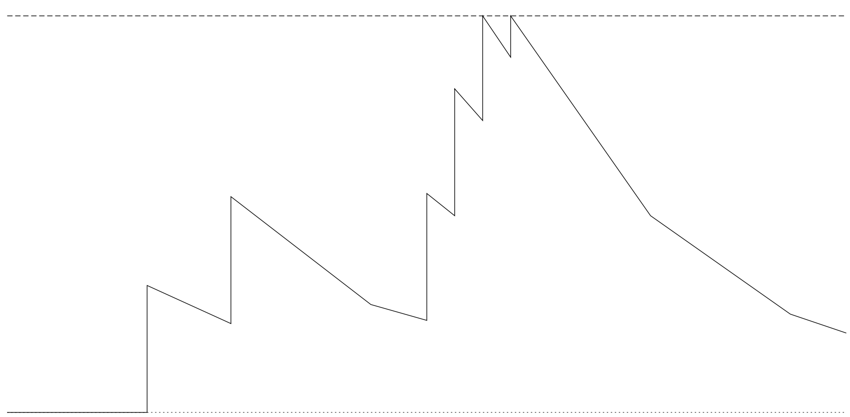
where w is the output weight, e is the eligibility for adaptation, σ is a scaling function that changes with the trial number T, and f is a feedback signal (+1 for success, -1 for failure). The scaling function σ is used to allow for large changes to the weights in early training trials and smaller changes in subsequent trials.



A replacing eligibility trace.



An accumulating eligibility trace.



A saturating eligibility trace.

The times of the firing events.

Figure 4.4: Replacing, accumulating, and saturating eligibility traces resulting from multiple firing events.

Large weight changes in early trials are permissible because the initial output weights are entirely random and therefore not important. However, smaller changes in later trials are desirable because as the system gains experience it should begin to retain what it has learned. Each new learning experience should be balanced against what has been learned before, so that when the system has gone through many trials it should not be greatly influenced by a single new experience. For backing a truck with a single trailer, σ is defined to be

$$\sigma(T) = \frac{1}{1 + \frac{T-1}{10}} \tag{4.4}$$

where T is the trial number. (Since we use integer arithmetic, we calculate $e\,\sigma(T)$ by dividing e by 1 for the first 10 trials, dividing e by 2 for the second 10 trials, dividing e by 3 for the third 10, etc.)

4.1.4 Performance of the Basic System

This basic system is able to learn good performance on a simulated truck-backing task. In this task, each trial was started with the rear of the trailer six feet from the goal and with initial random goal and hitch angles. The random goal angles were uniformly distributed over the range $-45°$ to $+45°$ and the random hitch angles were uniformly distributed over the range $-15°$ to $+15°$. Given an eight by eight partitioning of the input space, and therefore input and output networks of sixty-four units each, the system was able to arrive at a high level of success, roughly 90%, in 500 trials. Average results for 100 runs on this task, each starting from random performance and progressing to competence, are shown in Figure 4.5, and a typical trial from near the end of one run is shown in Figure 4.6.

This basic learning system bears some similarities to the BOXES system of Michie and Chambers, but is most similar to the Associative Search Element (ASE) of Barto, Sutton, and Anderson. Barto et al. paired the ASE with a second element, known as the Adaptive Critic Element (ACE) to improve its performance on their chosen

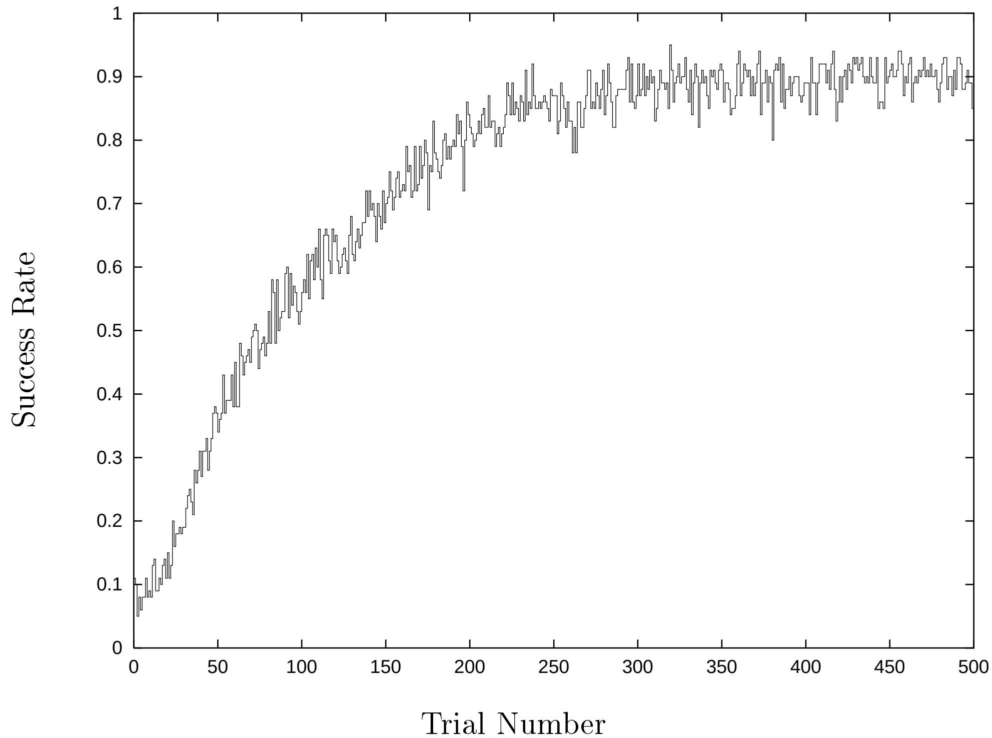
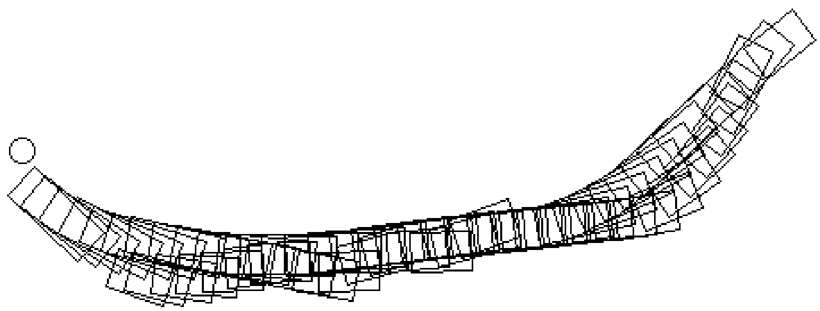


Figure 4.5: Average performance for 100 runs of 500 trials each using the basic learning system with a uniform eight by eight partitioning on the truck-backing task.

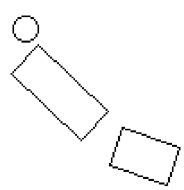
task of pole-balancing. While our basic learning system is compatible with adaptive heuristic critic methods such as ACE, we have chosen not to use such methods in our learning system, due to the computational overhead they require (see Chapter 6).

While the basic learning system is able to achieve good performance on the truck-backing task using a uniform eight by eight partitioning of the input space, questions arise. One such question is, would it be possible for the system to achieve equally good performance using fewer neural units? The simplest way to answer this question is to reduce the number of partitioning regions, and hence the corresponding number of neural units in both the input and output networks. With a uniform six by six partitioning, and thirty-six neural units in each network, the system performance declines to an average success rate of approximately 75% after 500 trials (see Fig-

A random starting position.



The movement of the rig, shown every tenth time step.



The final position of the rig at the goal.

Figure 4.6: An example trial using the basic learning system with a uniform eight by eight partitioning. The trial was sampled from near the end of a run of 500 trials.

ure 4.7). This seems to imply that such a reduction in partition regions and neural units will result in poorer system performance. However, perhaps the difficulty is not the lack of partition regions, but their boundaries. Since we do not wish to create a system that requires user analysis of the task to determine more appropriate boundaries, the answer seems to lie in having the system learning its own partitioning. This possibility is addressed in Section 4.2.

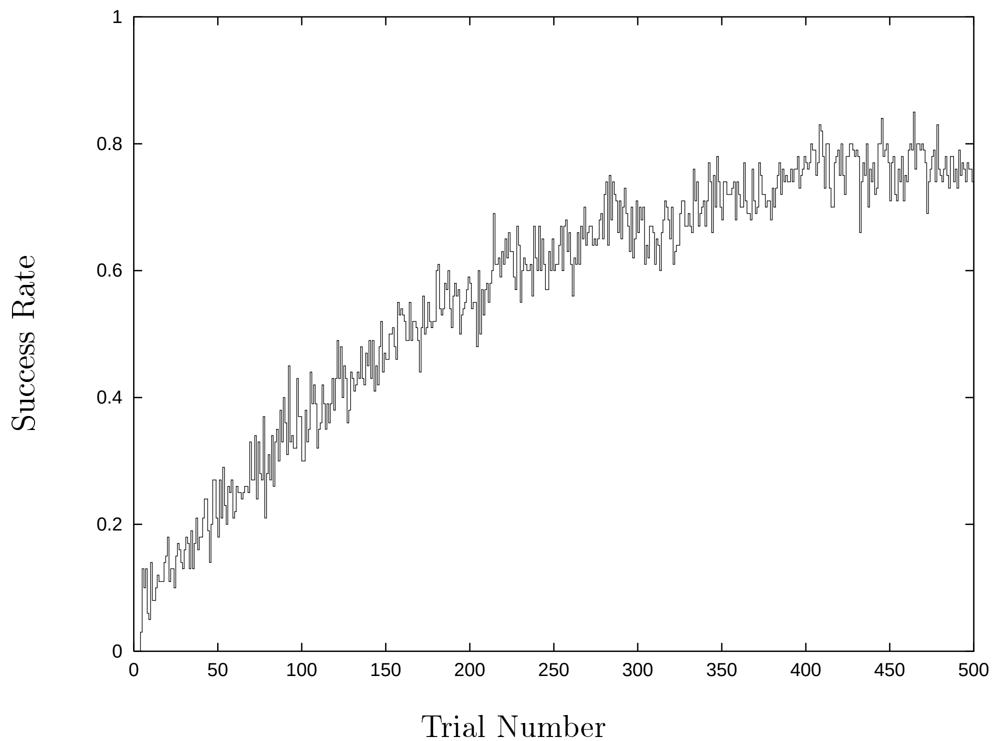


Figure 4.7: Average performance for 100 runs of 500 trials each using the basic learning system with a uniform six by six partitioning on the truck-backing task.

Another question is, would it be possible for the system to achieve good performance more quickly? In the basic learning system, the units of the output network learn completely independently of one another. This independence, while potentially useful in some cases, may not always be so. Instead, it may contribute to the *structural credit assignment problem*. If some environment states are rarely encountered, it may be difficult to determine the correct control decisions for those states. We may, however, be able to make use of the continuity of the environment we inherit from the physical plant we are to control. This possibility is addressed in Section 4.3.

Naturally, these two possibilities beg a third question which is, can a learned partitioning be combined with an exploitation of the underlying continuity of the environment to create a system that learns rapidly while using fewer neural units? This possibility is addressed in Section 4.4.

4.2 Input Partition Learning

To learn a partitioning without supervision requires a self-organizing system. We have chosen to implement this using a version of Kohonen's Self-Organizing Map (SOM) [69, 70, 71]. The SOM makes use of a topological arrangement of the units. The d_I-tuple labeling of the units in the input network (see Subsection 4.1.2) can be seen as defining a set of coordinates for each unit in a d_I-dimensional topology space. This allows for the definition of a *distance* function for the units in topology space. This is typically defined as the Euclidean distance between coordinates in topology space. For the sake of computational efficiency, however, we use the maximum difference between coordinates of the units. E.g. for units (1,1) and (2,3) in a two-dimensional network, the distance between them would be $max(|1-2|, |1-3|)$ or 2.

The distance function is used indirectly through the definition of *neighborhoods*. A neighborhood may have any width from zero (the neighborhood is restricted to the unit itself) to twice the maximum distance between units in topology space (the entire network is in the neighborhood) and varies with trial number, starting large and subsequently decreasing. Formally, if U is the set of units u in the network, D the distance function defined on topology space, and W a trial number dependent function specifying the width of the neighborhood, then the neighborhood N of unit

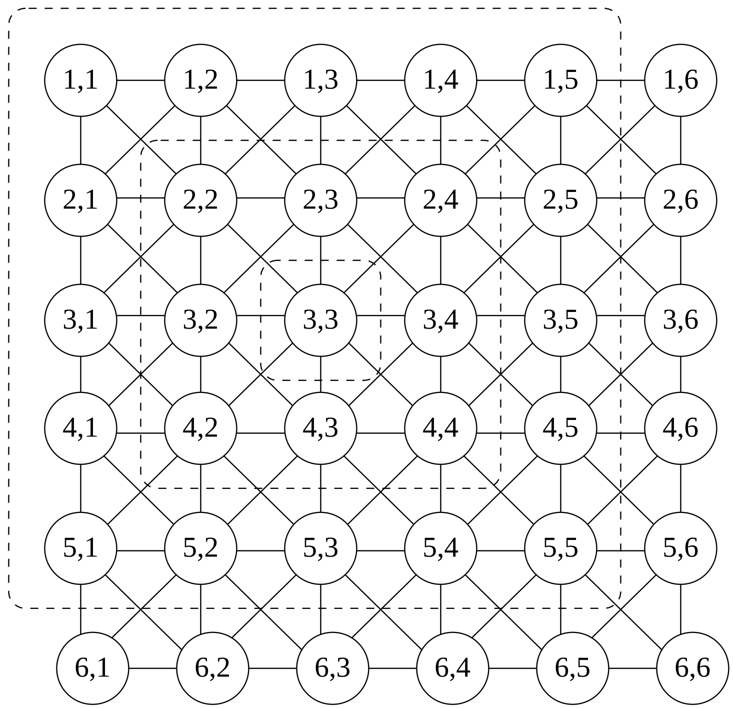


Figure 4.8: A six by six network of units with an eight-connected neighborhood relation, showing neighborhoods of width zero, two, and four around unit (3,3).

n on trial number T is defined as

$$N_n(T) = \{u \in U \mid D(n, u) \leq W(T)/2\} \tag{4.5}$$

4.2.1 Learning a Direct Partitioning

For a two-dimensional input space we may use a two-dimensional input network, as in the basic learning system, but with the addition of a planar topological arrangement. Using the maximum difference between coordinates of the units as a distance function gives us an eight-connected neighborhood relation. That is, each unit not along an edge of the network is connected to eight closest neighbors which, together with the unit itself, comprise a neighborhood of width two (see Figure 4.8).

Using such a neighborhood relation defined on the input network, we may allow for adaptation of the input weights defining the partition regions. At the start of each run, all units are initialized to random values, rather than to correspond to a uniform partitioning of the input space (see Figure 4.9).

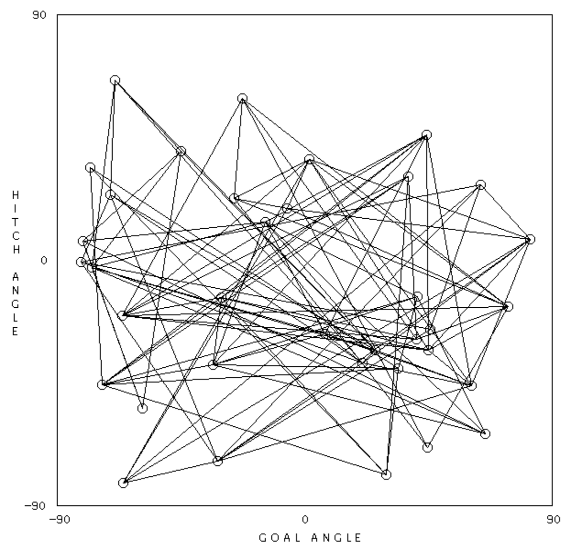


Figure 4.9: An initial random placement of a six by six network of input units in input space at the start of a run. Neighborhood topology shown as lines connecting units.

As with the basic learning system, on each time step a new input vector $\bar{x}$ is given to the input network and is compared to the weight vectors using the similarity measure S, a winning unit is determined, and an output unit is selected. However, in addition, the input vectors for each time step are recorded and at the end of the trial are again presented to the network for input-space partition-learning. As each input vector is re-presented to the input network, a winning unit is determined as before. Now, however, the weights of the winning unit and all other units in its neighborhood are updated according to the equation

$$\bar{v}^{new} = \bar{v}^{old} + \alpha(T)\gamma(t_{final})(\bar{x} - \bar{v}^{old})f' \tag{4.6}$$

where $\bar{v}$ is the weight vector being updated, f' is a variation of the feedback signal, α and γ are functions that determine the influence of the input vector ($0 \leq \alpha, \gamma \leq 1$), and t_{final} is the total number of time steps in this trial.

The feedback signal f' is given the value +1 for failure and 0 for success. This equation with this feedback signal can be seen as embodying the heuristic that if failure occurs, then the partitioning in the vicinity of the path traced through input space on that trial should be refined and if, on the contrary, success occurs, then the partition regions through which the path was traced are sufficient and should be maintained.

Like σ, the scaling factor for output weights, α starts large and decreases with trial number, as the system learns from random values to experience that should be retained. For this application, α is defined to be

$$\alpha(T) = \frac{1}{T} \tag{4.7}$$

The function γ ensures that all trials are weighted equally for purposes of partitioning the space, regardless of their length in time steps. It is defined as

$$\gamma(t_{final}) = \frac{2}{t_{final}} \tag{4.8}$$

Using Equations 4.2 and 4.6, there is competition amongst all the units in a network to be the unit selected and cooperation within a neighborhood as the units change their weights towards the same target. By repeated presentations of data to the network, the network self-organizes so that closely neighboring units have similar values for their weights (and therefore respond to similar input) while units that, according to the network topology, are far from one another have different values for their weights (and therefore respond to different input). (See Figure 4.10).

As the neighborhood size $W(T)$ and the gain parameter $\alpha(T)$ both decrease to zero, the network converges to a set of input weights that specify the final partitioning of the input space (see Figures 4.11 and 4.12).

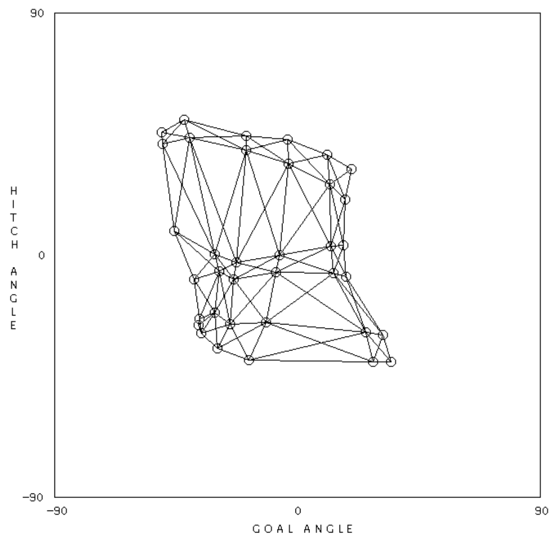


Figure 4.10: The placement of input units in input space after 50 trials. The neighborhood topology is shown as lines connecting units.

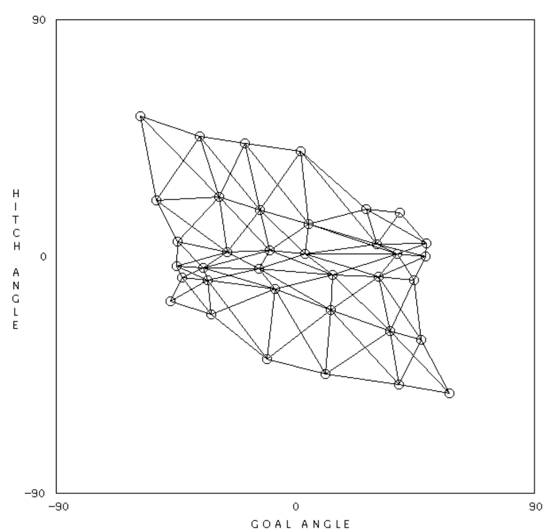


Figure 4.11: The arrangement of the input units in input space after 500 trials with neighborhood topology shown as lines connecting units.

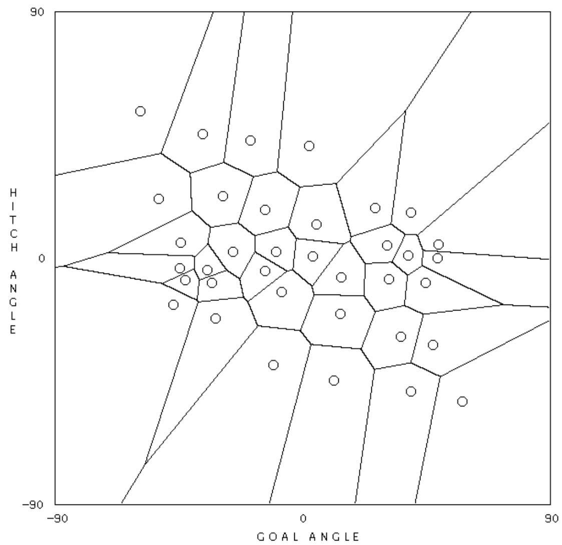


Figure 4.12: The arrangement of the input units in input space after 500 trials with the Voronoi diagram that corresponds to the partition regions.

4.2.2 Performance: Learning a Direct Partitioning

Using thirty-six units in both the input and output networks, the system learning a direct partitioning is able to achieve the high level of success (again, roughly 90% in 500 trials) that the basic system required sixty-four units per network to achieve, for the same task. Average results for 100 runs on this task, each starting from random performance and progressing to competence, are shown in Figure 4.13, and a typical trial from near the end of one run is shown in Figure 4.14.

Learning direct partitioning of the input space for control applications is similar to the work done in supervised learning by Ritter, Martinetz, and Schulten [87, 112, 113, 115, 88] and by Finton and Hu in reinforcement learning [38].

Learning a direct partitioning of the input space through self-organization allows the system to arrive at good performance on the truck-backing task with fewer units in both the input and output networks. However, it comes with considerable overhead. Part of this overhead is due to the fact that we are learning at all; if a good answer

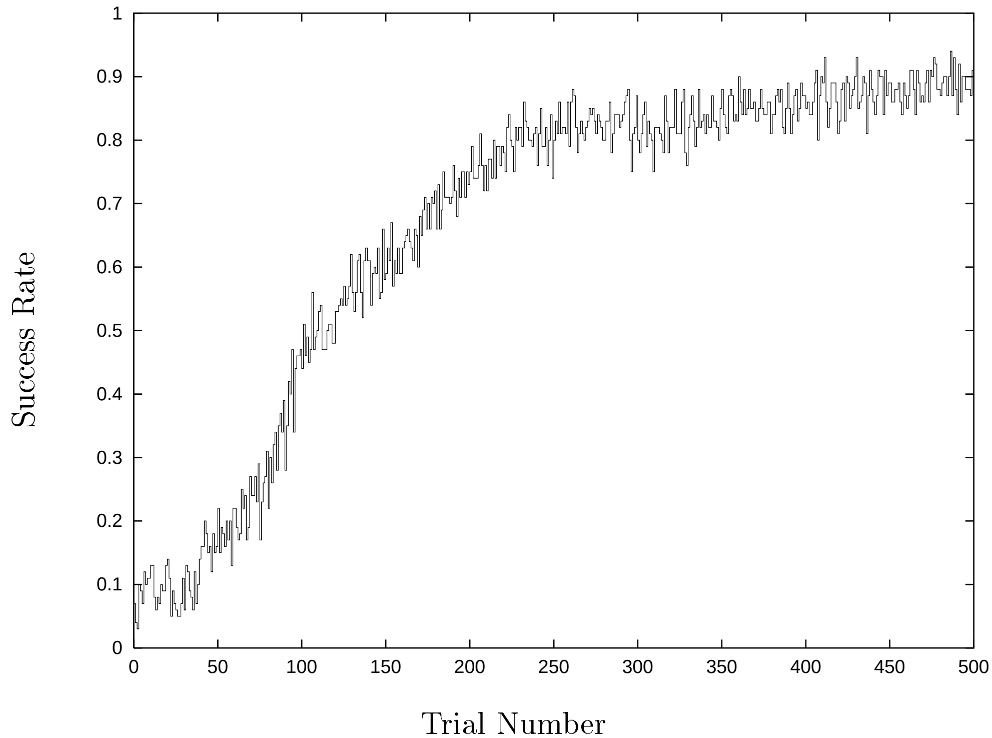


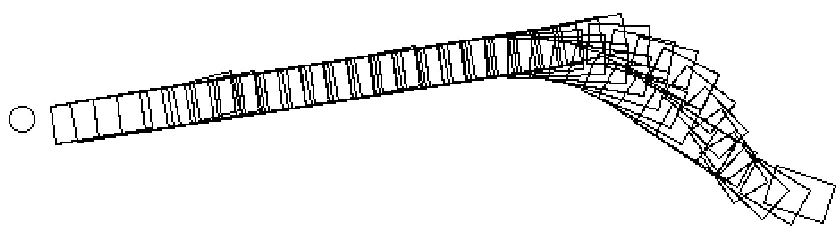
Figure 4.13: Average performance for 100 runs of 500 trials each using a six by six input network learning a direct partitioning on the truck-backing task.

is known, then having the system learn may not be as prudent as simply coding the answer. On the other hand, part of this overhead is due to the generality of the solutions that may be realized with this partition-learning system. As can be seen from Figure 4.12, the partition regions learned are relatively arbitrary convex polygons. Perhaps this is more general than is necessary for good performance.

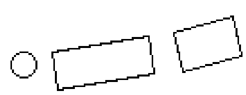
4.2.3 Learning an Indexed Partitioning

Rather than partitioning the entire input space with a single network of the same dimensionality as the space, we can use multiple lower-dimensional networks. If the dimensionality of one of these lower-dimensional networks is d_l, then that network is responsible for d_l-dimensions of the input space. To cover the input space then, the

A random starting position.



The movement of the rig, shown every tenth time step.



The final position of the rig at the goal.

Figure 4.14: An example trial for a six by six input network learning direct partitioning. The trial was sampled from near the end of a run of 500 trials.

sum of all d_l for the set L of all lower-dimensional input networks should equal or exceed the dimensionality of the input space, that is

$$d_I \leq \sum_{l \in L} d_l \tag{4.9}$$

where d_I is the dimensionality of the input space.

In the simplest case, d_l is one for all networks, and so each dimension of the input space is learned by its own network. This can be thought of as a component-wise classification; each component of the input vector is classified by a network that learns based on the distribution of that component. In the present application, the two-dimensional input space is covered by two one-dimensional networks.

The selection of units from all input networks should allow for the selection of a single corresponding unit from the output network. Further, for purposes of combining input partition-learning with cooperative response-learning (Subsection 4.4), this correspondence should be such that adjacent regions of the input space should result in the selection of output units adjacent in their topology space. For this reason, it is necessary for the output network to have the same dimensionality as the input space. Since we use two one-dimensional networks in the present application, the resulting one-dimensional topological coordinates of the units in the two input networks are used as indices into a two-dimensional output network. The combination of all weights from all input networks, then, can be seen as a partitioning of the input space. (See Figure 4.15.) This partitioning gives a discretization of the input space and allows for a single output response to be learned for each of the resulting discrete input regions.

Because the system uses only one-dimensional input networks, the partition regions are rectangular in the input space (see Figure 4.16). For higher-dimensional spaces, component-wise classification would result in partition regions that were hyper-rectangles.

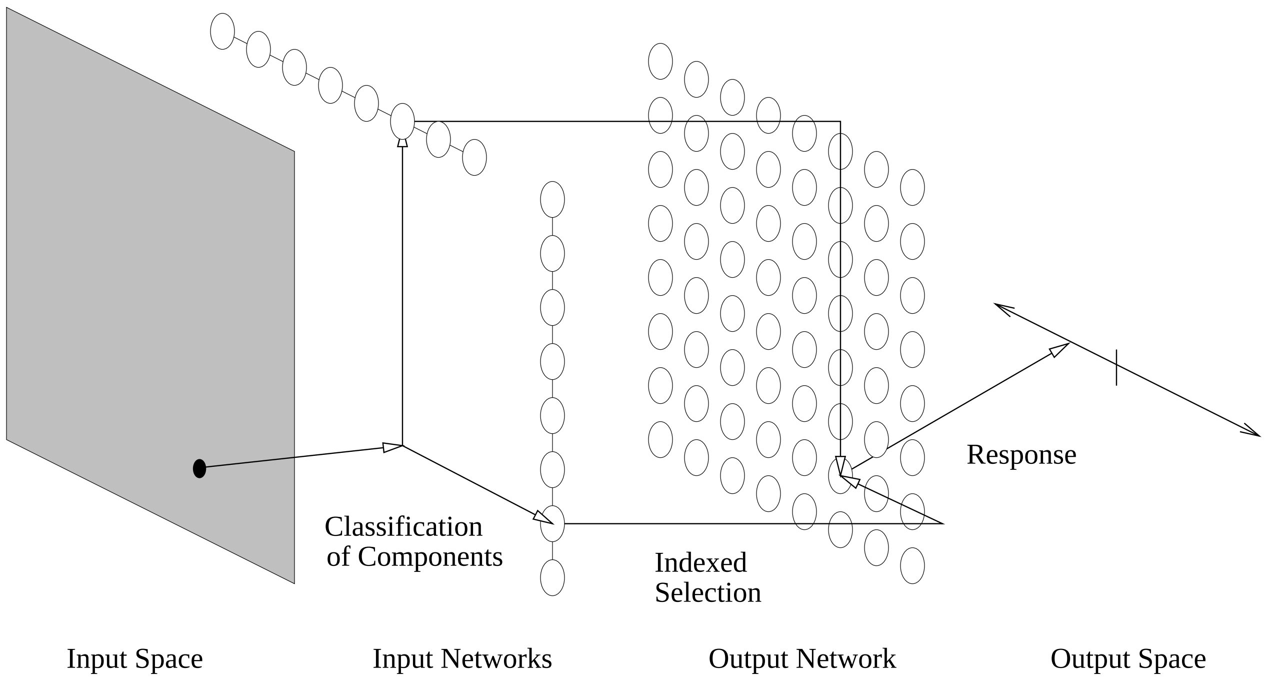


Figure 4.15: A mapping from input space to output space using component-wise classification and indexed partitioning.

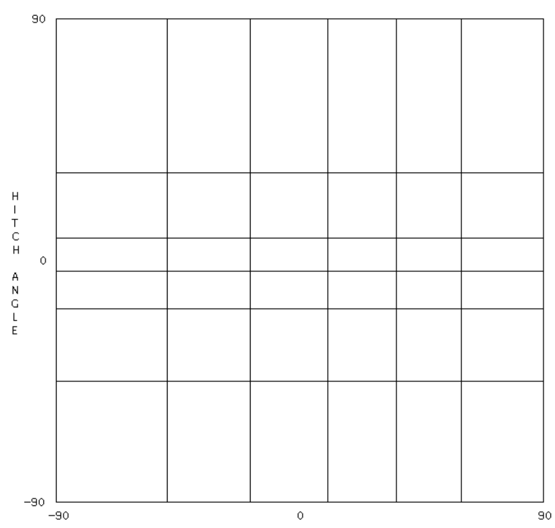


Figure 4.16: The indexed partitioning of the input space after 500 trials.

4.2.4 Performance: Learning an Indexed Partitioning

Using two six-unit input networks and a single thirty-six unit output network, the system learning an indexed partitioning is also able to achieve a level of success (90% or slightly higher in 500 trials) that the basic system required sixty-four units per network to achieve, for the same task. Average results for 100 runs on this task, each starting from random performance and progressing to competence, are shown in Figure 4.17. Besides requiring less overhead than the direct partition-learning, indexed partition-learning works more rapidly for this task (see Chapter 6).

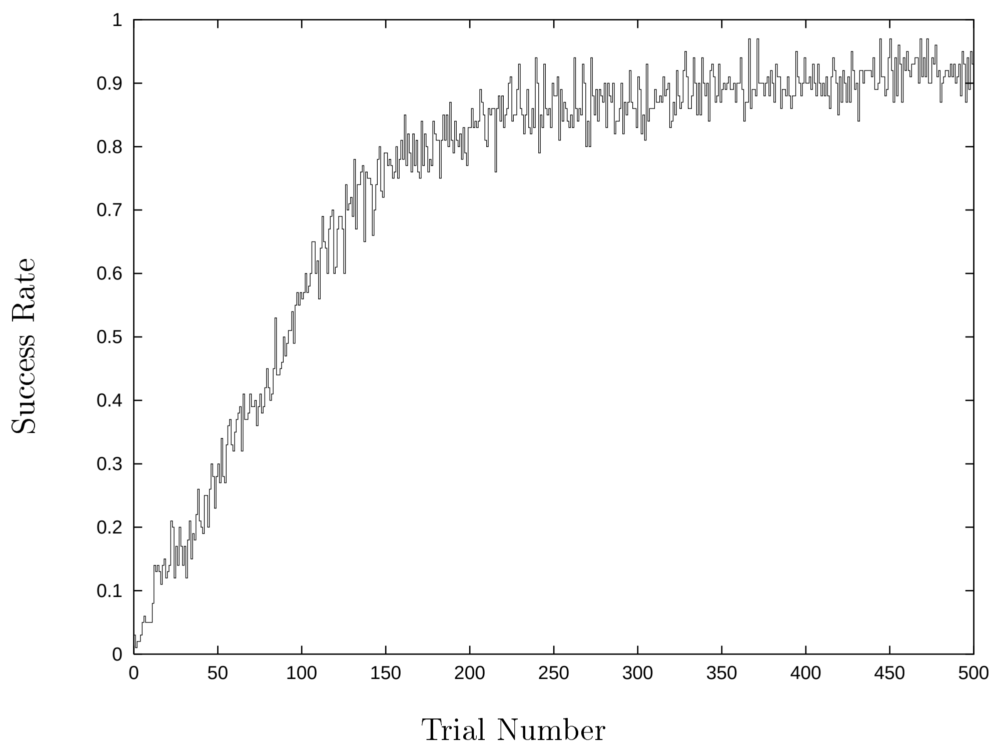


Figure 4.17: Average performance for 100 runs of 500 trials each using two six unit input networks to index into a six by six output network on the truck-backing task.

We are aware of no other research investigating the use of indexed partitionings or other uses of multiple lower-dimensional networks in place of a single higher-dimensional one.

4.3 Cooperative Response Learning

Because our environment is continuous, the individual states dealt with by our learning system are abstractions found through partitioning the input space. This means that we know about adjacency relations between states.

It is reasonable to postulate that similar control actions may be appropriate for environment states that are adjacent or nearby. By allowing units corresponding to adjacent partition regions of the input space to learn from the experiences of one another, we may be able to ameliorate the problem of structural credit assignment.

4.3.1 Neighborhoods and Response Learning

We define a topological arrangement of the units in the output network. For our two-dimensional output network, we use a planar topology with the units' numbered labels as coordinates in topology space and the maximum distance between units as our distance function. This gives us an eight-connected neighborhood relation (refer back to Section 4.2.1 and Figure 4.8).

Starting from the basic learning system, it is easy to arrange for adjacent units in the output network to give responses for adjacent regions of the input space. We simply assign input units with consecutively numbered labels to adjacent regions of input space. The direct selection of output units, then, assures us that output units adjacent according to their topological arrangement correspond to regions adjacent in input space.

This topological arrangement is used indirectly through the neighborhood relation on trial completion. After success or failure is signaled and each unit's weight is updated according to its eligibility, inter-neural cooperation takes place. This consists of units updating their weights a second time, this time based on the weight values of the other units in their neighborhood. For the present application, the neighborhood has a width of two for the first 250 trials and a width of zero thereafter.

Each individual unit i is influenced by its neighbors according to the following equation:

$$w_i = (1 - \beta(T))w_i + \beta(T) \sum_{n \in N_i} \frac{w_n}{m} \tag{4.10}$$

where each w is a weight, N_i is the neighborhood of unit i, m is the number of units in that neighborhood, and β determines the degree to which a unit's value is "smoothed" with that of its neighbors. Like σ, the scaling factor for individual output weight adjustment, β starts large and decreases with trial number. This means that each unit's value becomes more independent from those of its neighbors as time passes. In this application, β is defined to be

$$\beta(T) = \frac{1}{2 + \frac{T-1}{10}} \tag{4.11}$$

where T is the trial number.

4.3.2 Performance of the Cooperative Response-Learning System

Using sixty-four units in both the input and output networks, the system using a fixed input partitioning and cooperative response learning is able to achieve spectacular success on the basic truck backing task. Average results for 100 runs on this task, each starting from random performance and progressing to competence, are shown in Figure 4.18. An average success rate of greater than 95% is achieved by the system after only 50 trials and an average success rate of 100% is achieved by the system after roughly 250 trials. Clearly, the addition of cooperative response learning through inter-neural cooperation within neighborhoods has increased the learning ability of the basic system.

While a few authors have proposed methods of generalization or interpolation useful for dealing with the structural credit assignment problem, only Ritter, Martinetz, and Schulten [113] have used a neighborhood-based approach.

The performance of the cooperative output learning system with a fixed input partitioning is amazingly good on the task given but the system is somewhat brittle

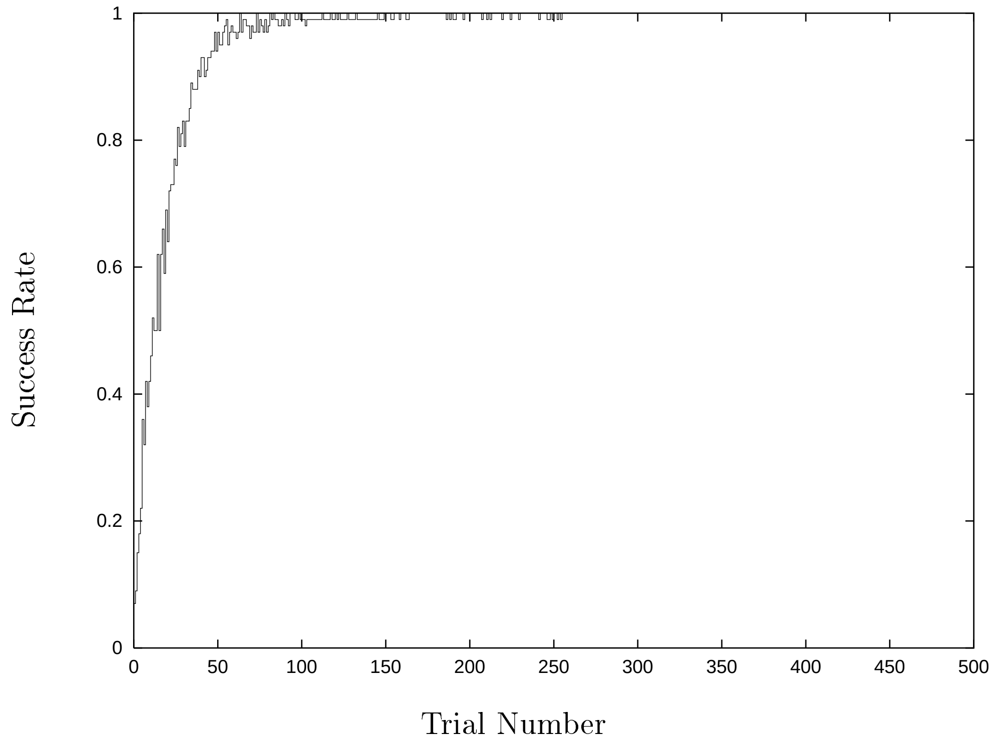


Figure 4.18: Average performance for 100 runs of 500 trials each using cooperative response learning with a uniform eight by eight partitioning on the truck-backing task.

— it depends on a good initial partitioning for good performance. If we give the system a more difficult version of the same basic task where the initial partitioning is less satisfactory, the results may be much worse.

One simple variation on the truck-backing task is to relax the goal angle constraint and start the rig with a greater range of initial hitch and goal angles. Rather than having a failure signal generated when the goal angle reaches $\pm 90°$, we may allow for goal angles up to $\pm 180°$, with the tracking head able to rotate indefinitely.

This version of the task does present one problem, however — the system is no longer sure to reach either a success or a failure state in a finite number of time steps. It is now possible for the rig to back roughly straight in any direction, or in a circle, indefinitely. For this reason, we must add another failure condition (see Figure 4.19). Failure is signaled if the rig backs for more than some number of time steps; one thousand was chosen as successful trials may be completed in a few hundred time steps, given the rig parameters (see Appendices A and B).

Allowing the starting goal angle to range up to $\pm 180°$ and the starting hitch angle to range up to $\pm 65°$ makes the task much more difficult, but all initial positions should allow for successful backing. (If an starting hitch angle of magnitude greater than $65°$ is given to the system, the rig will reach a jack-knife position, regardless of the steering command given.)

To apply the learning system with a fixed input partitioning and cooperative response learning to this version of the task, the input space is again evenly partitioned throughout its range. Because the goal angle range has been doubled, the goal angle partitions are now twice as far apart as with the previous version of the task.

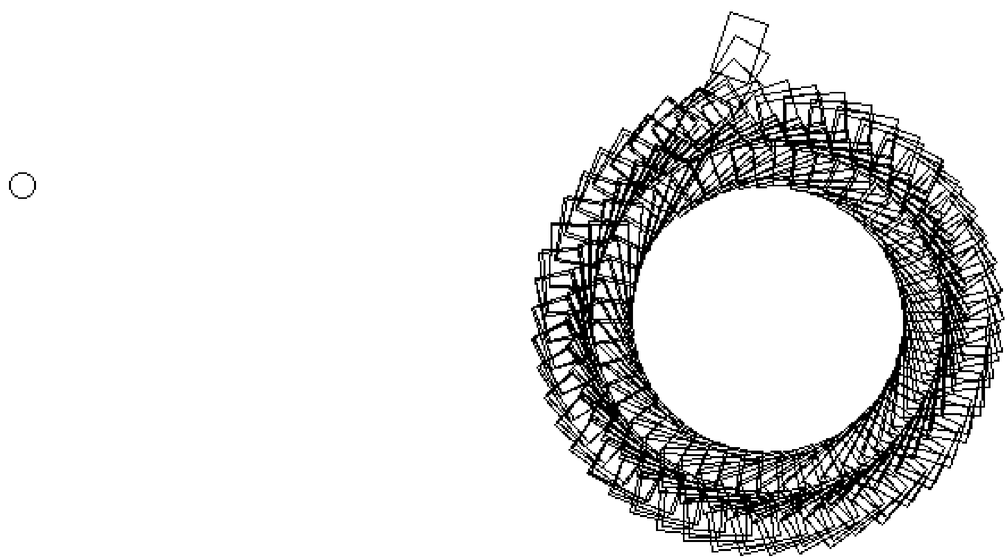
On this version of the task, the performance of the learning system with a fixed input partitioning and cooperative response learning is seriously degraded. Using a uniform eight by eight partitioning (sixty-four output units), the system is able to achieve less than a 65% average success rate in 500 trials (see Figure 4.20). Using a uniform six by six partitioning (thirty-six output units), the system performance drops to an average success rate of roughly 20% after 500 trials (see Figure 4.21).

One possible solution to the low success rate of the learning system on this version of the truck-backing task, is to have the system learn the input partitioning as well as use cooperative response learning.

4.4 Combining Input Partition Learning and Cooperative Response Learning

All mechanisms necessary for input partition learning and cooperative response learning have already been presented. To create a system to learn a direct partitioning

A random starting position.



The movement of the rig, shown every tenth time step.

The final position of the rig at time step 1000.

Figure 4.19: An example trial showing the additional failure condition needed with a tracking head that can rotate indefinitely.

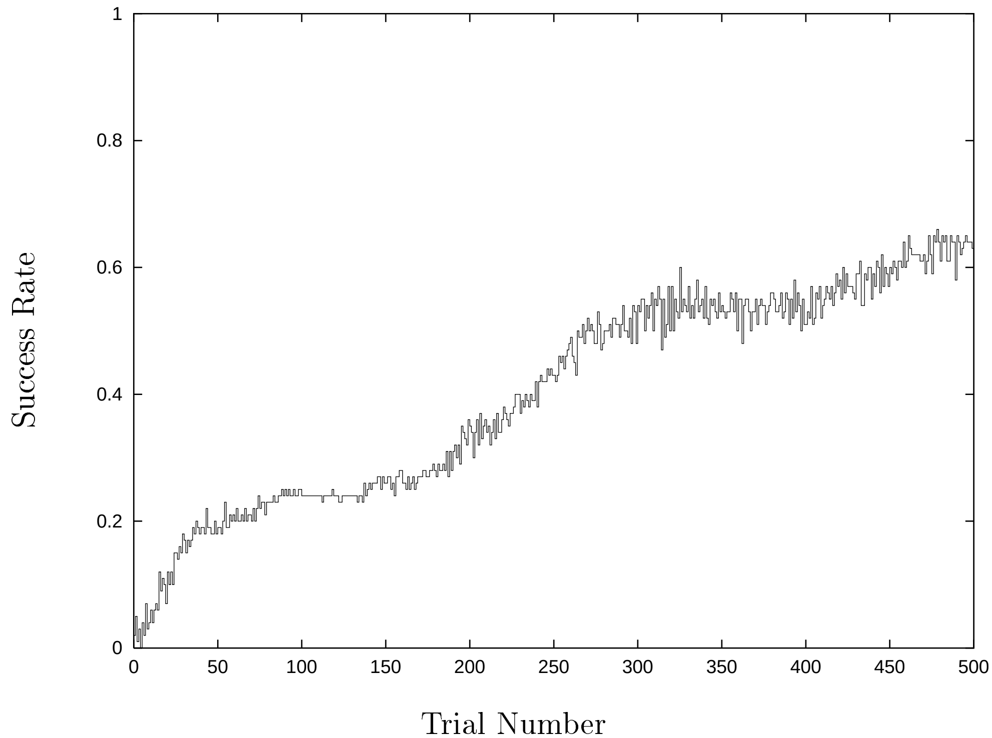


Figure 4.20: Average performance for 100 runs of 500 trials each using cooperative response learning with a uniform eight by eight partitioning on the truck-backing task with initial hitch angles up to $\pm 65°$ and initial goal angles up to $\pm 180°$.

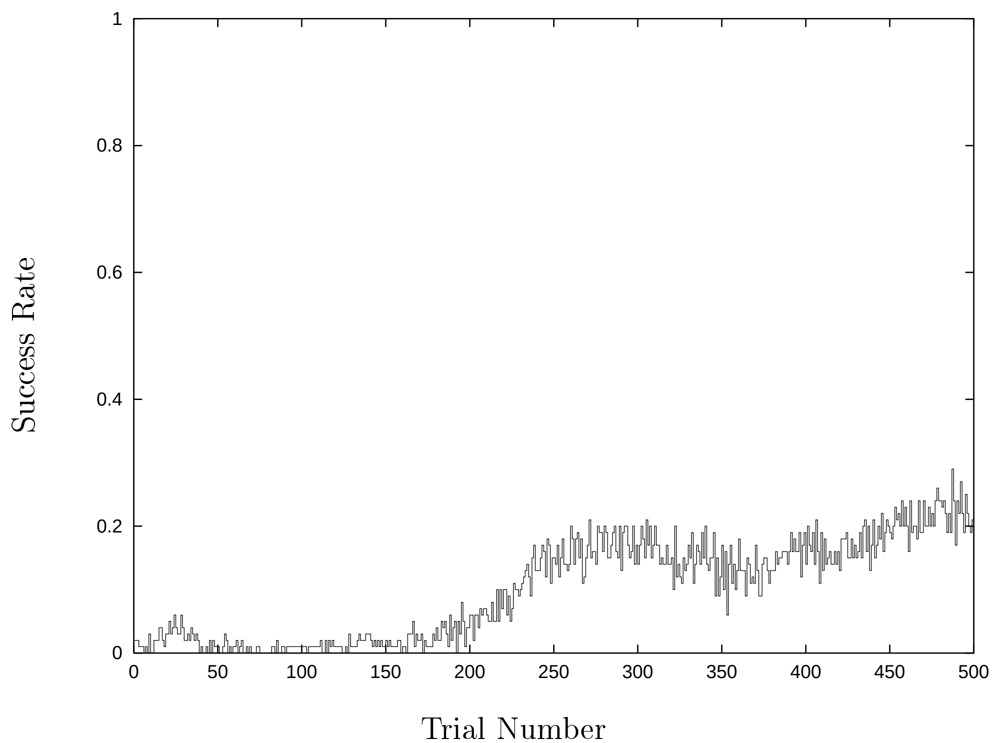


Figure 4.21: Average performance for 100 runs of 500 trials each using cooperative response learning with a uniform six by six partitioning on the truck-backing task with initial hitch angles up to $\pm 65°$ and initial goal angles up to $\pm 180°$.

and use cooperative response learning, all we need to do is combine these mechanisms. When an input vector is given to the input network, it classifies the vector and selects a responding unit producing both an output response and an eligibility trace for adaptation, just as with the basic learning system. At the end of each trial, the input weights are adjusted to give a new partitioning, just as described previously for learning a direct partitioning (Subsection 4.2.1) and the output weights are adjusted cooperatively, just as described previously for cooperative response learning (Section 4.3).

4.4.1 Performance: Learning a Direct Partitioning while Learning Responses Cooperatively

Using thirty-six units in both the input and output networks, the system learning a direct partitioning while utilizing cooperative response learning is able to achieve significantly better performance on the more difficult version of the truck-backing task presented above (Section 4.3). Average results for 100 runs with initial hitch angles up to $\pm 65°$ and initial goal angles up to $\pm 180°$ are shown in Figure 4.22. An average success rate of nearly 50% is achieved after 500 trials as compared with the average success rate of roughly 20% for the system using only cooperative response learning for this same problem and network size.

While this result clearly shows that better results may be obtained if a better than uniform partitioning is learned, there remains a big problem with this combined system. This problem can be seen if we look at the performance of this combined system with sixty-four units for both the input and output networks. Average results for this larger network size on 100 runs with initial hitch angles up to $\pm 65°$ and initial goal angles up to $\pm 180°$ are shown in Figure 4.23. Here the system only achieves a success rate of roughly 40% after 500 trials. This is *worse* than the success rate for both the smaller combined network (50%, see Figure 4.22) and the system using cooperative response learning with a uniform eight by eight partitioning (65%,

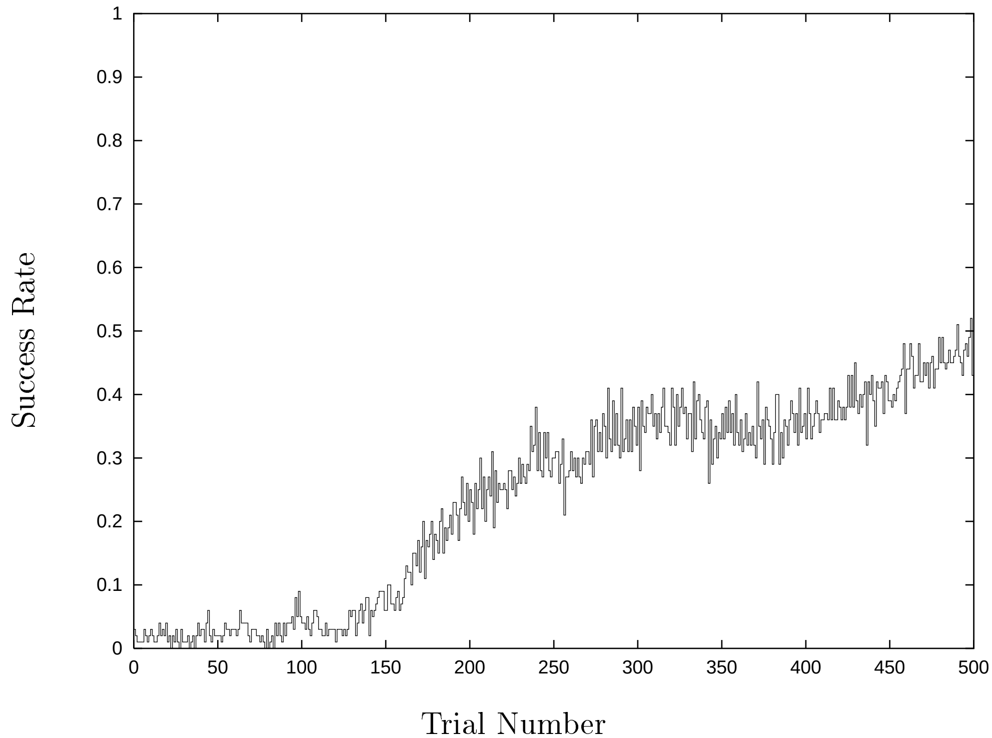


Figure 4.22: Average performance for 100 runs of 500 trials each using cooperative response learning while learning a direct partitioning with thirty-six units in the input network and thirty-six units in the output network. The learning system is applied to the truck-backing task with initial hitch angles up to $\pm 65°$ and initial goal angles up to $\pm 180°$.

see Figure 4.21). The obvious question is, why? This question will be answered in Chapter 6.

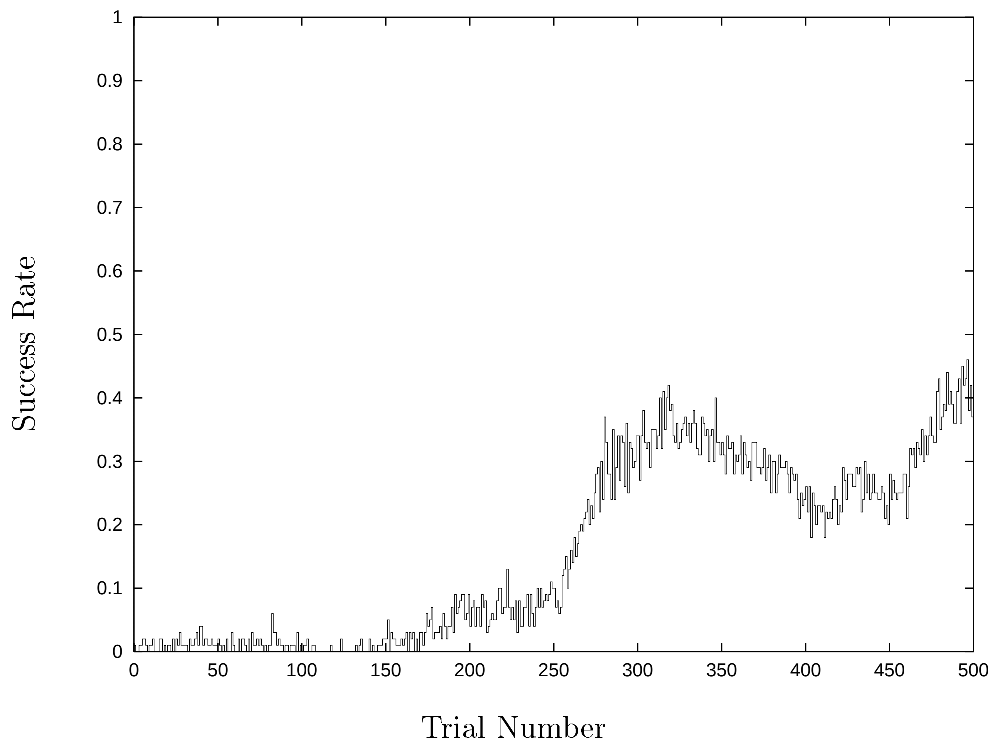


Figure 4.23: Average performance for 100 runs of 500 trials each using cooperative response learning while learning a direct partitioning with sixty-four units in the input network and sixty-four units in the output network. The learning system is applied to the truck-backing task with initial hitch angles up to $\pm 65°$ and initial goal angles up to $\pm 180°$.

4.4.2 Performance: Combining Indexed Partition Learning with Cooperative Response Learning

With two one-dimensional input networks of eight units each, and a two-dimensional output network of sixty-four units, the system combining indexed partition learning and cooperative response learning is able to achieve much better results than either the system with a fixed input partitioning and cooperative output learning or the system combining direct partition learning and cooperative output learning, when applied to the more difficult version of the truck-backing task. Average results for 100 runs on this task, each starting from random performance and progressing to competence, are shown in Figure 4.24. An average success rate of greater than 85% is achieved by the system in a period of 500 trials (versus roughly 65% for the fixed partitioning system and only 50% for the system learning a direct partitioning).

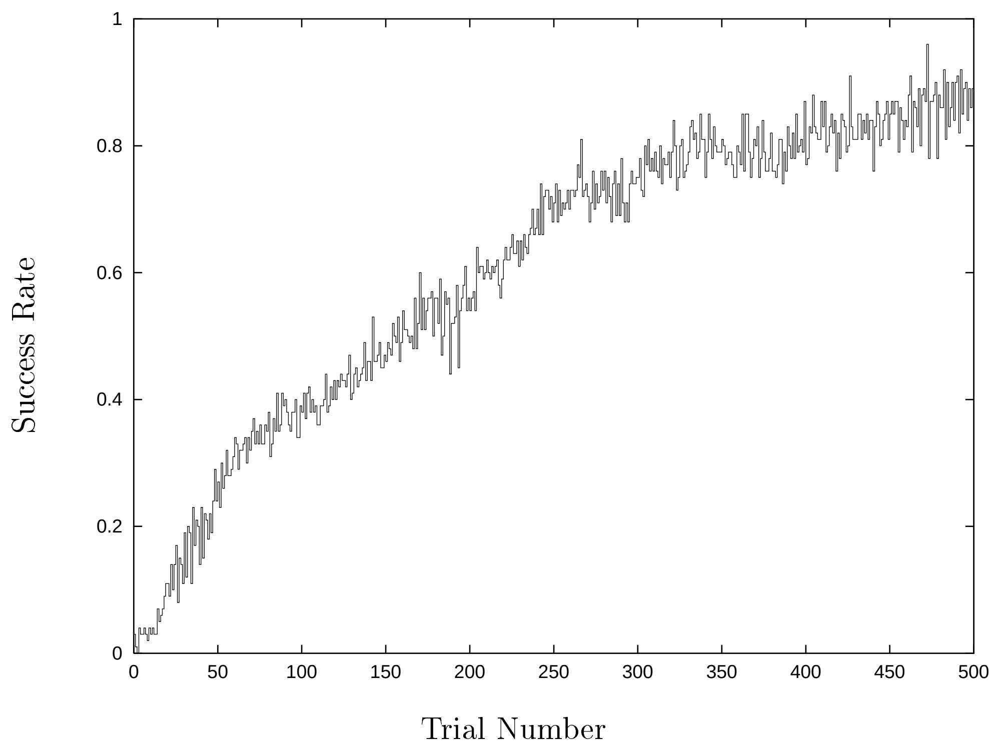


Figure 4.24: Average performance for 100 runs of 500 trials each using cooperative response learning with a learned indexed partitioning using two eight unit input networks and a single sixty-four unit output network. The learning system is applied to the truck-backing task with initial hitch angles up to ±65° and initial goal angles up to ±180°.

Perhaps even more impressively, with two one-dimensional input networks of six units each, and a two-dimensional output network of thirty-six units, the system using a learned, indexed input partitioning and cooperative response learning is able to achieve an average success rate of roughly 80% in 500 trials (versus 20% for the fixed partitioning system with thirty-six units per network and 40% for the system learning a direct partitioning). Average results for 100 runs on this task using the smaller networks is shown in Figure 4.25 and an example trial is shown in Figure 4.26.

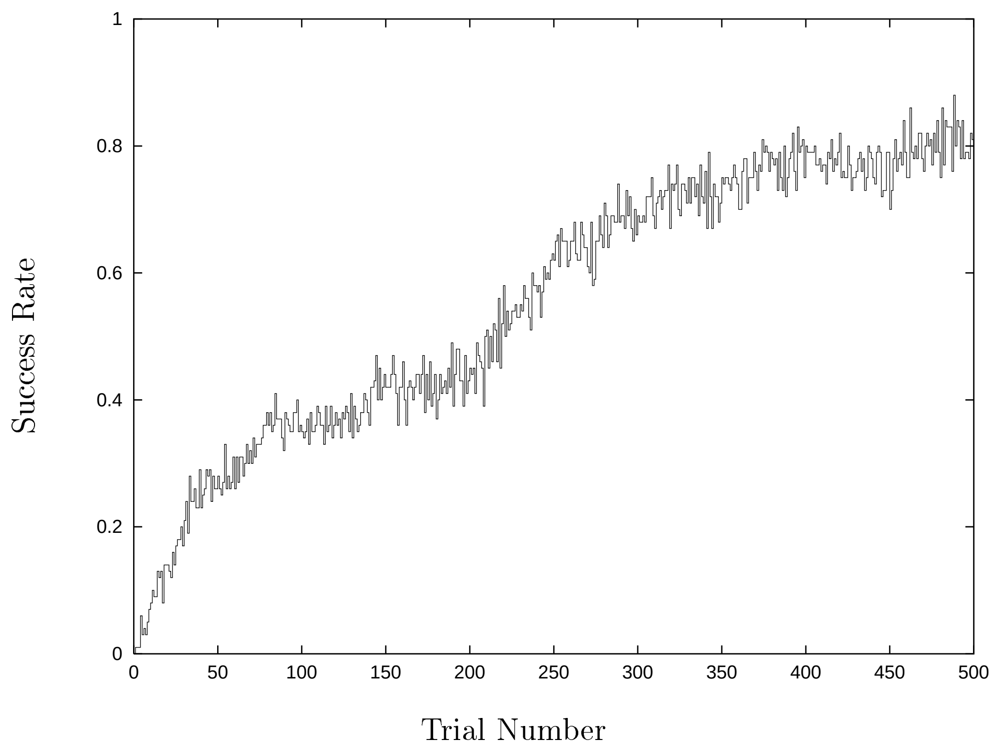
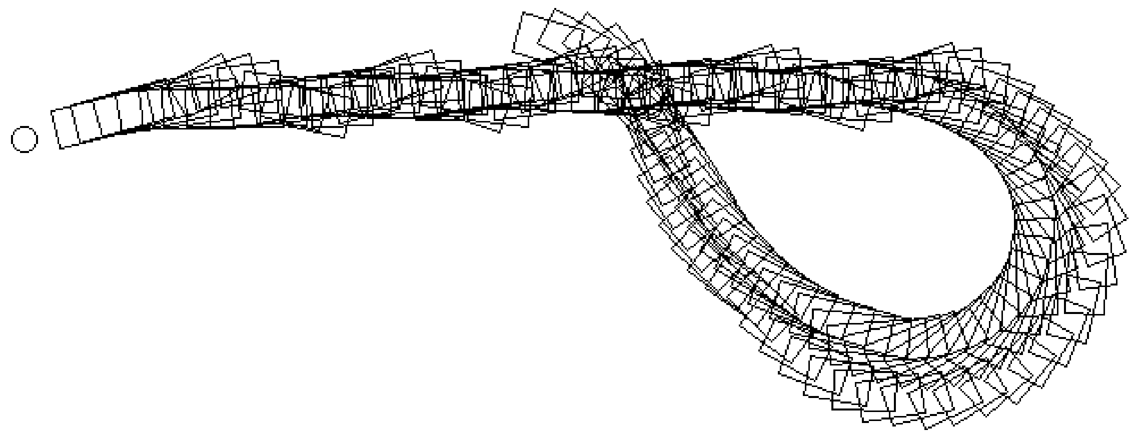
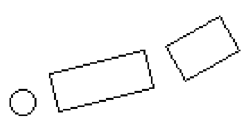


Figure 4.25: Average performance for 100 runs of 500 trials each using cooperative response learning with a learned indexed partitioning using two six unit input networks and a single sixty-four unit output network. The learning system is applied to the truck-backing task with initial hitch angles up to $\pm 65°$ and initial goal angles up to $\pm 180°$.

A random starting position.



The movement of the rig, shown every tenth time step.



The final position of the rig at the goal.

Figure 4.26: An example trial using the complete learning system with a learned, indexed partitioning using two one-dimensional, six unit input networks, and a thirty-six unit output network with cooperative response learning. The trial was sampled from near the end of a run of 500 trials.

Chapter 5

EXPERIMENTS AND RESULTS

Two types of experiments are presented: Simulation experiments and real-world experiments. Simulation experiments allow us to explore several variations on both learning system and application domain, yet still collect suffient data from which to draw conclusions. Real-world experiments allow us to ensure that the system is truly capable of learning on real robots, despite the noise, uncertainty, and constraints imposed by implementing the system using real robotic hardware.

In all of the experiments, both simulation and real-world, the truck-backing problem is used as our application domain. For most of the experiments, the rig consists of a truck cab hinged to a single trailer, as briefly mentioned in Section 4.1 and illustrated again in Figure 5.1. However, for one set of simulation experiments, the rig consists of a truck with two trailers as illustrated in Figure 5.2.

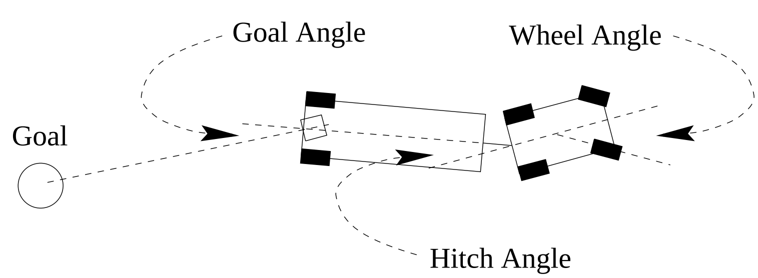


Figure 5.1: The basic truck-backing problem.

The input to the learning system comes from the goal and hitch angles and the output from the learning system is used to select the angle of the steering wheels at the front of the truck. Since the output is thresholded at zero, the steering wheels

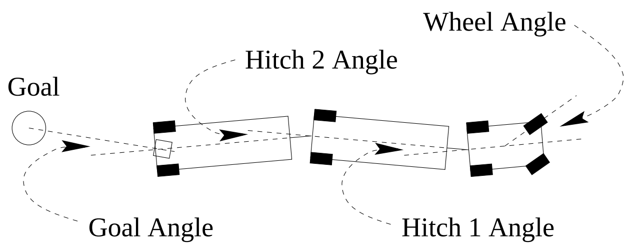


Figure 5.2: The two-trailer-backing problem

may either turn *hard right* or *hard left* with no other steering values possible. We have chosen to define a hard turn as 30° from parallel with the spine of the trailer.

The feedback to the learning system is a simple boolean value. If the rear of the rig reaches the goal, success is signaled. If one or more of the failure conditions (see Subsections 5.1.1, 5.1.2, and 5.1.3) occurs, failure is signaled. The system is clocked to operate in discrete time units. No other sensory data or feedback is available to the learning system.

Whereas some formulations of the truck-backing problem include as inputs such variables as the x and y position of the robot in Cartesian space, we do not. This is because we wish to test our learning system on a real robot as well as in simulation. While it is trivial to determine a robot's x and y coordinates in simulation (in fact, these variables are found in the mathematical model used in the simulation) it is extremely difficult for the real robot to acquire its own x and y coordinates relative to some fixed origin in the environment.

Each experiment uses a particular version of the learning system and consists of a number of runs. In each of the simulation experiments the number of runs is 100 and the results of all 100 runs are averaged together and presented in a single graph. For each run, the learning system starts with random values for all of the weights it is to learn; experience is gained over a series of learning trials.

At the start of each learning trial, the rig is positioned at some predetermined distance from the goal and with an initial set of goal and hitch angles randomly selected from within a predetermined range. The initial distance to the goal and the ranges of the starting angles varies from experiment to experiment, but is fixed within all the trials of each particular experiment. In all of the experiments the distribution of the randomly selected starting angles is uniform within the given range.

Each trial proceeds with the rig rotating its drive wheels (located on the truck cab) in reverse, pushing the rig backwards. On each time step, the current goal and hitch angles are given as input to the learning system and the system responds with its output — the steering wheels are turned hard right or hard left and the drive wheels rotate through some fixed angle. This continues until either success or failure is signaled and the trial ends. At the end of each trial, the learning system updates its weight values based on the experiences of that trial.

As long as there are more trials remaining in a run, the rig is again positioned at the initial distance from the goal but with new random goal and hitch angles. The weights of the neural units in the learning system are not reset in the middle of a run; this ensures that the experience of the new trials is added to that already gained.

When the final trial of a run is completed, however, all of the learned weights are given new random values and another run is started. When all runs are completed, the experiment is concluded.

5.1 Simulation Experiments

The simulation experiments cover three cases from the application domain of truck-backing:

1. Backing a truck with a single trailer starting from relatively small initial angles.

2. Backing a truck with a single trailer starting from relatively large initial angles.

3. Backing a truck with two trailers.

		Response Learning	
		Individual	Cooperative
	Fixed	1. Fixed & Individual	2. Fixed & Cooperative
Partition Learning	Direct	3. Direct & Individual	4. Direct & Cooperative
	Indexed	5. Indexed & Individual	6. Indexed & Cooperative

Table 5.1: Six versions of the learning system.

Both of the first two cases have been mentioned previously (see Chapter 4). They provide variation in the details of the task to be learned and so help to reveal strengths or weaknesses in the system that may have gone unnoticed in a single set of experiments. The third case allows us to look at a generalization of the problem by expanding the input space to three dimensions (from two). The presentation of the simulation results is organized around these three cases and they are treated in Subsections 5.1.1, 5.1.2, and 5.1.3 respectively.

For all three cases we examine the learning abilities of six versions of the learning system. These six versions come from our two possibilities for response learning — individual or cooperative — paired with our three possibilities for input partition learning — fixed (no learning), direct, and indexed. The six versions are listed in Table 5.1. The layout of the six versions in the table is followed in the pages of result graphs that follow. That is, each results page contains six graphs — one for each version — with the graph for the basic learning system (version 1, fixed input partitioning and individual response learning) in the upper left, the graph for the simple cooperative response learning system (version 2) in the upper right, and so on.

To examine how the versions of the learning system function using networks of different sizes, results are presented for each version with networks ranging in size from two neural units per dimension of the input space up to ten neural units per dimension. Note that for application cases 1 and 2 (both using a single trailer), there are two dimensions in the input space while for application case 3 (using two trailers) there are three dimensions in the input space. This means that for of all versions of the learning system, for cases 1 and 2 there are from four (two by two) to one hundred (ten by ten) neural units in each system's output network, while for case 3 there are from eight (two by two by two) to one thousand (ten by ten by ten) neural units in each system's output network.

Further note that for learning system versions 1 through 4 (with either fixed or learned-direct partitioning), each system has the same number of neural units in its input network as it does in its output network. However, for learning system versions 5 and 6 (with a learned-indexed partitioning), there are generally fewer neural units in the input networks. For application cases 1 and 2, the learned-indexed partitioning versions has from four neural units (two networks with two units each) to twenty neural units (two networks with ten units each) in their input networks, while for application case 3, they have from six neural units up to thirty in their input networks. The relevance of these differences is discussed in Chapter 6.

For each simulation case, results are summarized in a table that lists *ultimate success rate*. The ultimate success rate is defined to be the average success rate during the last 50 trials of each run.

The equations used in the simulation are given in Appendix A.

5.1.1 Truck-Backing with Small Angles

In this case of truck-backing with a single trailer, the following parameters are used to determine the initial starting position of the rig:

- The rear of the trailer is started 6 feet from goal.
- The goal angle range is from -45° to $+45^\circ$.

- The hitch angle range is from -15° to $+15^\circ$.

New initial conditions are chosen randomly at the start of each trial with a uniform distribution over the entire range.

The basic parameters of the simulated rig are:

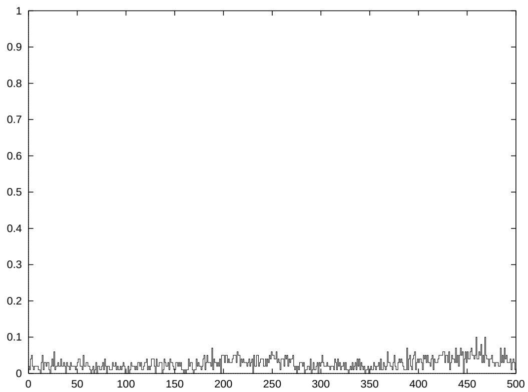
- The truck cab length is 7 inches.
- The trailer length is 11 inches.
- The distance moved by the front wheels of the truck on each time step is 0.25 inches.

For further details of the simulated rig, see Appendix A.

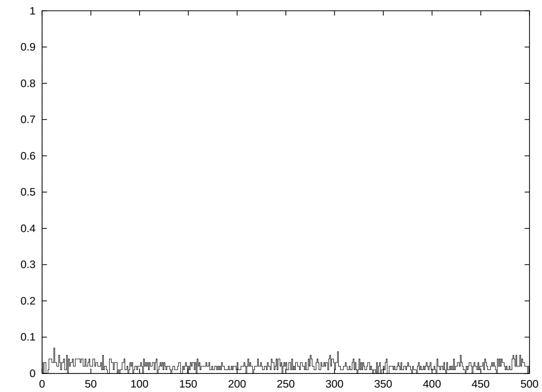
The boundary values for both the hitch and the goal angle are -90° and $+90^\circ$; if the value of either angle exceeds these boundaries, failure is signaled. Each run is 500 trials in length and each graph is made by averaging 100 runs together. Table 5.2 summarizes the ultimate success rates.

Neural Units Per Dimension	Learning System Version: 1	2	3	4	5	6
2	4.24%	2.34%	47.08%	45.40%	3.24%	2.94%
3	0.06%	0.18%	42.14%	42.16%	2.18%	1.90%
4	30.58%	78.02%	81.38%	80.11%	56.36%	47.26%
5	6.78%	99.40%	90.12%	91.12%	77.24%	77.66%
6	66.12%	92.82%	88.48%	92.78%	92.12%	93.00%
7	88.60%	99.82%	88.50%	94.08%	93.10%	95.94%
8	92.10%	100.00%	84.84%	93.92%	91.44%	95.84%
9	91.34%	95.96%	82.12%	90.40%	92.84%	97.24%
10	92.82%	98.70%	73.80%	86.26%	91.38%	96.32%

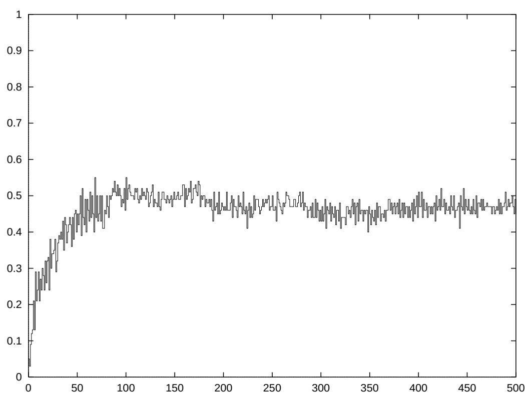
Table 5.2: Ultimate success rates for the learning systems on simulation case 1.



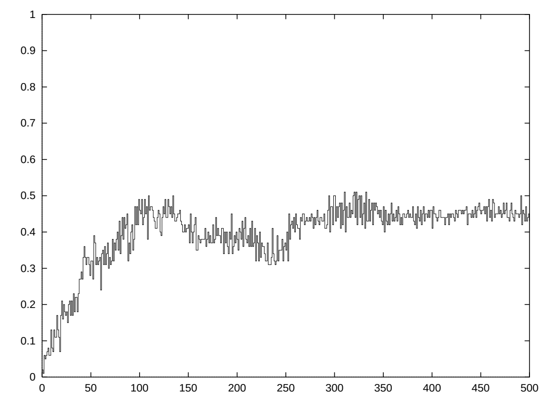
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



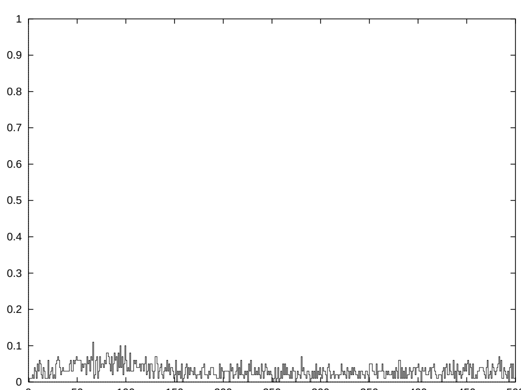
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



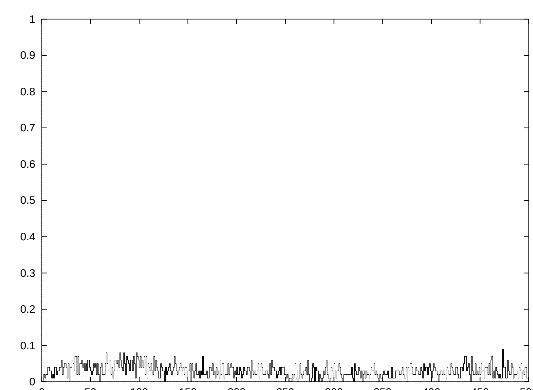
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

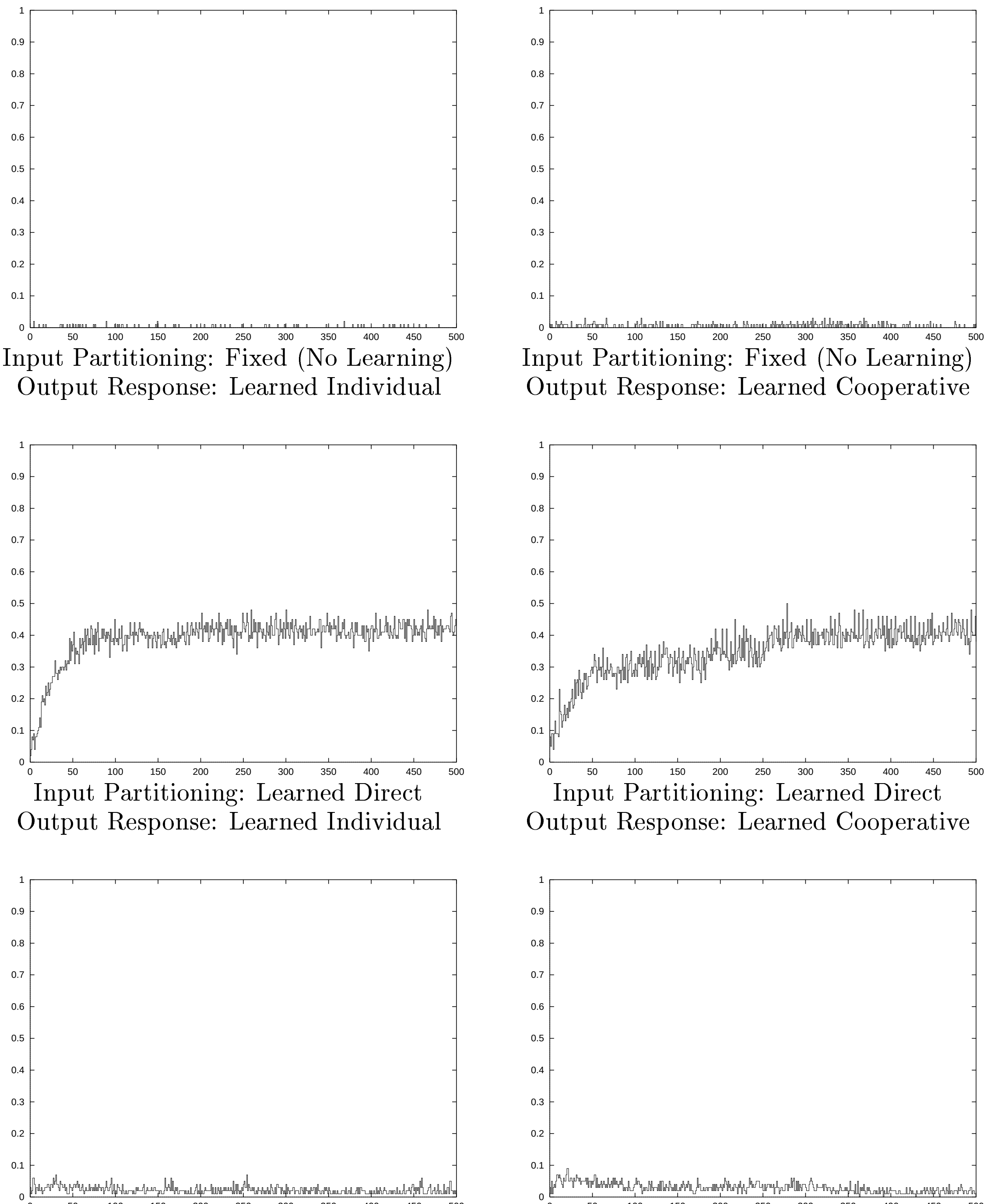


Input Partitioning: Learned Indexed
Output Response: Learned Individual



Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

Figure 5.3: Experiment Set 1. Truck-Backing with Small Angles. Dimensionality: 2. Neural units per dimension: 2. Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.



Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual

Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative

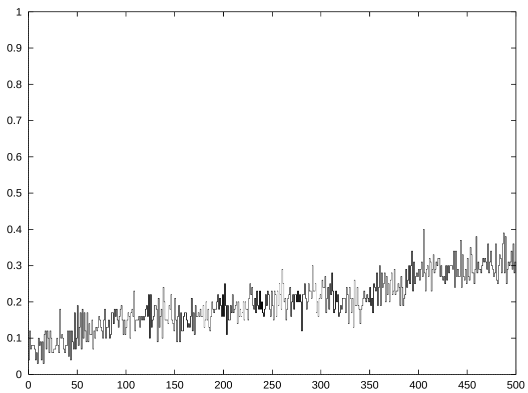
Input Partitioning: Learned Direct
Output Response: Learned Individual

Input Partitioning: Learned Direct
Output Response: Learned Cooperative

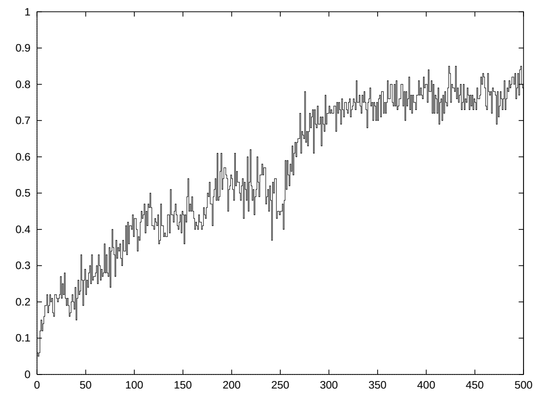
Input Partitioning: Learned Indexed
Output Response: Learned Individual

Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

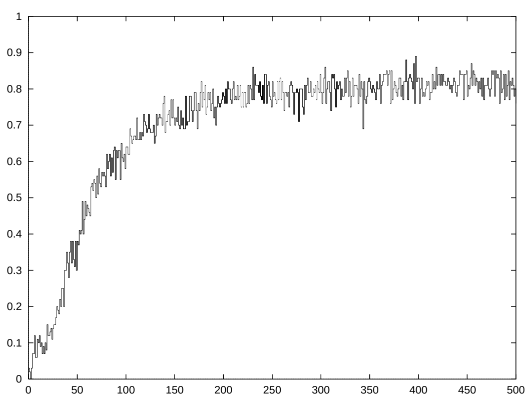
Figure 5.4: Experiment Set 2. Truck-Backing with Small Angles. Dimensionality: 2. Neural units per dimension: 3. Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.



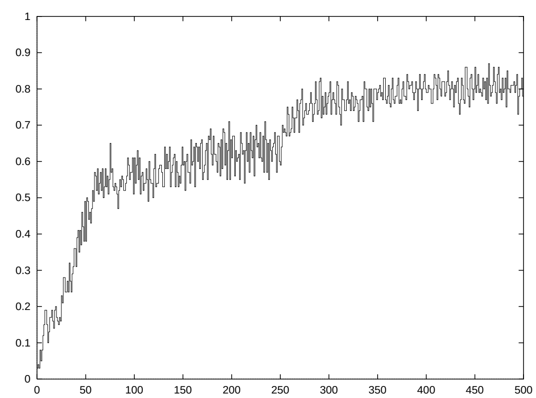
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



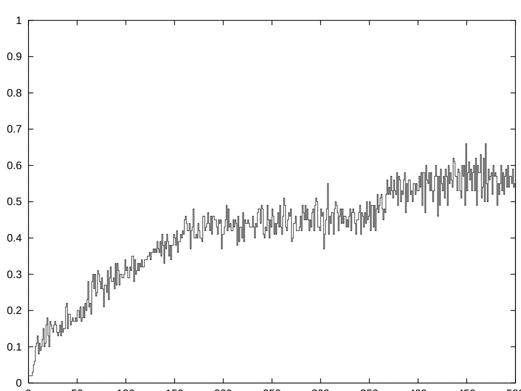
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



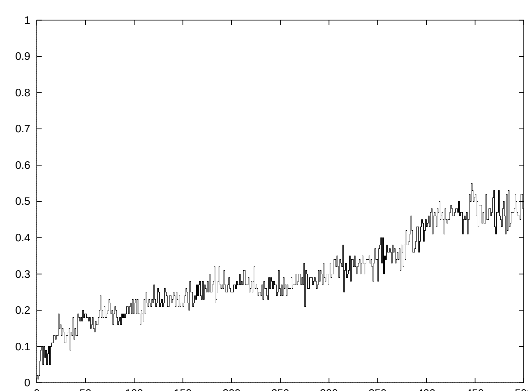
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

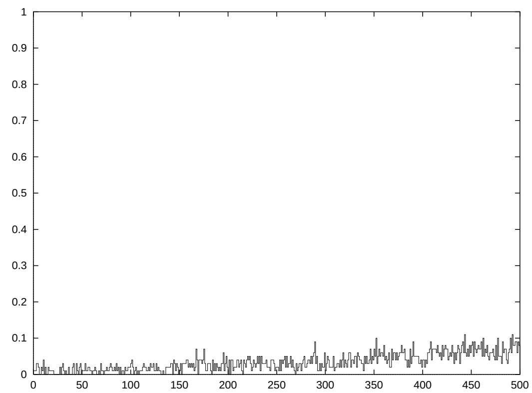


Input Partitioning: Learned Indexed
Output Response: Learned Individual

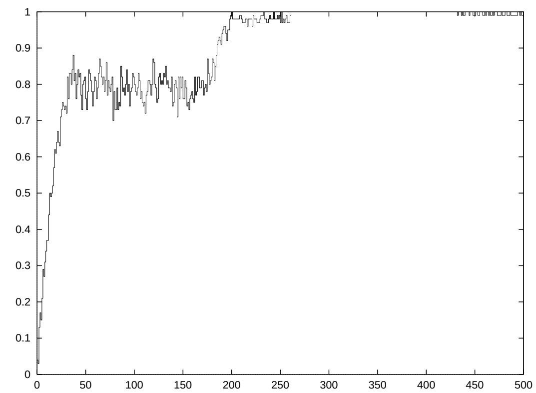


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

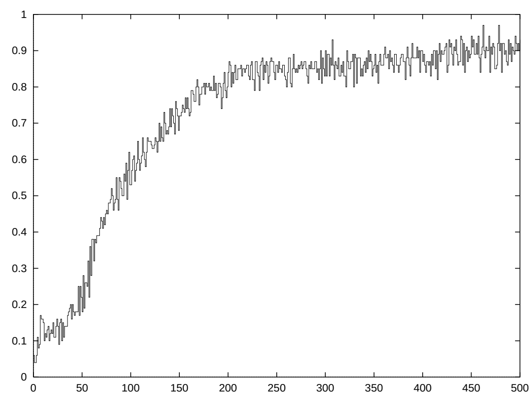
Figure 5.5: Experiment Set 3. Truck-Backing with Small Angles. Dimensionality: 2. Neural units per dimension: 4. Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.



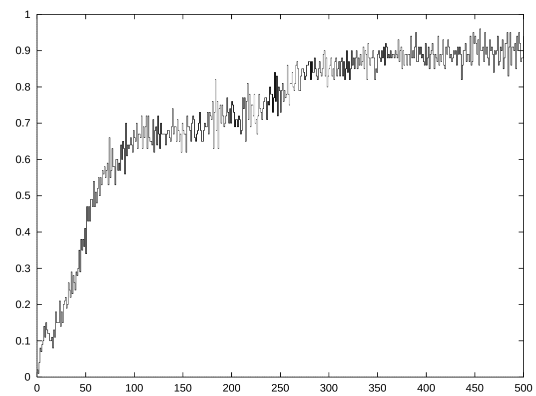
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



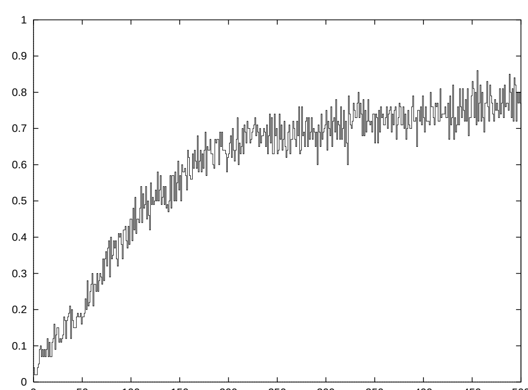
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



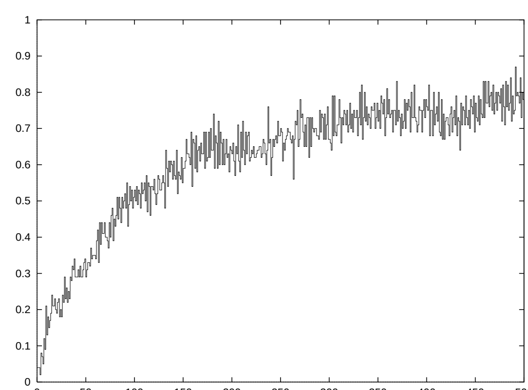
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

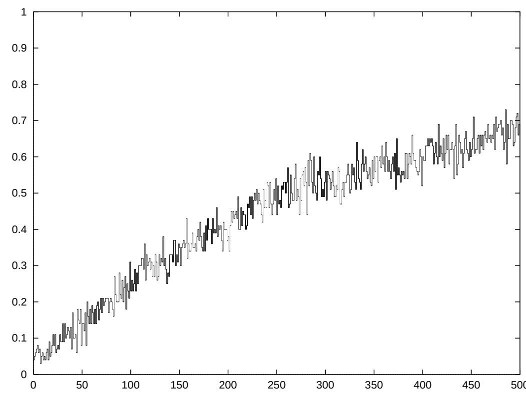


Input Partitioning: Learned Indexed
Output Response: Learned Individual

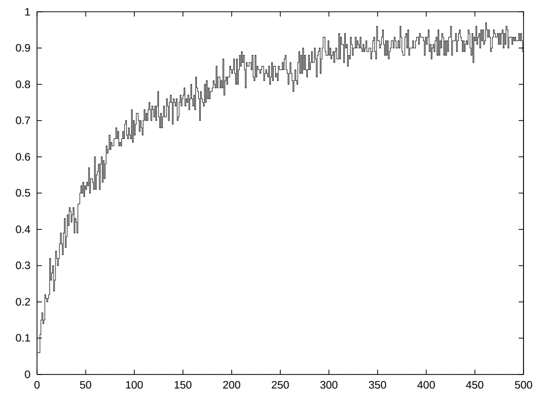


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

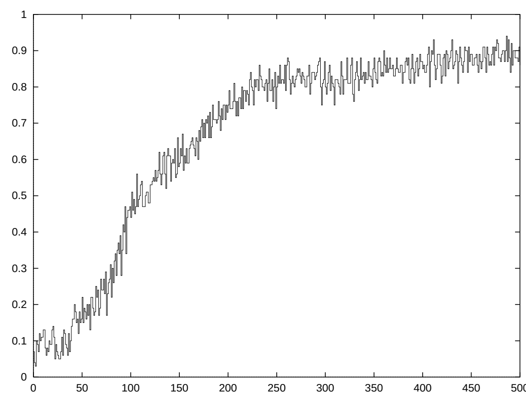
Figure 5.6: Experiment Set 4. Truck-Backing with Small Angles. Dimensionality: 2. Neural units per dimension: 5. Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.



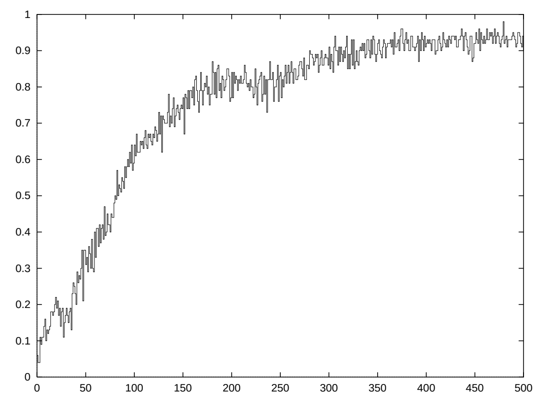
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



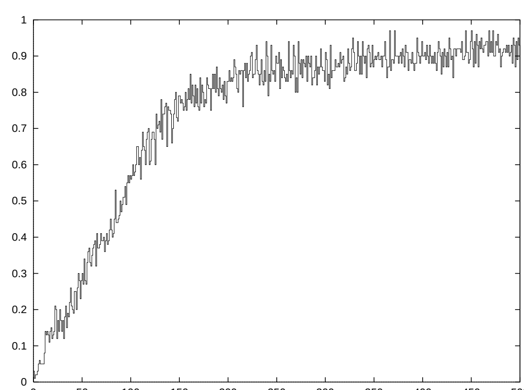
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



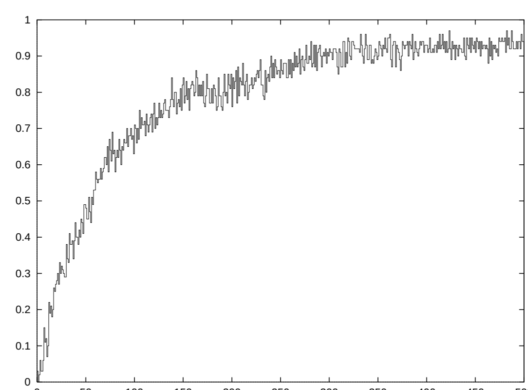
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

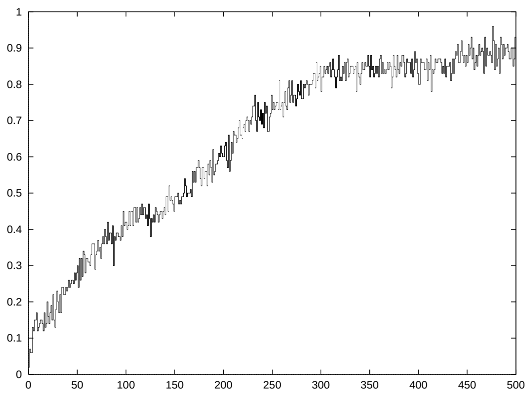


Input Partitioning: Learned Indexed
Output Response: Learned Individual

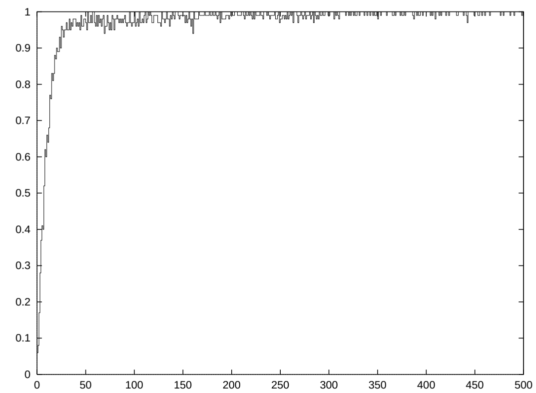


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

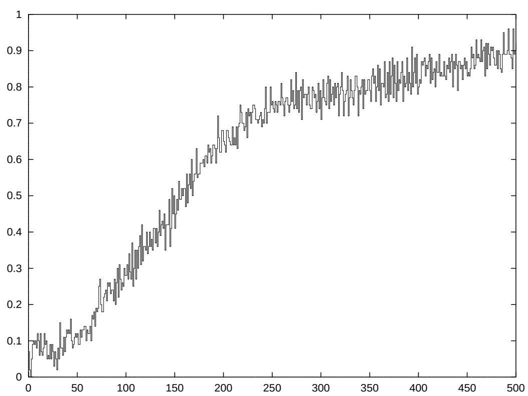
Figure 5.7: Experiment Set 5. Truck-Backing with Small Angles. Dimensionality: 2. Neural units per dimension: 6. Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.



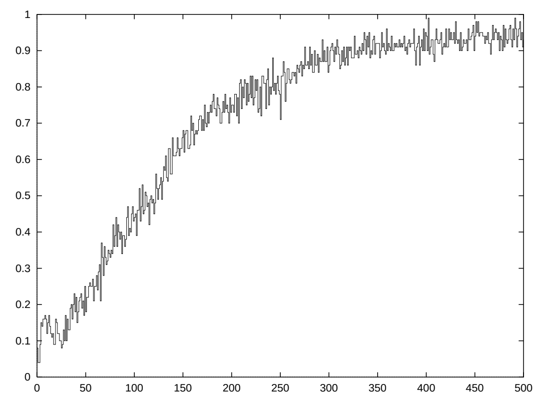
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



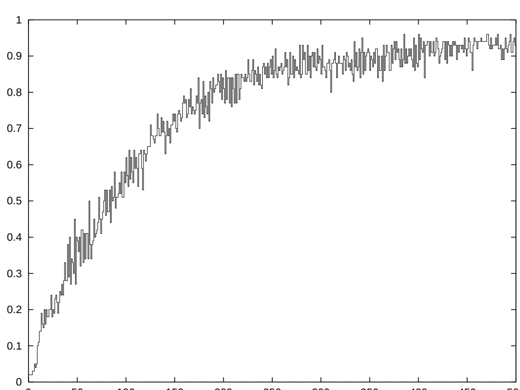
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



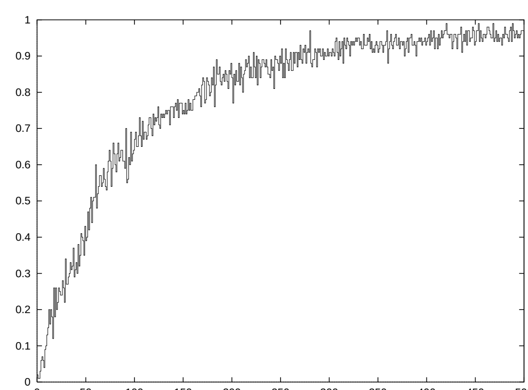
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

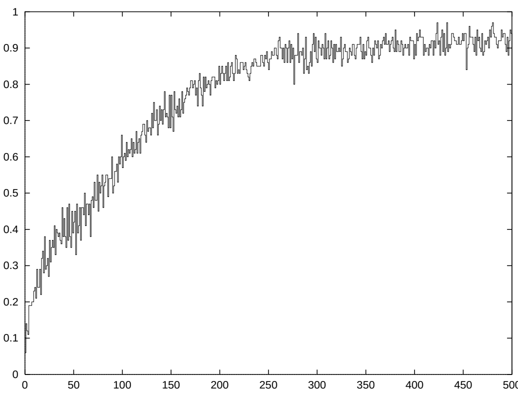


Input Partitioning: Learned Indexed
Output Response: Learned Individual

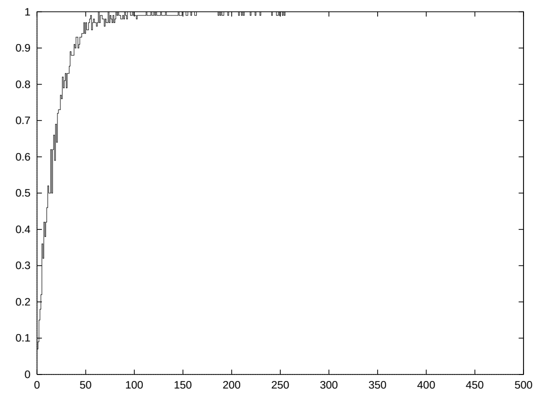


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

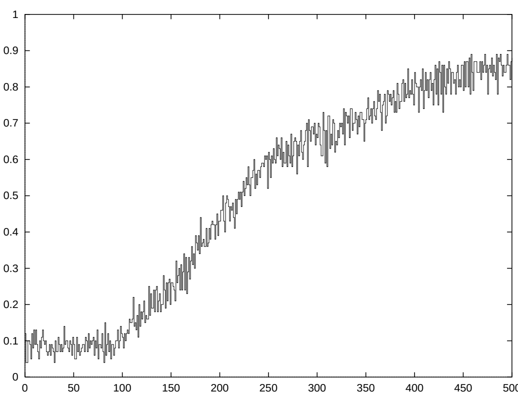
Figure 5.8: Experiment Set 6. Truck-Backing with Small Angles. Dimensionality: 2. Neural units per dimension: 7. Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.



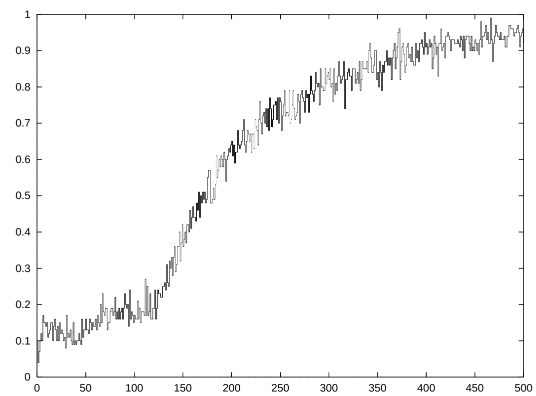
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



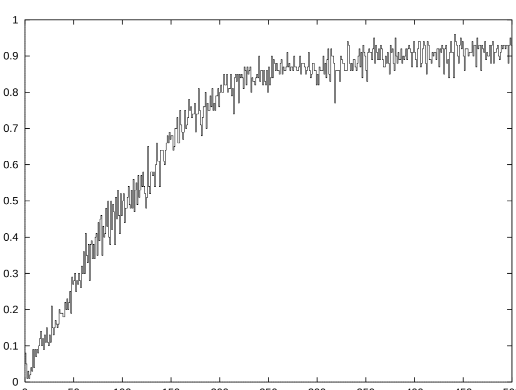
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



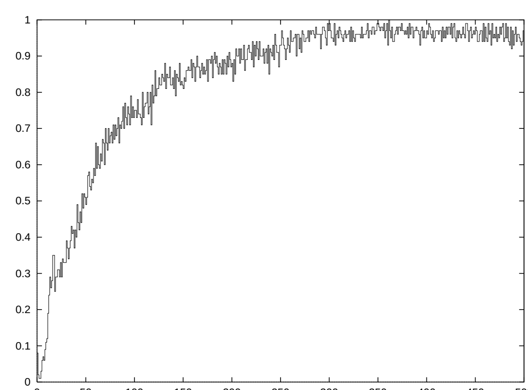
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

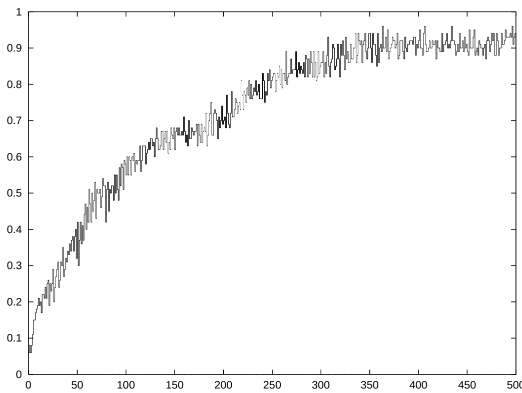


Input Partitioning: Learned Indexed
Output Response: Learned Individual

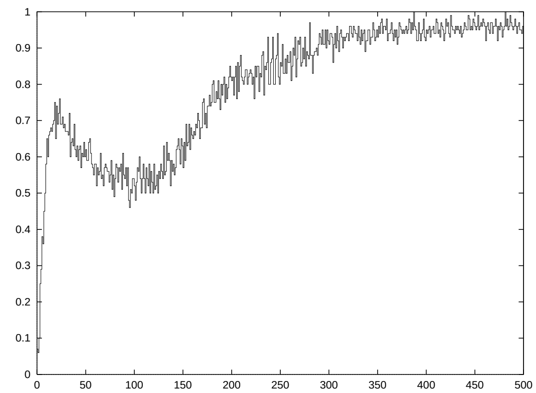


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

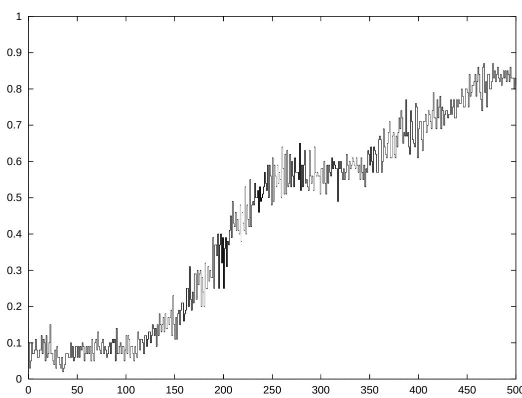
Figure 5.9: Experiment Set 7. Truck-Backing with Small Angles. Dimensionality: 2. Neural units per dimension: 8. Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.



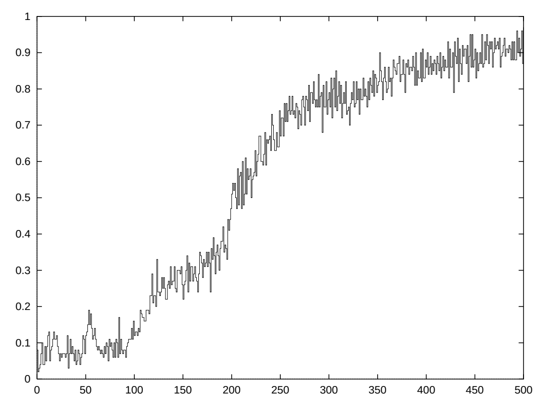
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



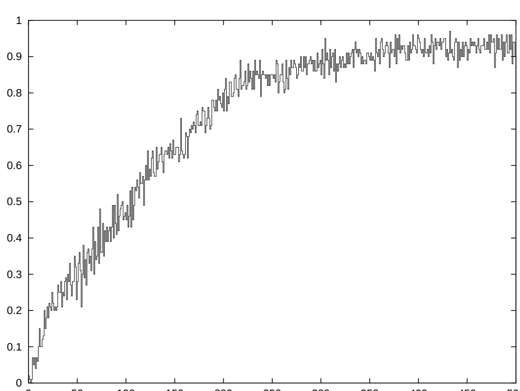
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



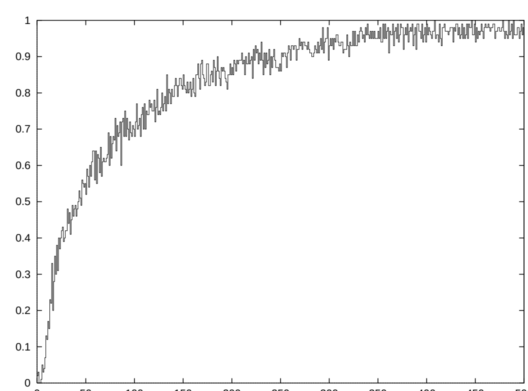
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

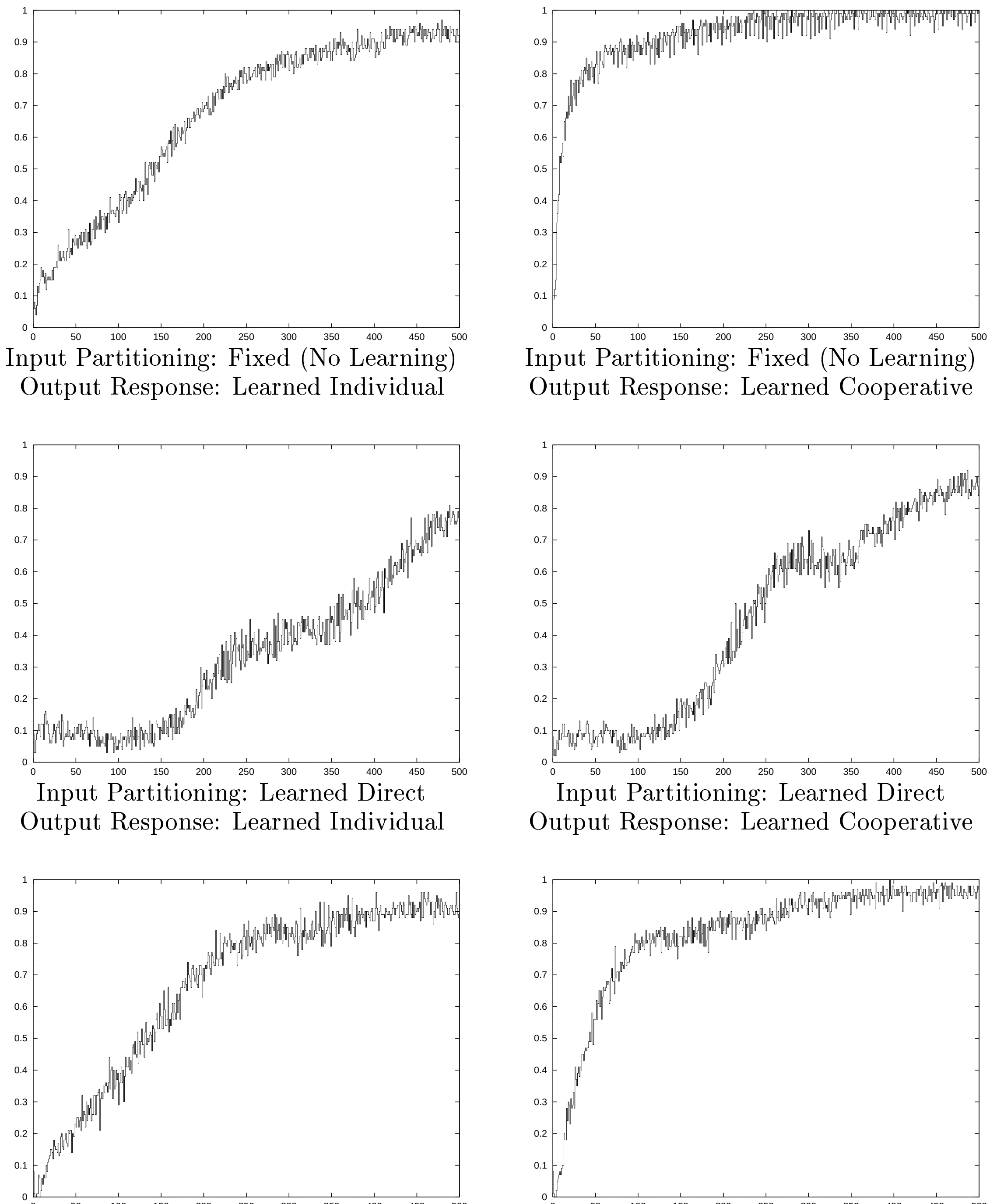


Input Partitioning: Learned Indexed
Output Response: Learned Individual



Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

Figure 5.10: Experiment Set 8. Truck-Backing with Small Angles. Dimensionality: 2. Neural units per dimension: 9. Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.



Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual

Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative

Input Partitioning: Learned Direct
Output Response: Learned Individual

Input Partitioning: Learned Direct
Output Response: Learned Cooperative

Input Partitioning: Learned Indexed
Output Response: Learned Individual

Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

Figure 5.11: Experiment Set 9. Truck-Backing with Small Angles. Dimensionality: 2. Neural units per dimension: 10. Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.

5.1.2 Truck-Backing with Large Angles

In the second case of truck-backing with a single trailer, the following parameters are used to determine the initial starting position of the rig:

- The rear of the trailer is started 6 feet from goal.
- The goal angle range is from $-180°$ to $+180°$.
- The hitch angle range is from $-65°$ to $+65°$.

New initial conditions are chosen randomly at the start of each trial with a uniform distribution over the entire range.

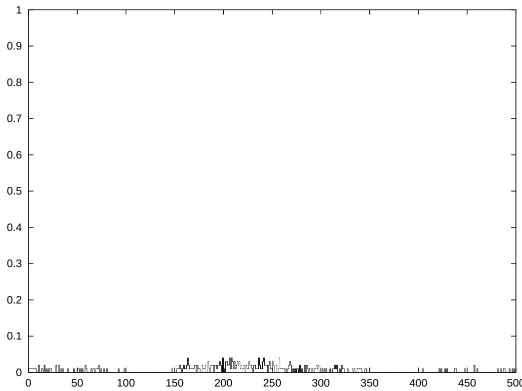
The basic parameters of the simulated rig are identical to those for truck backing with small angles (Subsection 5.1.1).

The boundary values for the hitch angle are $-90°$ and $+90°$; if the value of the hitch angle exceeds these boundaries, failure is signaled. There is no boundary value for the goal angle; the goal angle is measured from $-180°$ to $+180°$ with these two values considered identical. However, a second failure condition — a trial length in excess of 1000 time steps — is added. (See Subsection 4.3.2.)

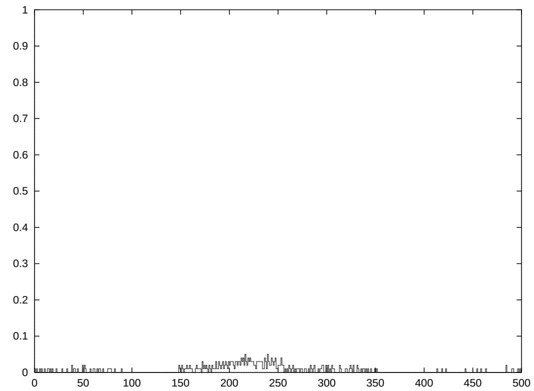
Each run is 500 trials in length and each graph is made by averaging 100 runs together.

Neural Units Per Dimension	Learning System Version: 1	2	3	4	5	6
2	0.20%	0.18%	3.44%	4.90%	4.88%	3.14%
3	0.00%	0.00%	14.46%	13.80%	36.18%	33.18%
4	0.56%	0.40%	28.12%	32.76%	64.48%	67.82%
5	1.12%	0.82%	34.12%	40.00%	57.26%	75.36%
6	35.56%	21.52%	53.05%	44.46%	67.32%	79.78%
7	62.58%	66.26%	63.80%	63.40%	64.86%	80.84%
8	62.44%	62.36%	32.44%	34.16%	60.24%	86.24%
9	59.20%	58.60%	22.94%	26.58%	66.14%	83.70%
10	40.58%	53.58%	10.42%	9.60%	59.72%	79.36%

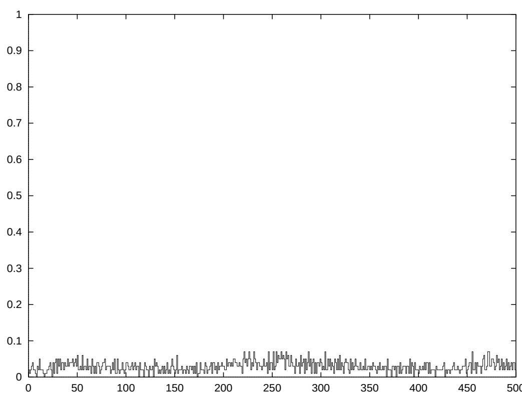
Table 5.3: Ultimate success rates for the learning systems on simulation case 2.



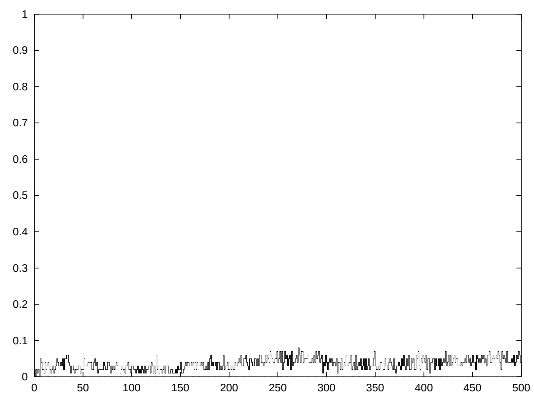
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



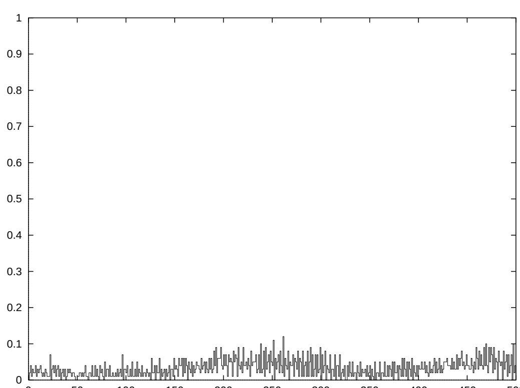
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



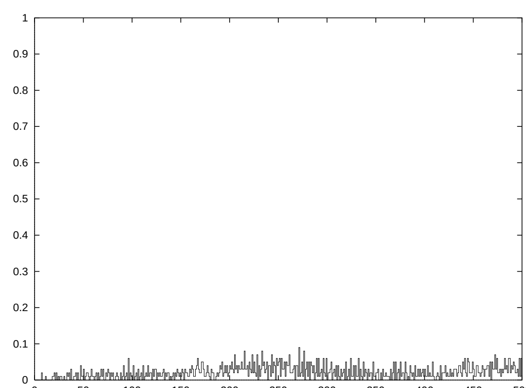
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

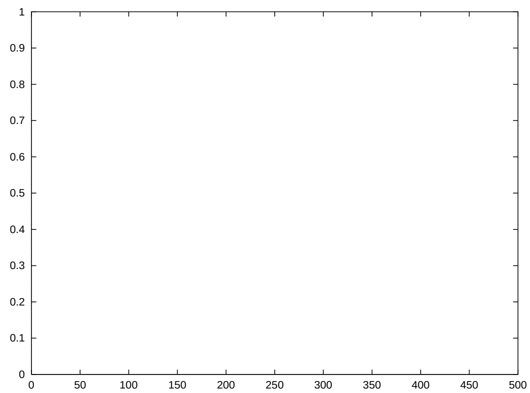


Input Partitioning: Learned Indexed
Output Response: Learned Individual

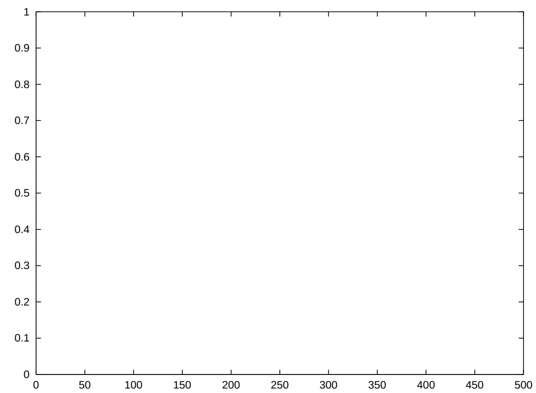


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

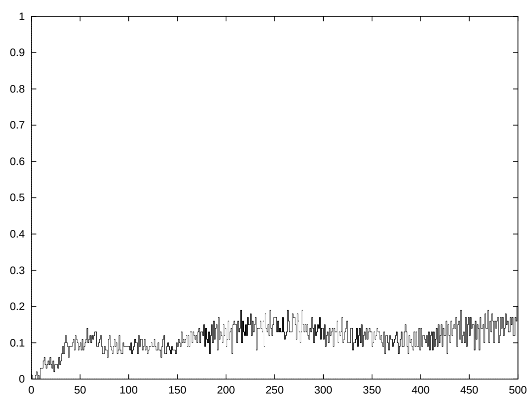
Figure 5.12: Experiment Set 10. Truck-Backing with Large Angles. Dimensionality: 2. Neural units per dimension: 2. Hitch Angle: $\pm 65^\circ$. Goal Angle: $\pm 180^\circ$.



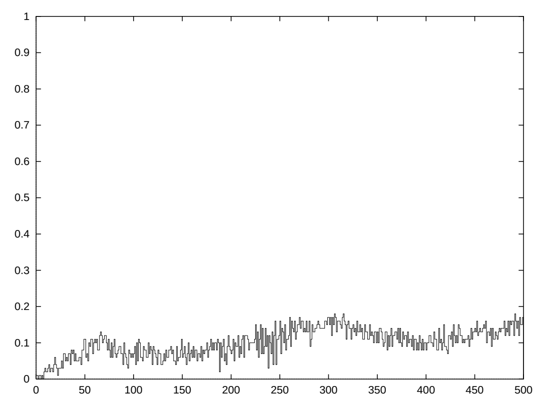
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



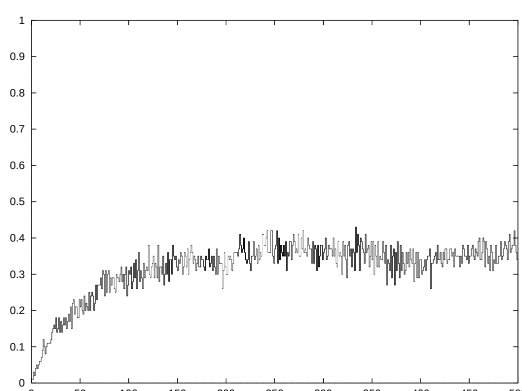
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



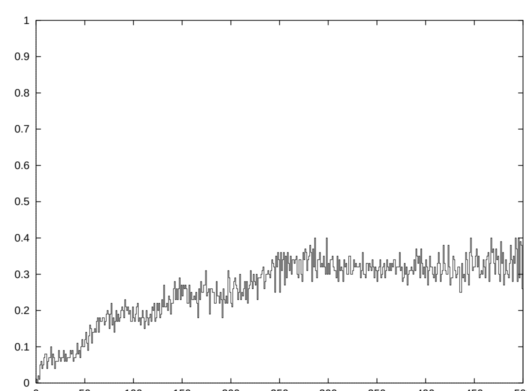
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

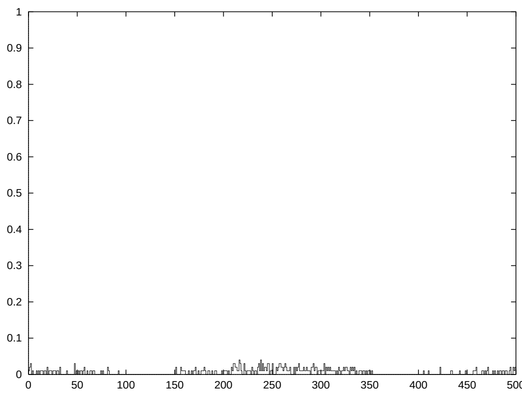


Input Partitioning: Learned Indexed
Output Response: Learned Individual

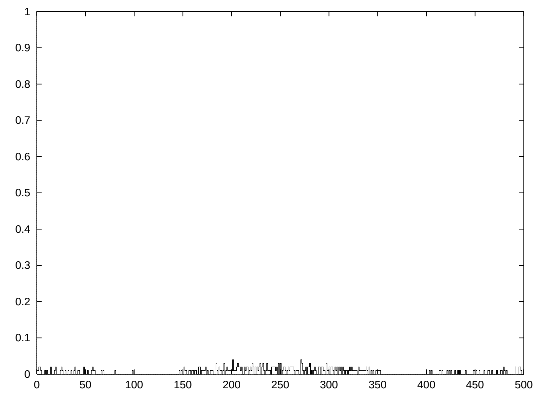


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

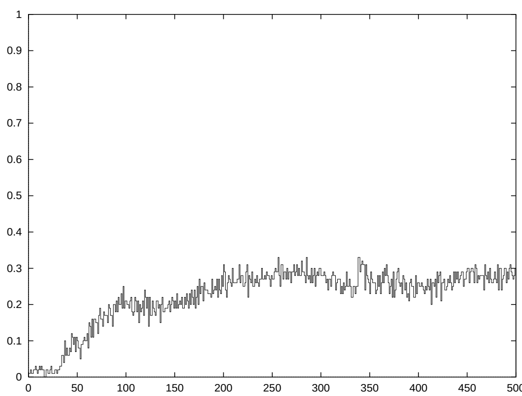
Figure 5.13: Experiment Set 11. Truck-Backing with Large Angles. Dimensionality: 2. Neural units per dimension: 3. Hitch Angle: $\pm 65°$. Goal Angle: $\pm 180°$.



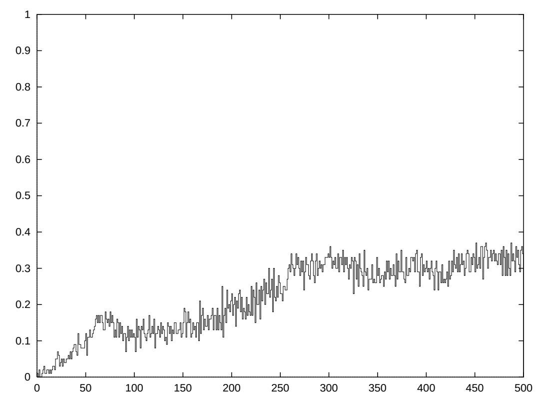
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



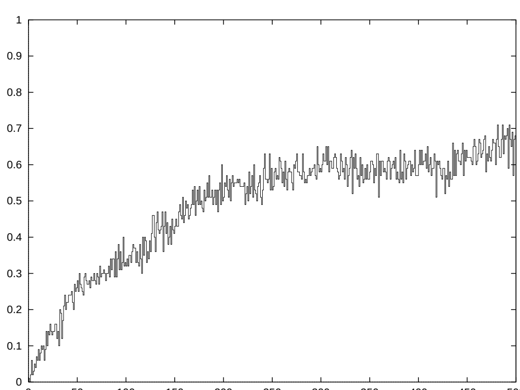
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



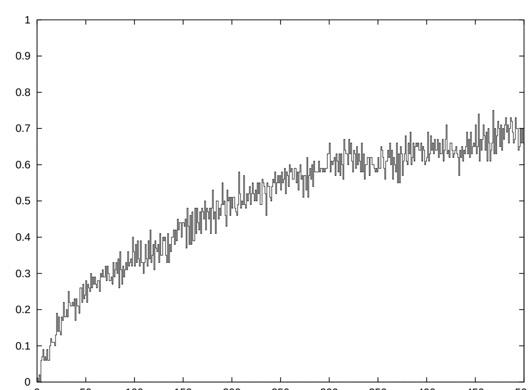
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

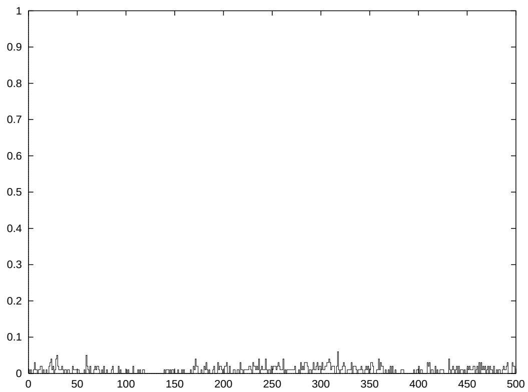


Input Partitioning: Learned Indexed
Output Response: Learned Individual

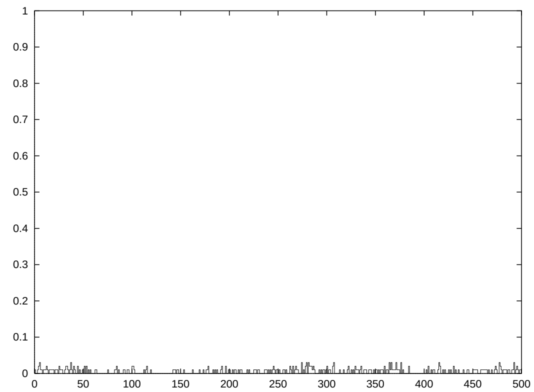


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

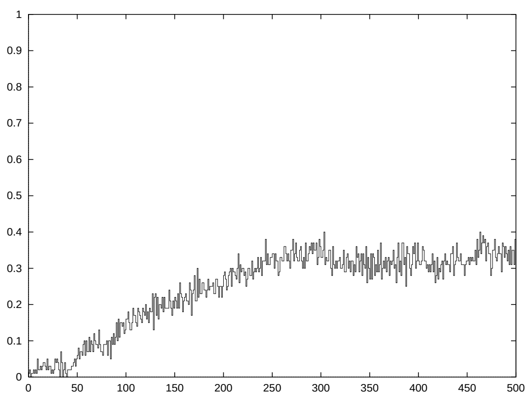
Figure 5.14: Experiment Set 12. Truck-Backing with Large Angles. Dimensionality: 2. Neural units per dimension: 4. Hitch Angle: $\pm 65°$. Goal Angle: $\pm 180°$.



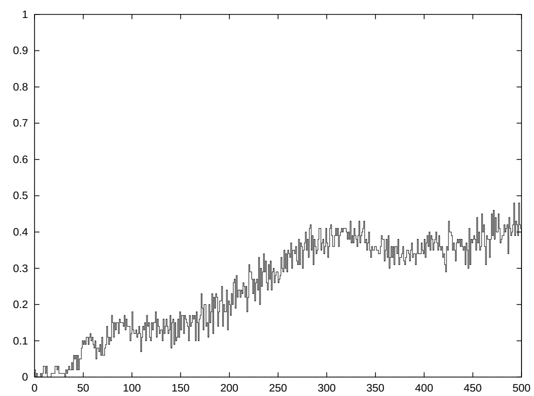
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



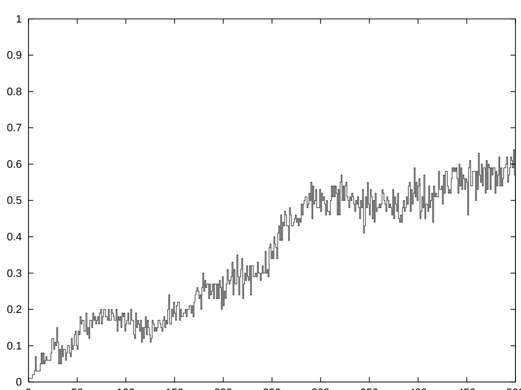
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



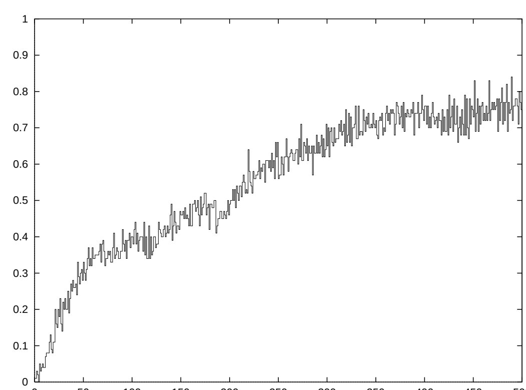
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

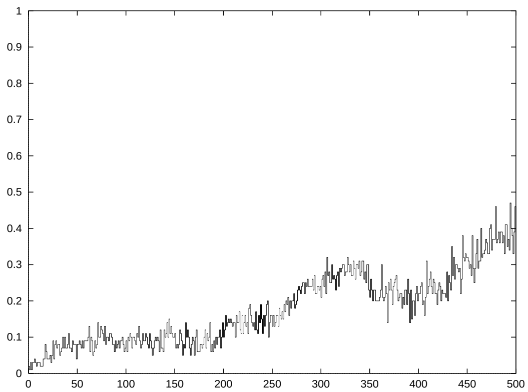


Input Partitioning: Learned Indexed
Output Response: Learned Individual

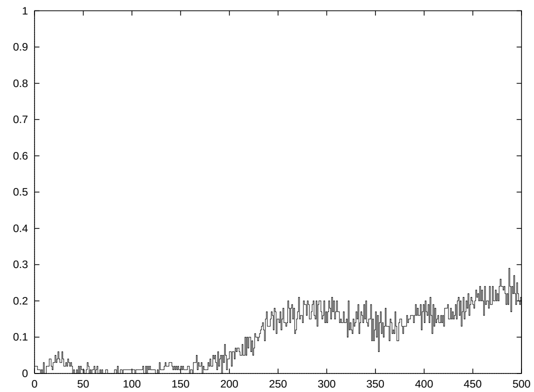


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

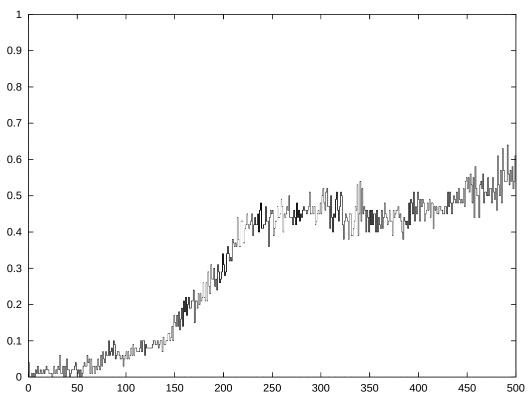
Figure 5.15: Experiment Set 13. Truck-Backing with Large Angles. Dimensionality: 2. Neural units per dimension: 5. Hitch Angle: $\pm 65^\circ$. Goal Angle: $\pm 180^\circ$.



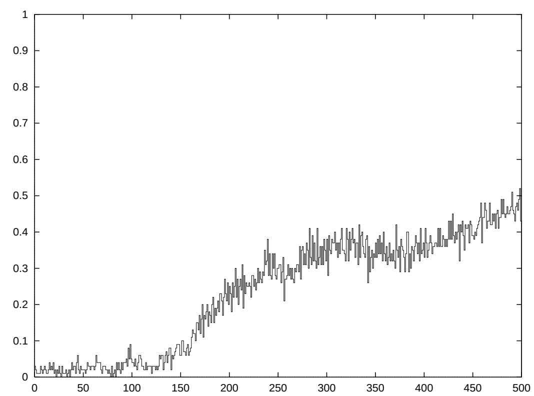
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



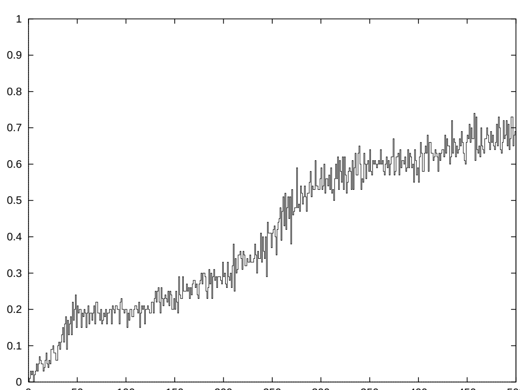
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



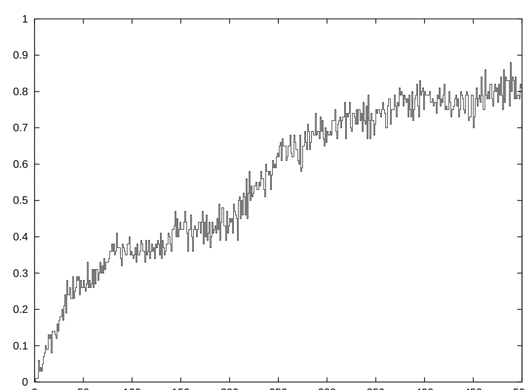
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

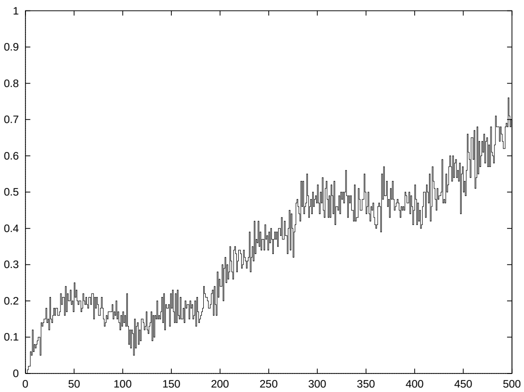


Input Partitioning: Learned Indexed
Output Response: Learned Individual

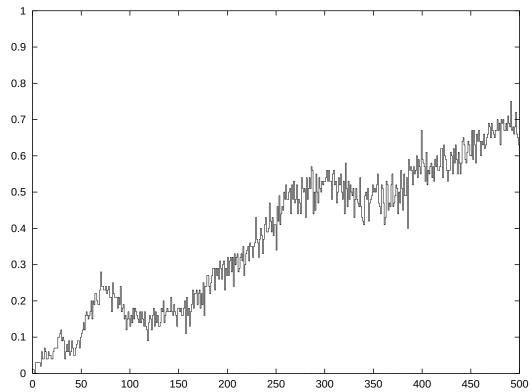


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

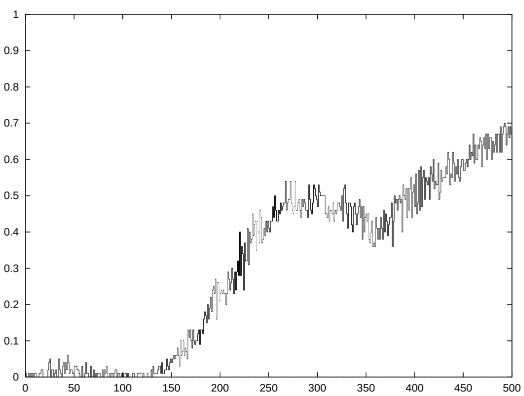
Figure 5.16: Experiment Set 14. Truck-Backing with Large Angles. Dimensionality: 2. Neural units per dimension: 6. Hitch Angle: $\pm 65°$. Goal Angle: $\pm 180°$.



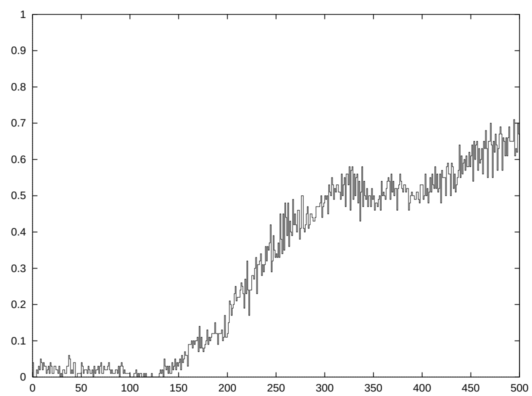
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



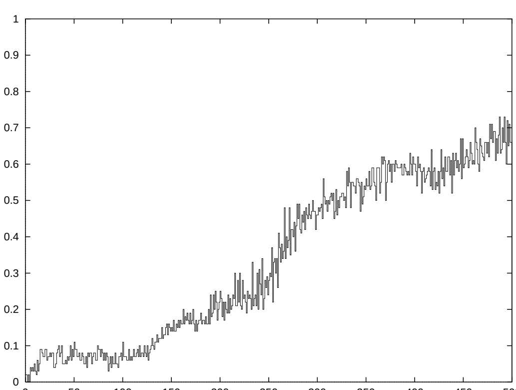
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



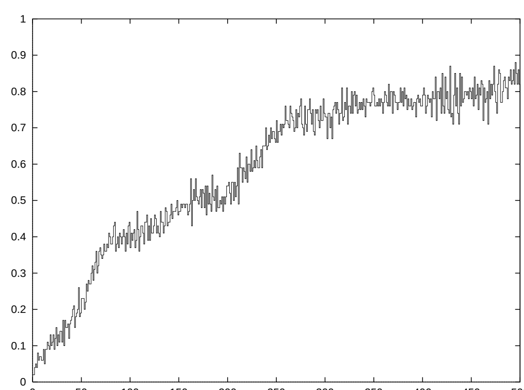
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

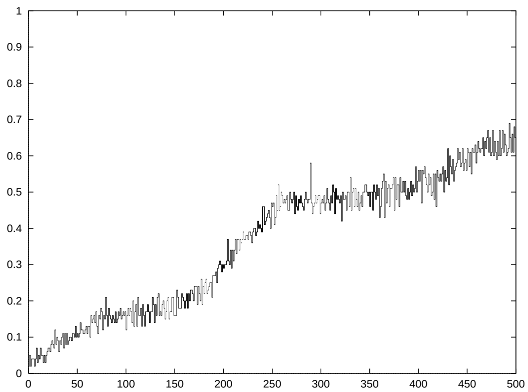


Input Partitioning: Learned Indexed
Output Response: Learned Individual

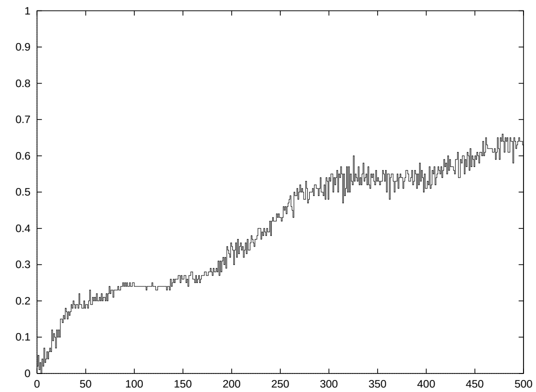


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

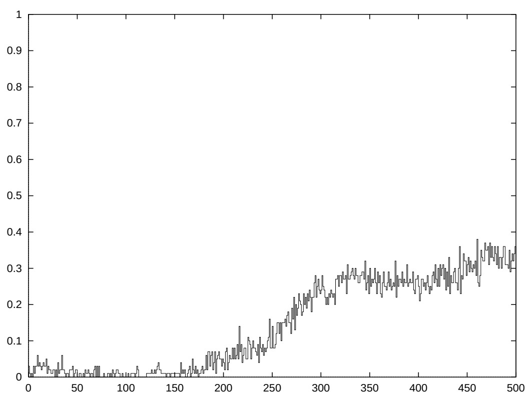
Figure 5.17: Experiment Set 15. Truck-Backing with Large Angles. Dimensionality: 2. Neural units per dimension: 7. Hitch Angle: $\pm 65°$. Goal Angle: $\pm 180°$.



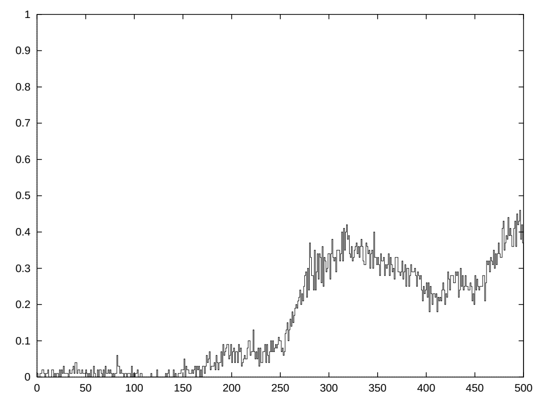
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



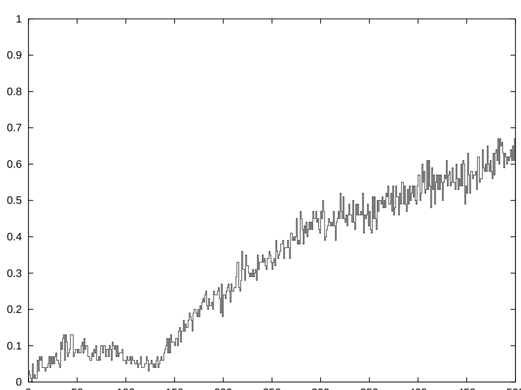
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



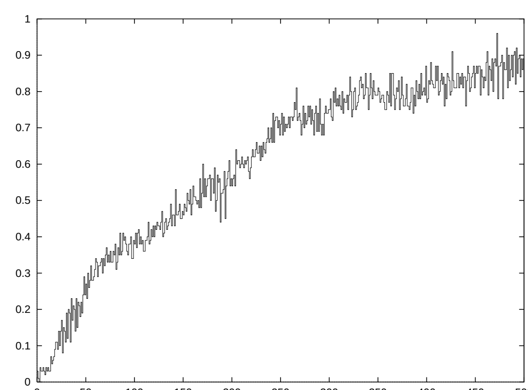
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

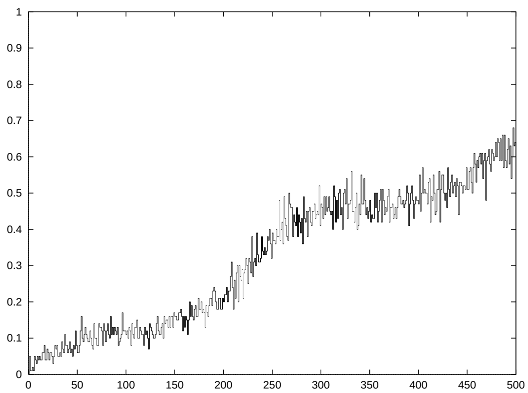


Input Partitioning: Learned Indexed
Output Response: Learned Individual

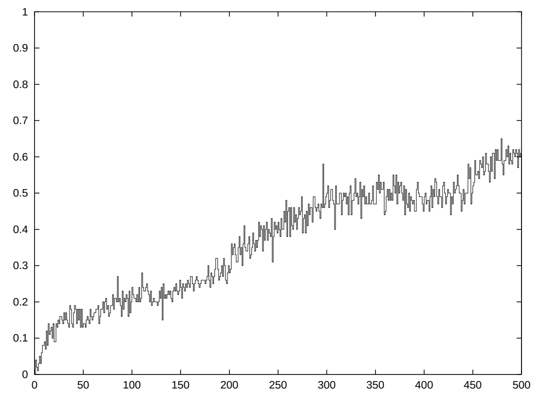


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

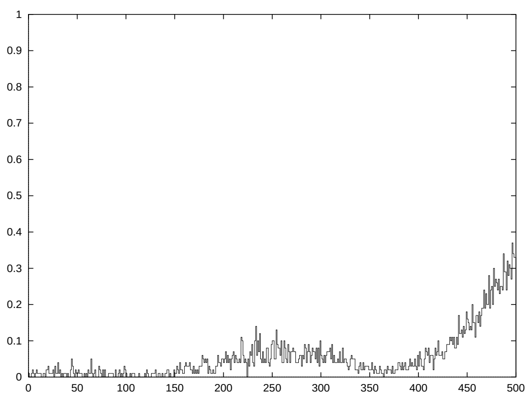
Figure 5.18: Experiment Set 16. Truck-Backing with Large Angles. Dimensionality: 2. Neural units per dimension: 8. Hitch Angle: $\pm 65°$. Goal Angle: $\pm 180°$.



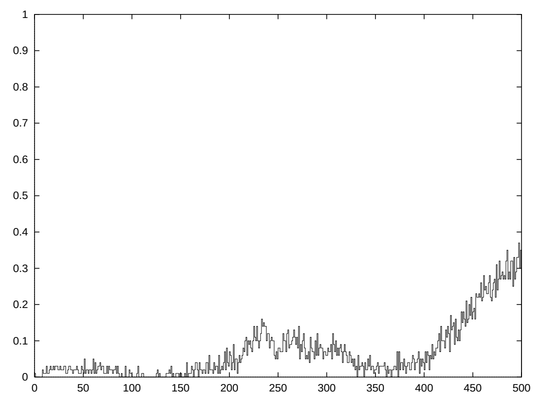
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



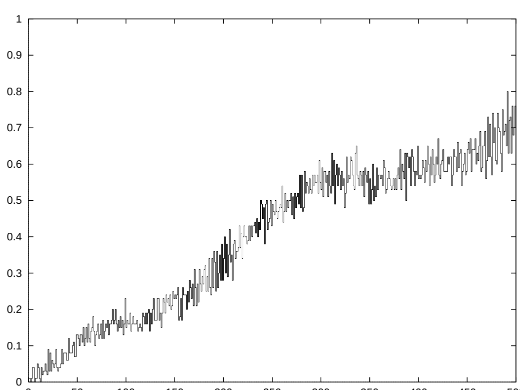
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



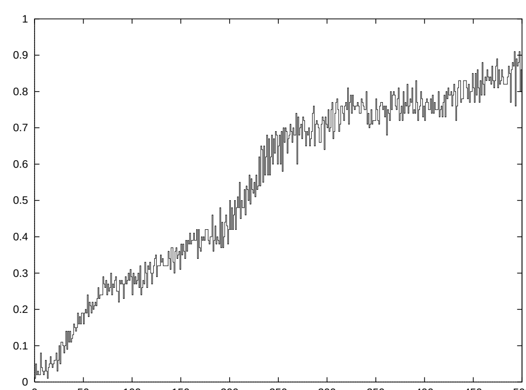
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative

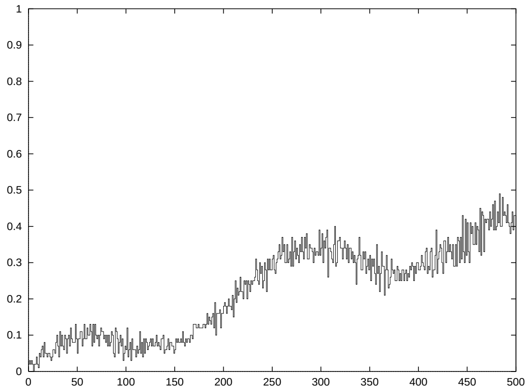


Input Partitioning: Learned Indexed
Output Response: Learned Individual

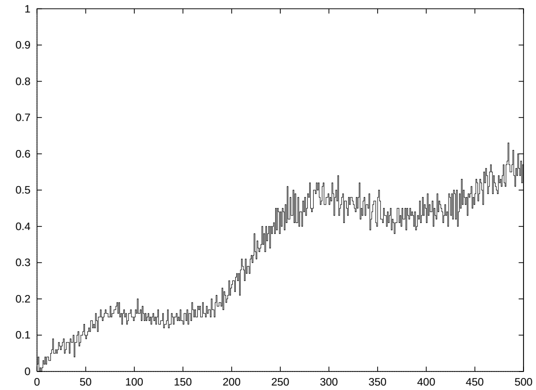


Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

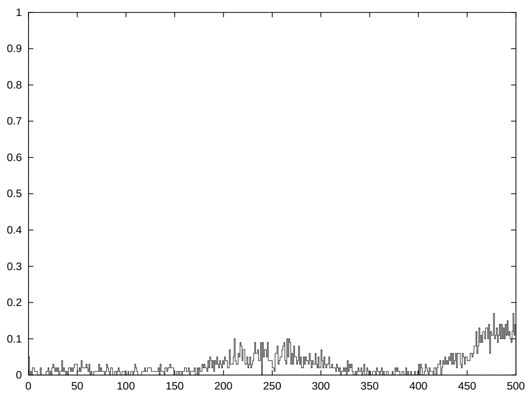
Figure 5.19: Experiment Set 17. Truck-Backing with Large Angles. Dimensionality: 2. Neural units per dimension: 9. Hitch Angle: $\pm 65°$. Goal Angle: $\pm 180°$.



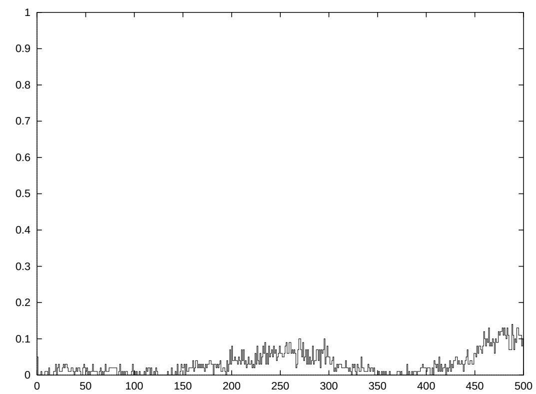
Input Partitioning: Fixed (No Learning)
Output Response: Learned Individual



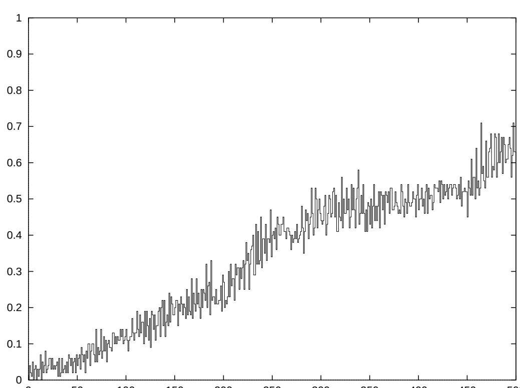
Input Partitioning: Fixed (No Learning)
Output Response: Learned Cooperative



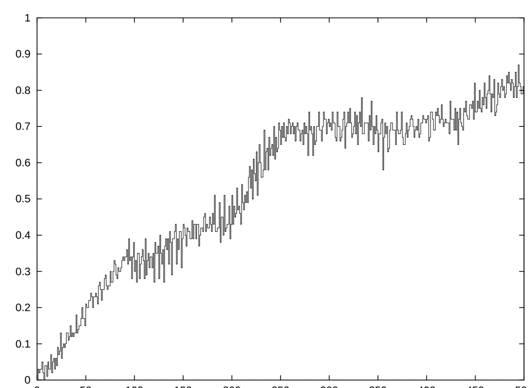
Input Partitioning: Learned Direct
Output Response: Learned Individual



Input Partitioning: Learned Direct
Output Response: Learned Cooperative



Input Partitioning: Learned Indexed
Output Response: Learned Individual



Input Partitioning: Learned Indexed
Output Response: Learned Cooperative

Figure 5.20: Experiment Set 18. Truck-Backing with Large Angles. Dimensionality: 2. Neural units per dimension: 10. Hitch Angle: $\pm 65°$. Goal Angle: $\pm 180°$.

5.1.3 Backing a Truck with Two Trailers

In the final simulation case, the task is complicated by adding a second trailer behind the first. Now the control system has three inputs:

1. The angle of hitch 1 between the truck and the first trailer.

2. The angle of hitch 2 between the first and second trailers.

3. The angle between the spine of the second trailer and the goal.

The addition of the second trailer adds one dimension to the input space of the problem when compared with the previous two cases. This is reflected in an additional dimension added to the networks of the learning systems used.

In all six learning system versions, the output network is now a three-dimensional cube, rather than a two-dimensional square as it was in the previous application cases. For learning system versions 1 through 4 (fixed and learned-direct partitionings) the input networks are also cubic. For learning system versions 5 and 6 (learned-indexed partitioning) there are three one-dimensional networks, rather than two.

The following parameters are used to determine the initial starting position of the rig:

- The rear of the second trailer is started from two to five feet from the target.

- The goal angle range is from $-6°$ to $+6°$.

- The range of the angle for hitch 1 is from $-6°$ to $+6°$.

- The range of the angle for hitch 2 is from $-6°$ to $+6°$.

New initial conditions are chosen randomly at the start of each trial with a uniform distribution over the entire range.

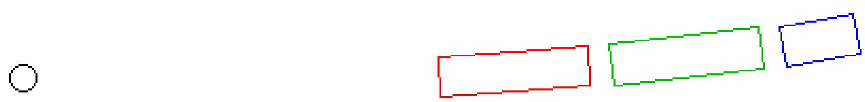
The basic parameters of the simulated rig are:

- The truck cab length is 8 inches.
- The length of trailer 1 is 16 inches.
- The length of trailer 2 is 16 inches.
- The distance moved by the front wheels of the truck on each time step is 0.25 inches.

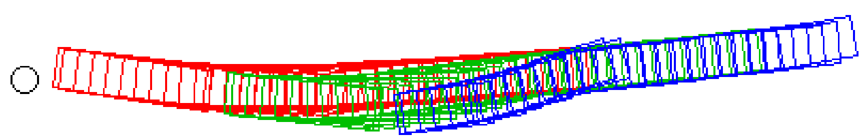
For further details of the simulated rig, see Appendix A.

Each run was 1000 trials in length and each graph is made by averaging 100 runs together. An example trial taken from near the end of one run is shown in Figure 5.21.

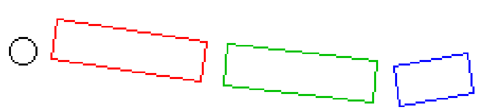
While for this case the maximum initial angles specified may seem small, it should be noted that the inherent instability in backing a rig with passive trailers means that, for a two trailer system like the one described, large differences in angles inevitably lead to a failure state, regardless of the control signal given. This is because, if the signs of the angles between adjacent units in the rig are opposite, then the active truck cab must swing out in arcs wider than those traced by the first trailer in order to move it around into position to correct the trajectory of the second trailer. However, in trying to bring the truck into such a position, the angle between the trailers and that between the second trailer and the goal are increased in magnitude as the entire rig backs up. In order for the system to be able to physically maneuver to the goal with larger possible initial angles, it would be necessary for the system to utilize forward motion, as well as backing.



A random starting position.



The movement of the rig, shown every fifteenth time step.

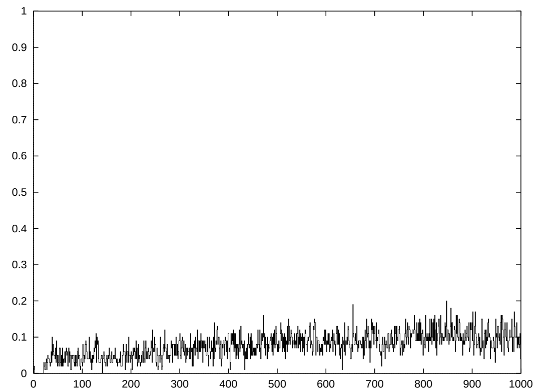


The final position of the rig at the goal.

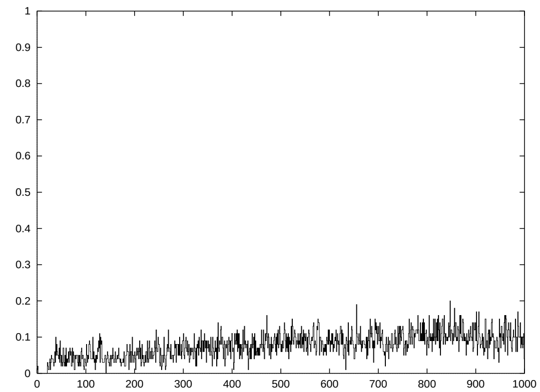
Figure 5.21: An example trial using the basic learning system with a uniform eight by eight partitioning. The trial was sampled from near the end of a run of 1000 trials.

Neural Units Per Dimension	Learning System Version 1	2	3	4	5	6
2	10.32%	10.32%	25.70%	28.46%	12.74%	25.84%
3	0.00%	0.00%	26.94%	29.72%	18.72%	5.84%
4	17.90%	18.82%	37.92%	47.54%	37.60%	45.32%
5	51.90%	62.42%	46.52%	69.84%	55.62%	72.28%
6	47.00%	92.28%	37.58%	72.50%	73.12%	78.76%
7	48.08%	65.40%	39.06%	66.60%	70.94%	77.02%
8	47.78%	95.10%	33.86%	48.66%	69.60%	77.66%
9	50.68%	64.50%	33.74%	40.10%	67.90%	77.96%
10	42.40%	60.12%	34.72%	35.74%	64.28%	75.62%

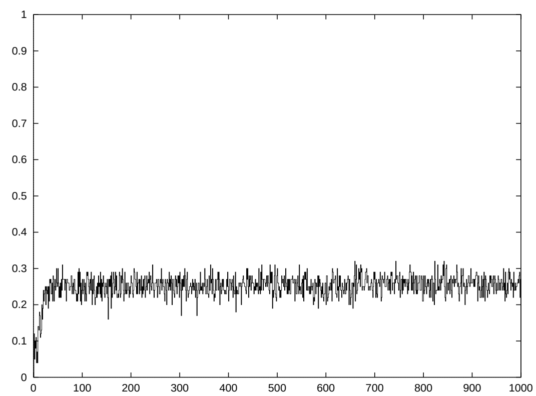
Table 5.4: Ultimate success rates for the learning systems on simulation case 3.



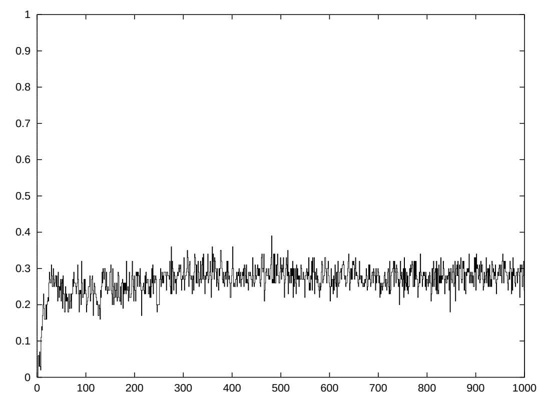
Input Partitioning: Fixed (No Learning)
Response: Learned Individual



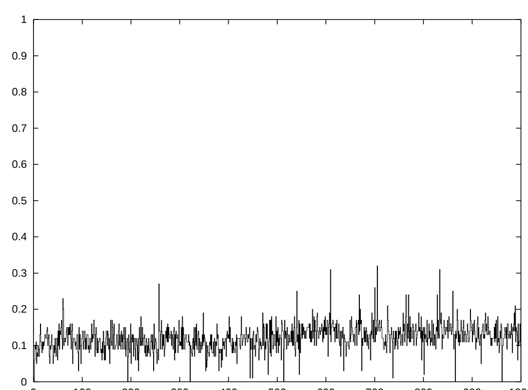
Input Partitioning: Fixed (No Learning)
Response: Learned Cooperative



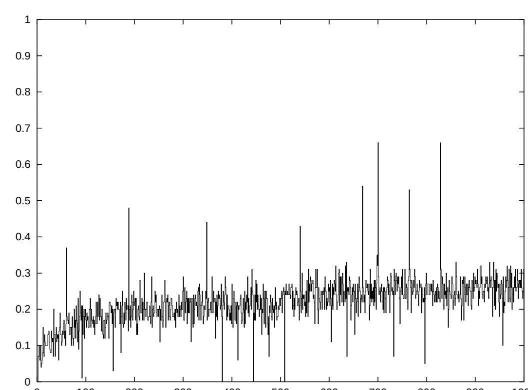
Input Partitioning: Learned Direct
Response: Learned Individual



Input Partitioning: Learned Direct
Response: Learned Cooperative

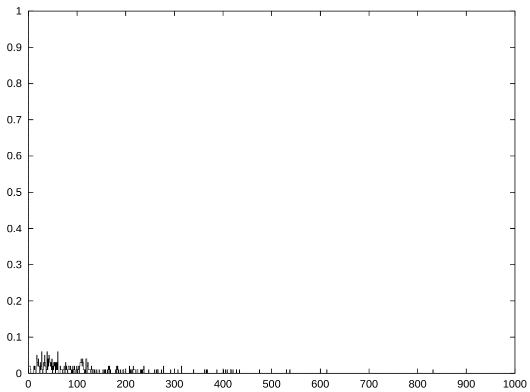


Input Partitioning: Learned Indexed
Response: Learned Individual

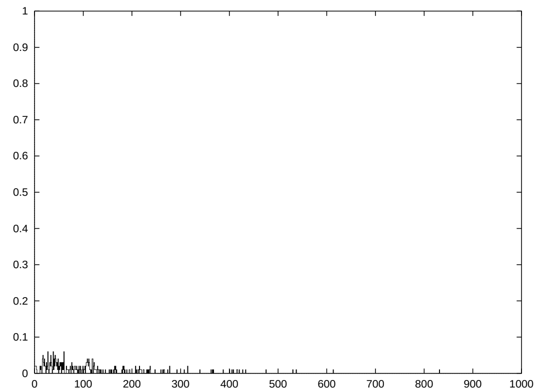


Input Partitioning: Learned Indexed
Response: Learned Cooperative

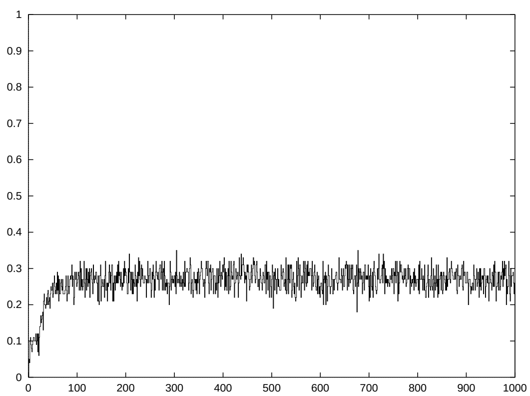
Figure 5.22: Experiment Set 19. Backing a Truck with Two Trailers. Dimensionality: 3. Neural units per dimension: 2. Hitch One Angle: $\pm 5^\circ$. Hitch Two Angle: $\pm 5^\circ$. Goal Angle: $\pm 5^\circ$.



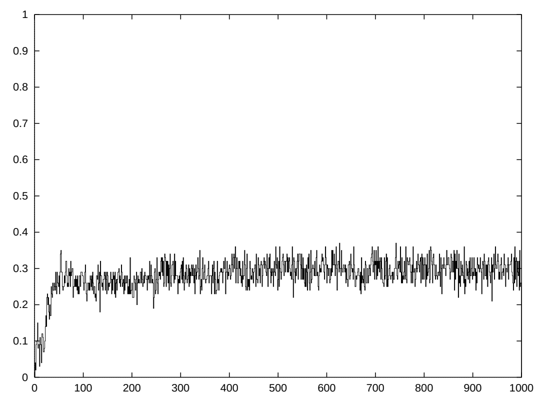
Input Partitioning: Fixed (No Learning)
Response: Learned Individual



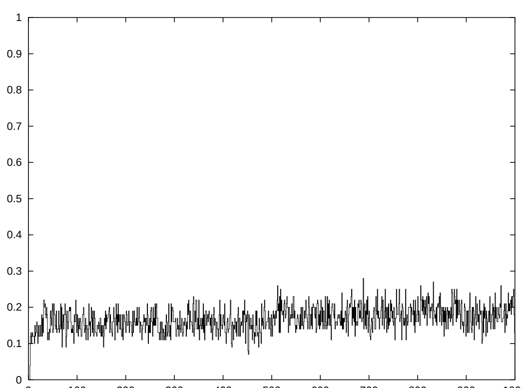
Input Partitioning: Fixed (No Learning)
Response: Learned Cooperative



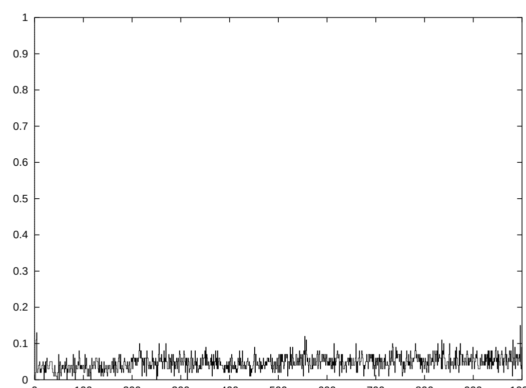
Input Partitioning: Learned Direct
Response: Learned Individual



Input Partitioning: Learned Direct
Response: Learned Cooperative

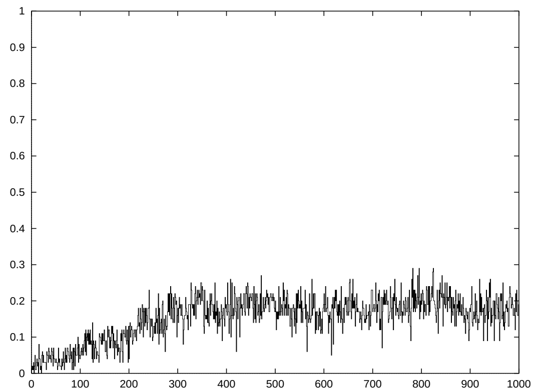


Input Partitioning: Learned Indexed
Response: Learned Individual

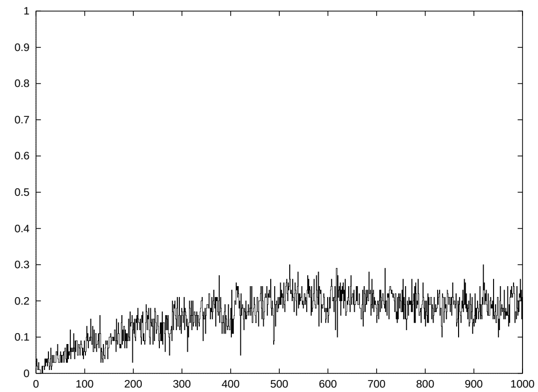


Input Partitioning: Learned Indexed
Response: Learned Cooperative

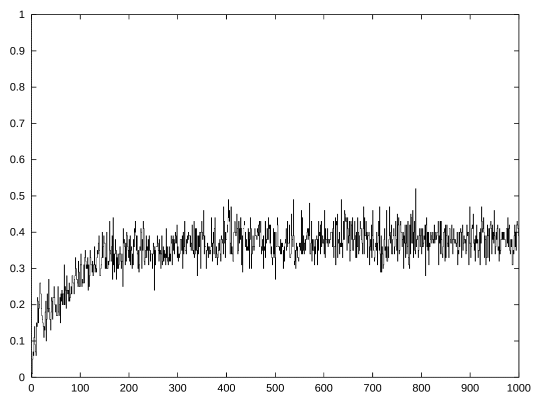
Figure 5.23: Experiment Set 20. Backing a Truck with Two Trailers. Dimensionality: 3. Neural units per dimension: 3. Hitch One Angle: $\pm 5^\circ$. Hitch Two Angle: $\pm 5^\circ$. Goal Angle: $\pm 5^\circ$.



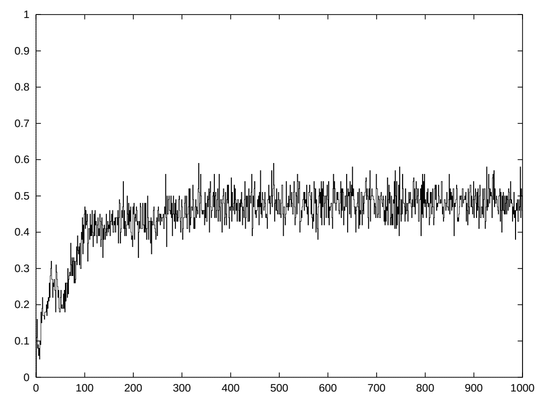
Input Partitioning: Fixed (No Learning)
Response: Learned Individual



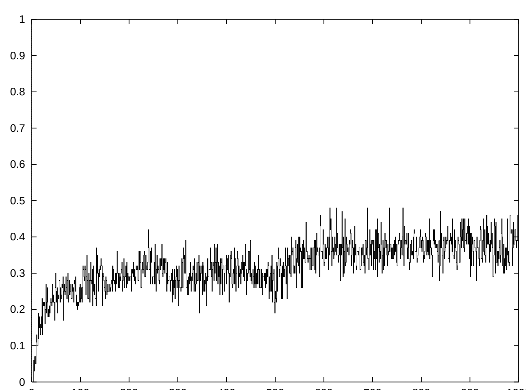
Input Partitioning: Fixed (No Learning)
Response: Learned Cooperative



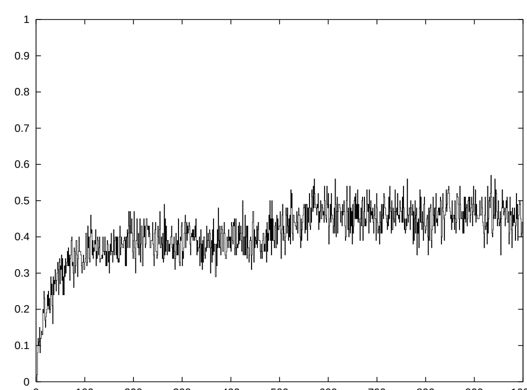
Input Partitioning: Learned Direct
Response: Learned Individual



Input Partitioning: Learned Direct
Response: Learned Cooperative

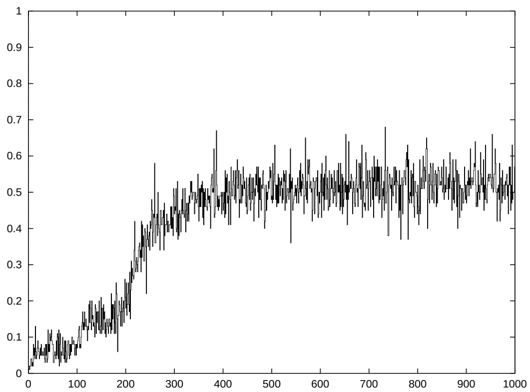


Input Partitioning: Learned Indexed
Response: Learned Individual

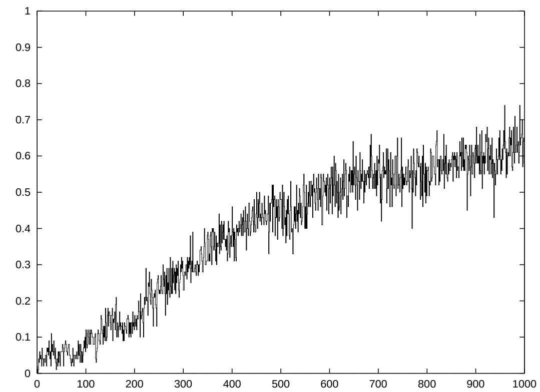


Input Partitioning: Learned Indexed
Response: Learned Cooperative

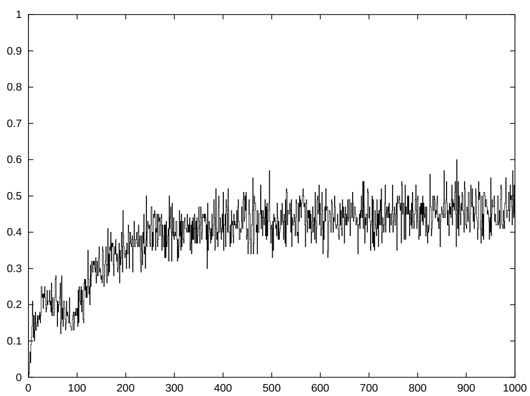
Figure 5.24: Experiment Set 21. Backing a Truck with Two Trailers. Dimensionality: 3. Neural units per dimension: 4. Hitch One Angle: $\pm 5°$. Hitch Two Angle: $\pm 5°$. Goal Angle: $\pm 5°$.



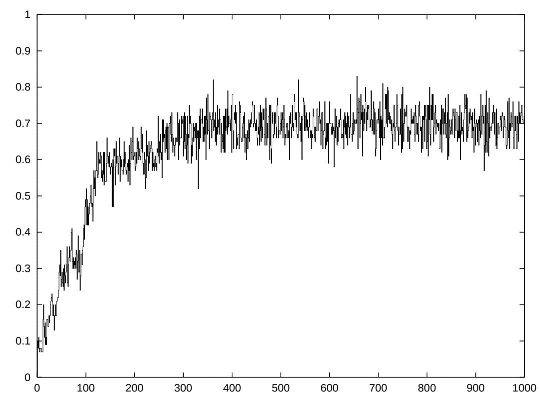
Input Partitioning: Fixed (No Learning)
Response: Learned Individual



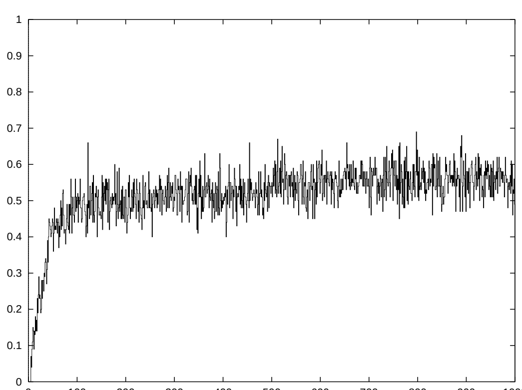
Input Partitioning: Fixed (No Learning)
Response: Learned Cooperative



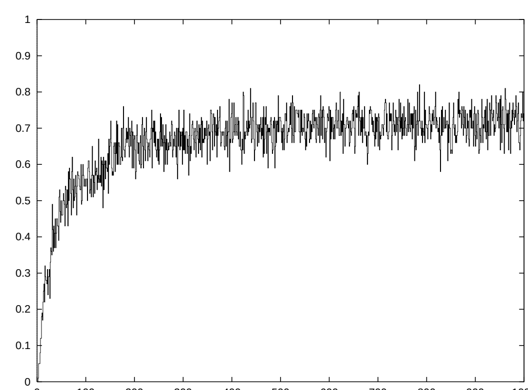
Input Partitioning: Learned Direct
Response: Learned Individual



Input Partitioning: Learned Direct
Response: Learned Cooperative

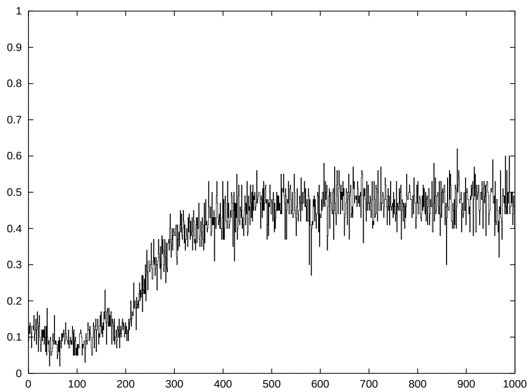


Input Partitioning: Learned Indexed
Response: Learned Individual

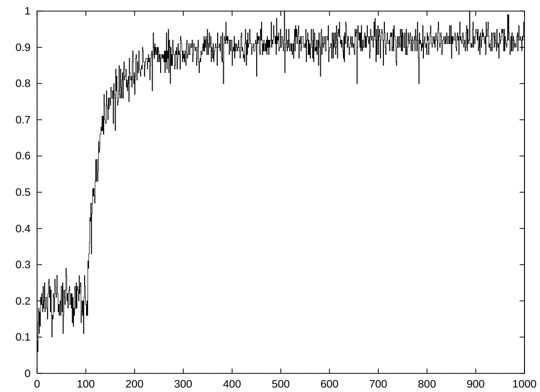


Input Partitioning: Learned Indexed
Response: Learned Cooperative

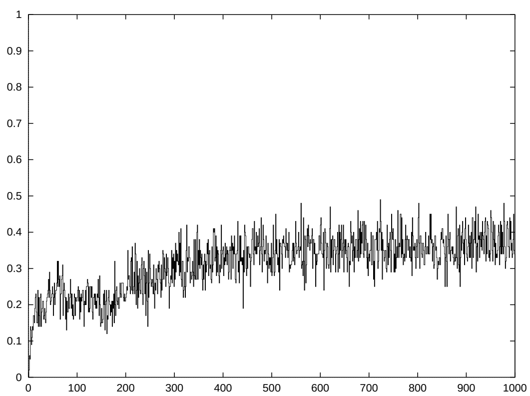
Figure 5.25: Experiment Set 22. Backing a Truck with Two Trailers. Dimensionality: 3. Neural units per dimension: 5. Hitch One Angle: $\pm 5°$. Hitch Two Angle: $\pm 5°$. Goal Angle: $\pm 5°$.



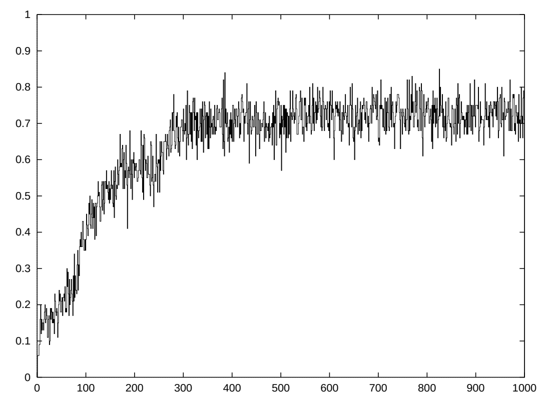
Input Partitioning: Fixed (No Learning)
Response: Learned Individual



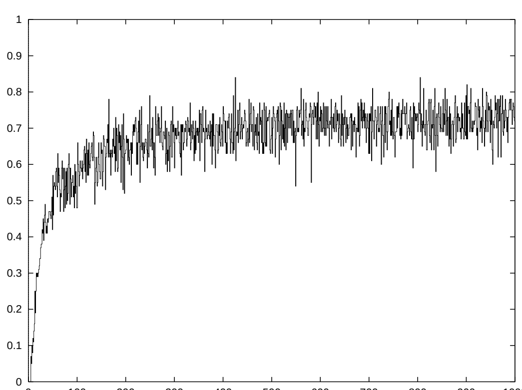
Input Partitioning: Fixed (No Learning)
Response: Learned Cooperative



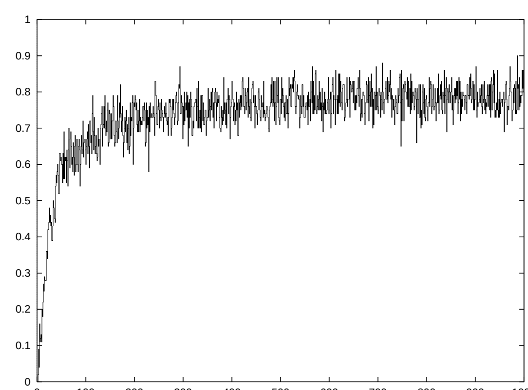
Input Partitioning: Learned Direct
Response: Learned Individual



Input Partitioning: Learned Direct
Response: Learned Cooperative

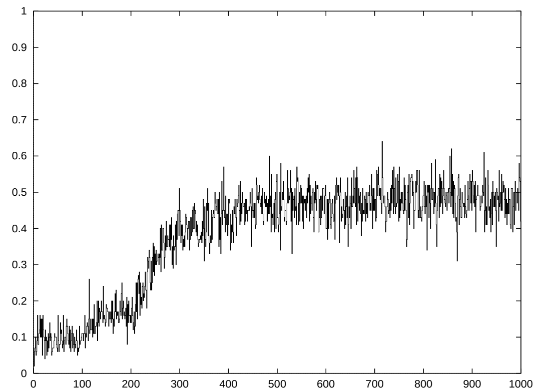


Input Partitioning: Learned Indexed
Response: Learned Individual

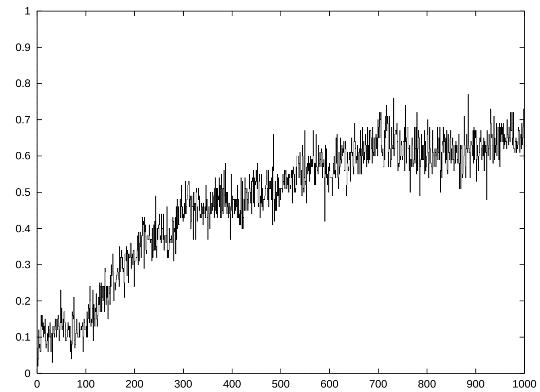


Input Partitioning: Learned Indexed
Response: Learned Cooperative

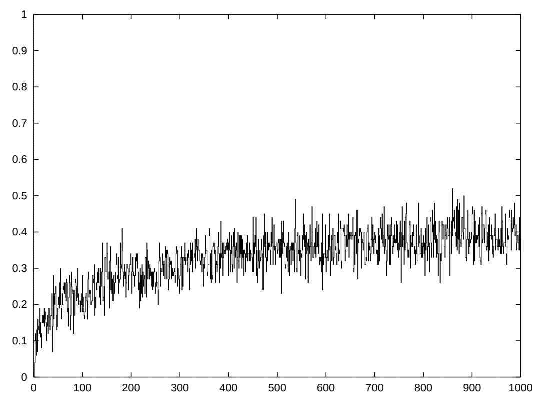
Figure 5.26: Experiment Set 23. Backing a Truck with Two Trailers. Dimensionality: 3. Neural units per dimension: 6. Hitch One Angle: $\pm 5^{\circ}$. Hitch Two Angle: $\pm 5^{\circ}$. Goal Angle: $\pm 5^{\circ}$.



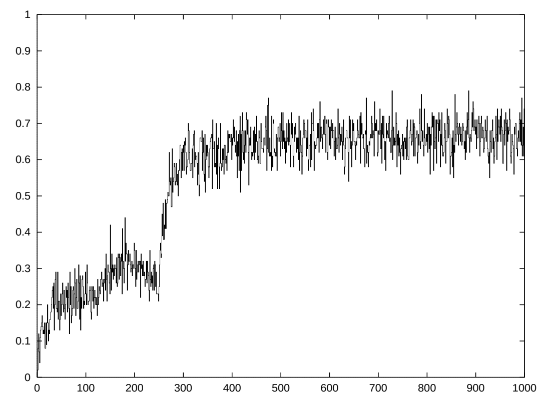
Input Partitioning: Fixed (No Learning)
Response: Learned Individual



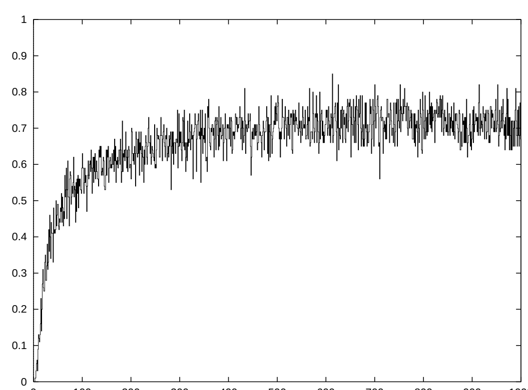
Input Partitioning: Fixed (No Learning)
Response: Learned Cooperative



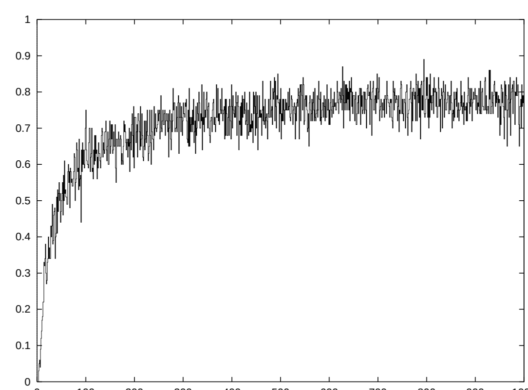
Input Partitioning: Learned Direct
Response: Learned Individual



Input Partitioning: Learned Direct
Response: Learned Cooperative

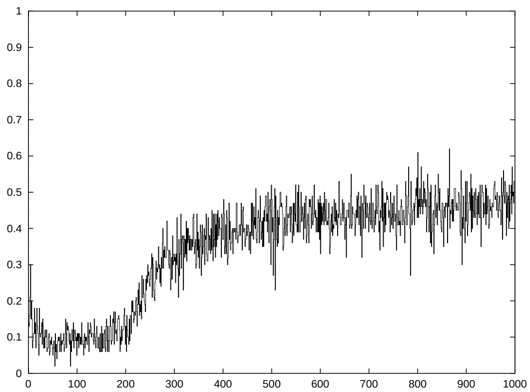


Input Partitioning: Learned Indexed
Response: Learned Individual

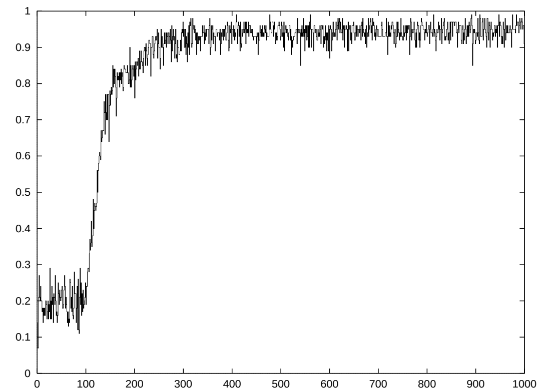


Input Partitioning: Learned Indexed
Response: Learned Cooperative

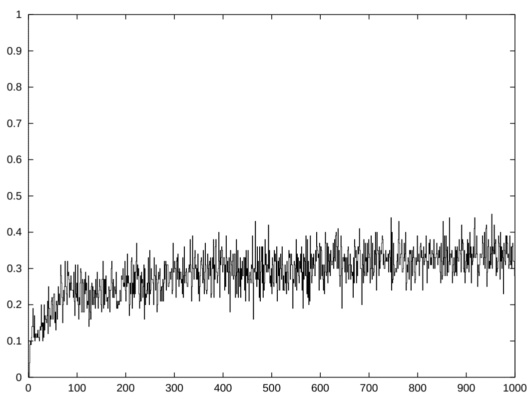
Figure 5.27: Experiment Set 24. Backing a Truck with Two Trailers. Dimensionality: 3. Neural units per dimension: 7. Hitch One Angle: $\pm 5°$. Hitch Two Angle: $\pm 5°$. Goal Angle: $\pm 5°$.



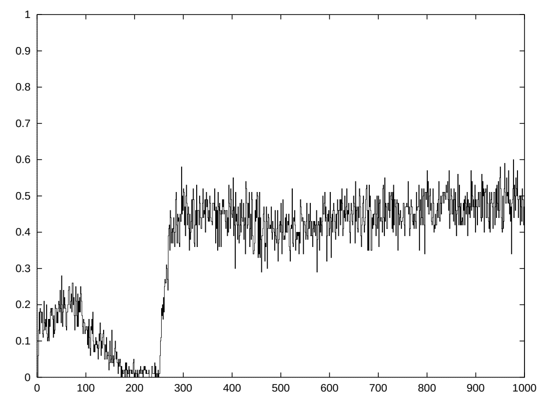
Input Partitioning: Fixed (No Learning)
Response: Learned Individual



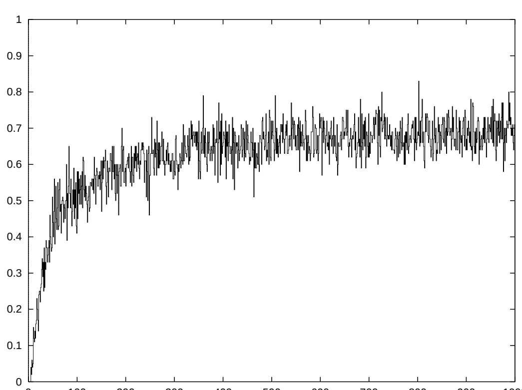
Input Partitioning: Fixed (No Learning)
Response: Learned Cooperative



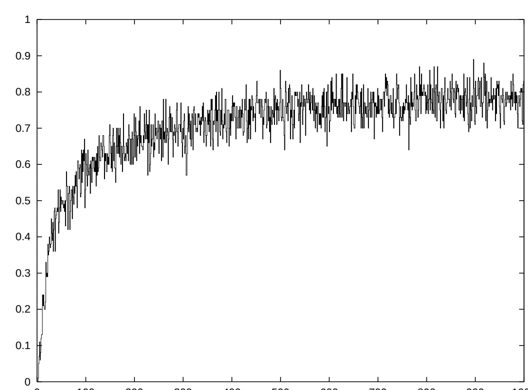
Input Partitioning: Learned Direct
Response: Learned Individual



Input Partitioning: Learned Direct
Response: Learned Cooperative

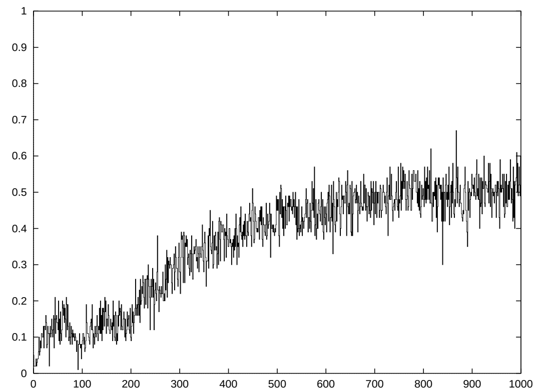


Input Partitioning: Learned Indexed
Response: Learned Individual

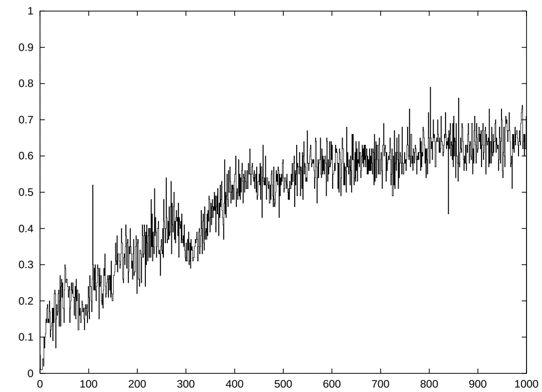


Input Partitioning: Learned Indexed
Response: Learned Cooperative

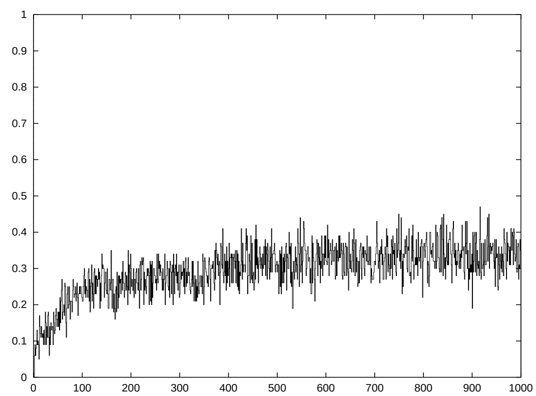
Figure 5.28: Experiment Set 25. Backing a Truck with Two Trailers. Dimensionality: 3. Neural units per dimension: 8. Hitch One Angle: $\pm 5°$. Hitch Two Angle: $\pm 5°$. Goal Angle: $\pm 5°$.



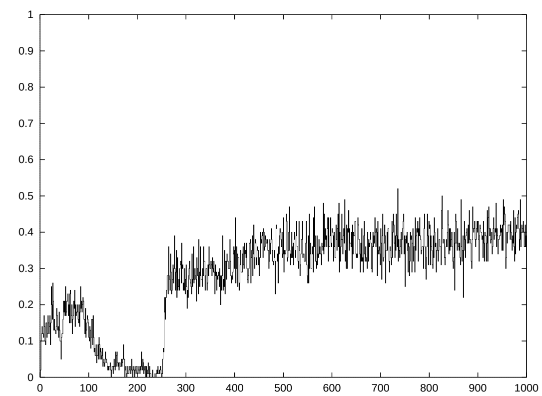
Input Partitioning: Fixed (No Learning)
Response: Learned Individual



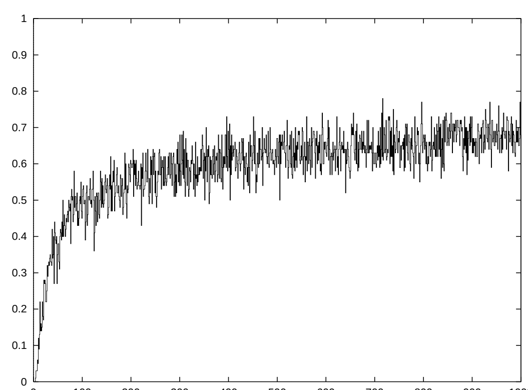
Input Partitioning: Fixed (No Learning)
Response: Learned Cooperative



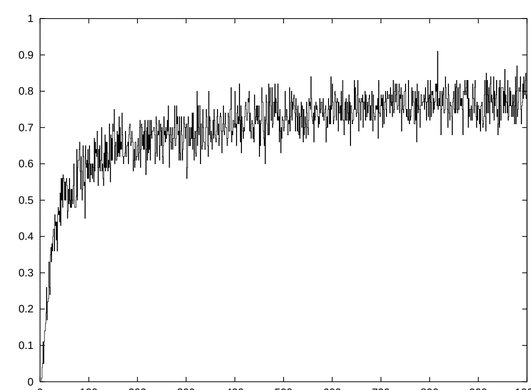
Input Partitioning: Learned Direct
Response: Learned Individual



Input Partitioning: Learned Direct
Response: Learned Cooperative

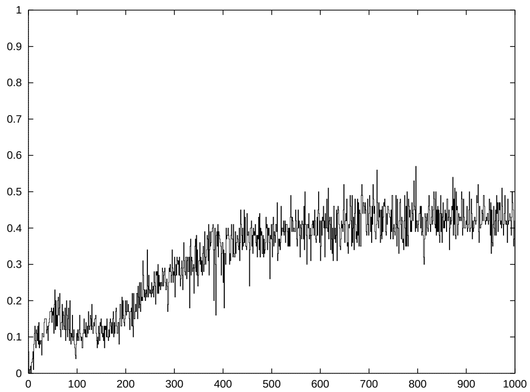


Input Partitioning: Learned Indexed
Response: Learned Individual

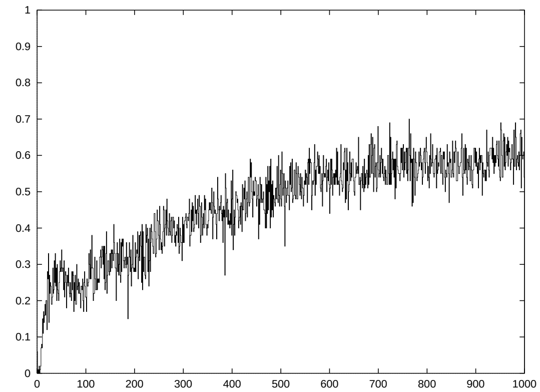


Input Partitioning: Learned Indexed
Response: Learned Cooperative

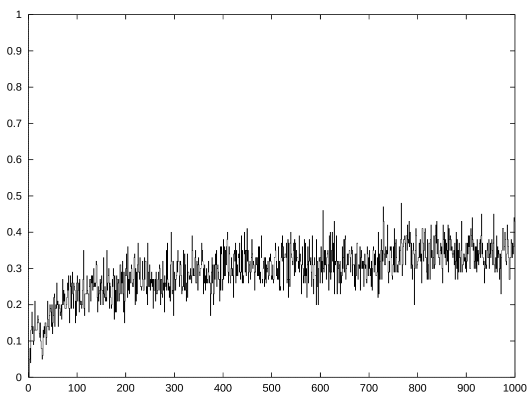
Figure 5.29: Experiment Set 26. Backing a Truck with Two Trailers. Dimensionality: 3. Neural units per dimension: 9. Hitch One Angle: $\pm 5°$. Hitch Two Angle: $\pm 5°$. Goal Angle: $\pm 5°$.



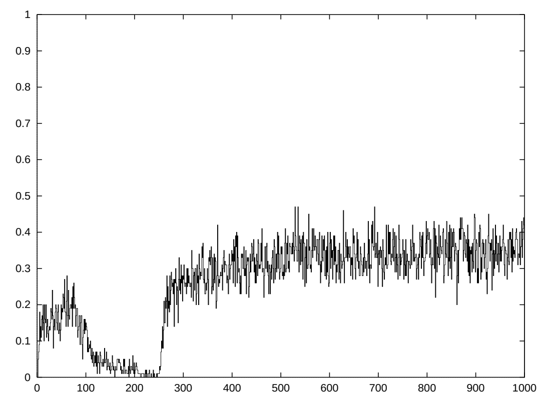
Input Partitioning: Fixed (No Learning)
Response: Learned Individual



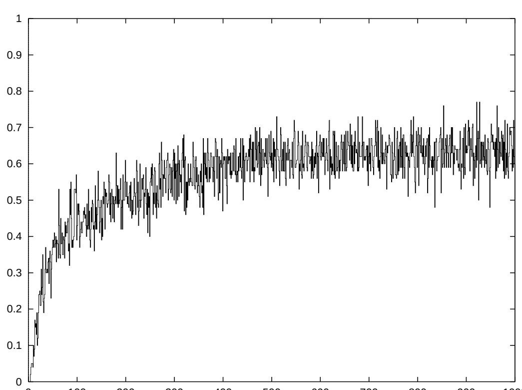
Input Partitioning: Fixed (No Learning)
Response: Learned Cooperative



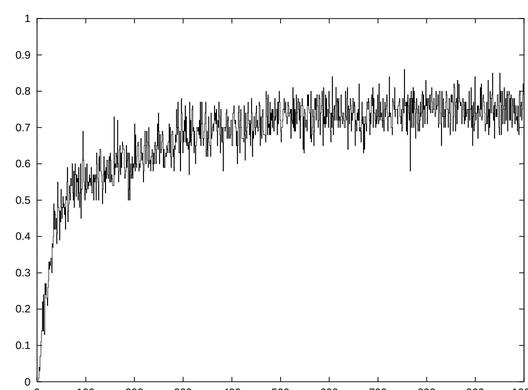
Input Partitioning: Learned Direct
Response: Learned Individual



Input Partitioning: Learned Direct
Response: Learned Cooperative



Input Partitioning: Learned Indexed
Response: Learned Individual



Input Partitioning: Learned Indexed
Response: Learned Cooperative

Figure 5.30: Experiment Set 27. Backing a Truck with Two Trailers. Dimensionality: 3. Neural units per dimension: 10. Hitch One Angle: $\pm 5°$. Hitch Two Angle: $\pm 5°$. Goal Angle: $\pm 5°$.

5.2 *Real-World Experiments*

The real-world experiments are done on the small robotic platform detailed in Appendix B. This robot, known as the Self-Positioning Trailer-Backing Mini-Robot (SPTBMin), was designed and built by Paul Rybski specifically for this task. Details of the robot itself are found in Appendix B.

To measure the hitch angle, SPTBMin is equipped with a potentiometer as the hinge between the truck and the trailer. To measure the goal angle, the goal is lighted and tracked with a light-sensitive rotating head, mounted on the rear of the trailer.

Only one case of the application domain is studied using this robot — one similar to simulation case 1, backing with a single trailer starting from relatively small angles. This case is chosen as it is the easiest to implement in the real world. (To implement a case similar to case 2, for instance, would require that the light-tracking head be able to rotate many times around — something that the hardware will not permit.)

The following parameters are used to determine the initial starting position of the rig:

- The front of the truck cab is started 8 feet from goal.
- The goal angle range is from $-45°$ to $+45°$.
- The hitch angle range is from $-15°$ to $+15°$.

Note that, unlike the simulation case, it is the front of the truck cab that starts at a predetermined distance from the goal, rather than the rear of the trailer. This is due to the methodology used to automate runs on the real robot versus the equations used to model the rig in simulation (see Appendices B and A, respectively). However, since the length of the rig was taken into consideration in choosing the starting distances, the effect is roughly the same.

The basic parameters of the real rig are:

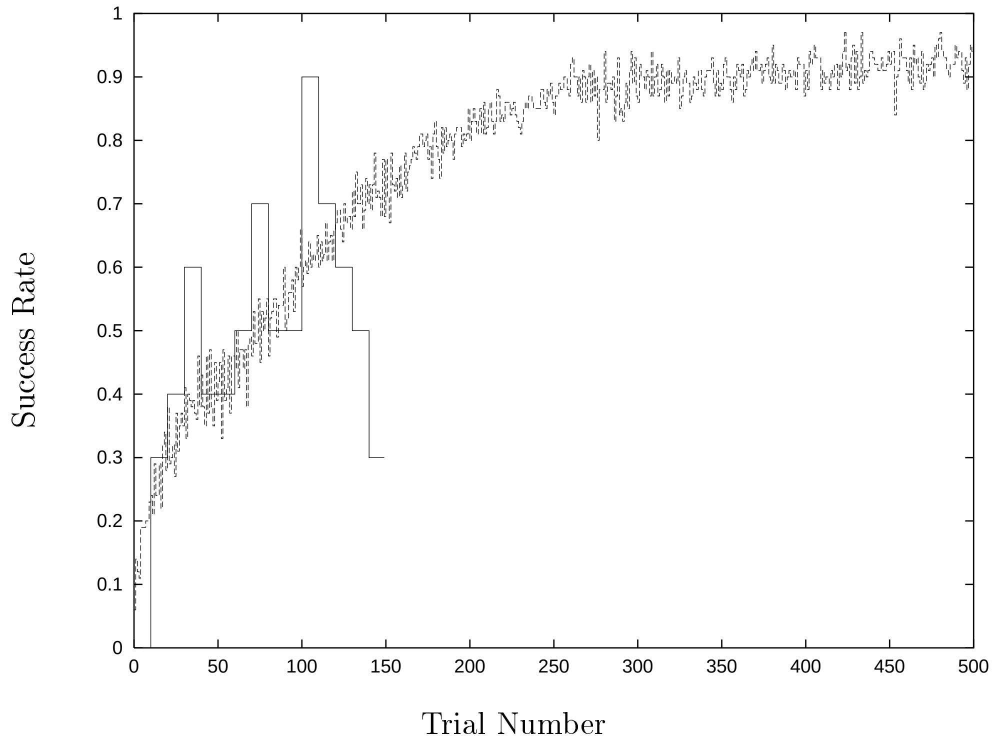
- The truck cab length is 7 inches.
- The trailer length is 11 inches.
- The distance moved by the rear (drive) wheels of the truck on each time step is 0.25 inches.

New initial conditions are chosen randomly at the start of each trial with a uniform distribution over the entire range and the rig positions itself at roughly the angles chosen.

Only four of the six possible versions of the learning system are implemented on SPTBMin. This is because the limited on board memory of the robot is not sufficient to hold both the code and data segments for the versions that learn direct partitionings (versions 3 and 4). Why these versions required more memory than the others, and the implications of this requirement, are discussed in Chapter 6.

In all four versions implemented, the number of neural units per dimension was chosen to be 8.

Each run was stopped at 150 trials (roughly 2.5 hours of learning). Each graph shows a single run. To avoid the graphs containing values of only zero (failure) or one (success) for each trial number, each set of ten trials in a run is averaged with itself. That is, if there are eight failures and two successes in the first ten trials of a run, then the average success rate of 20% will be plotted for the first ten trials. The graphs are plotted against a background showing the performance of the same learning system version on simulation case 1.

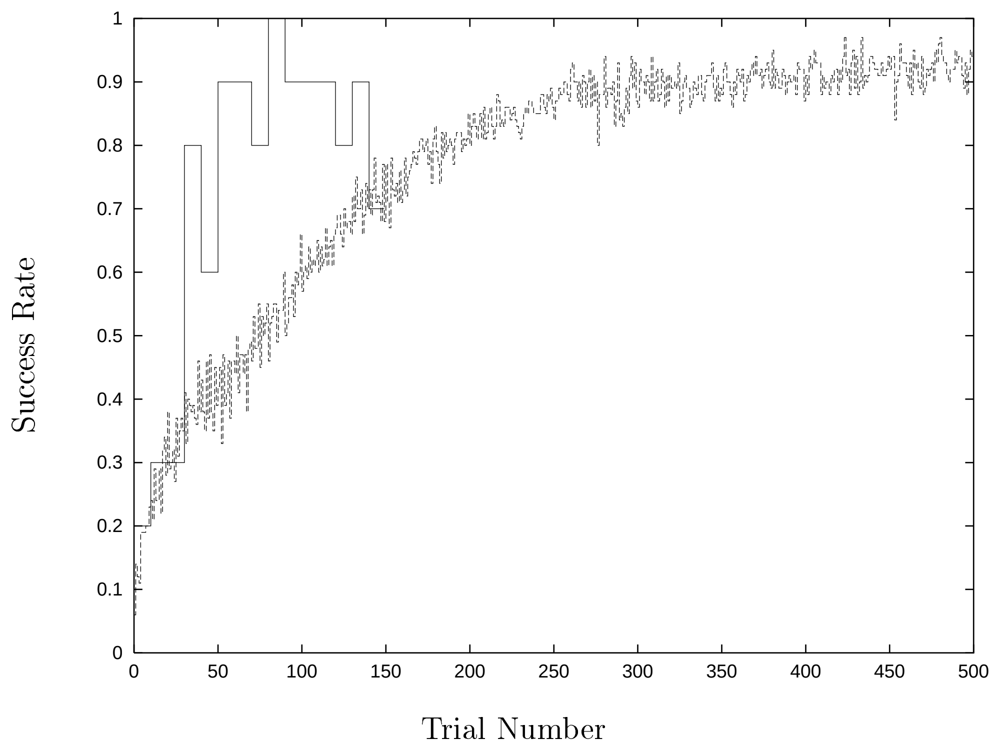


Input Partitioning: Fixed (No Learning). Output Response: Learned Individual.

Dimensionality: 2. Neural units per dimension: 8.

Hitch Angle: ±15°. Goal Angle: ±45°.

Figure 5.31: Run number one of learning system version 1. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.

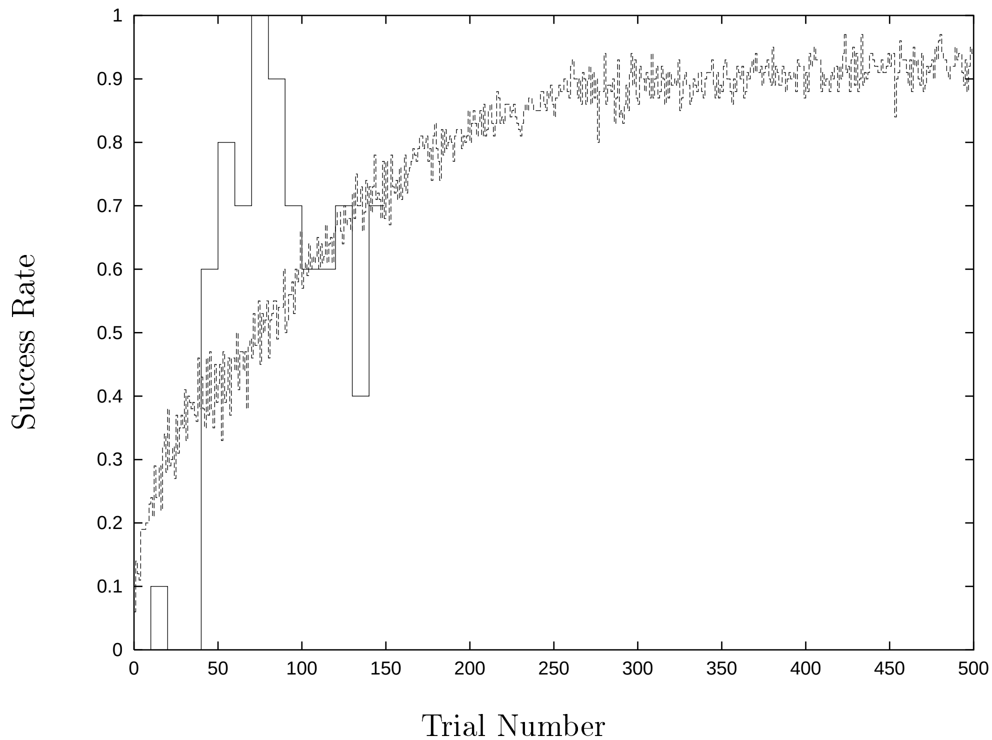


Input Partitioning: Fixed (No Learning). Output Response: Learned Individual.

Dimensionality: 2. Neural units per dimension: 8.

Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.

Figure 5.32: Run number two of learning system version 1. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.

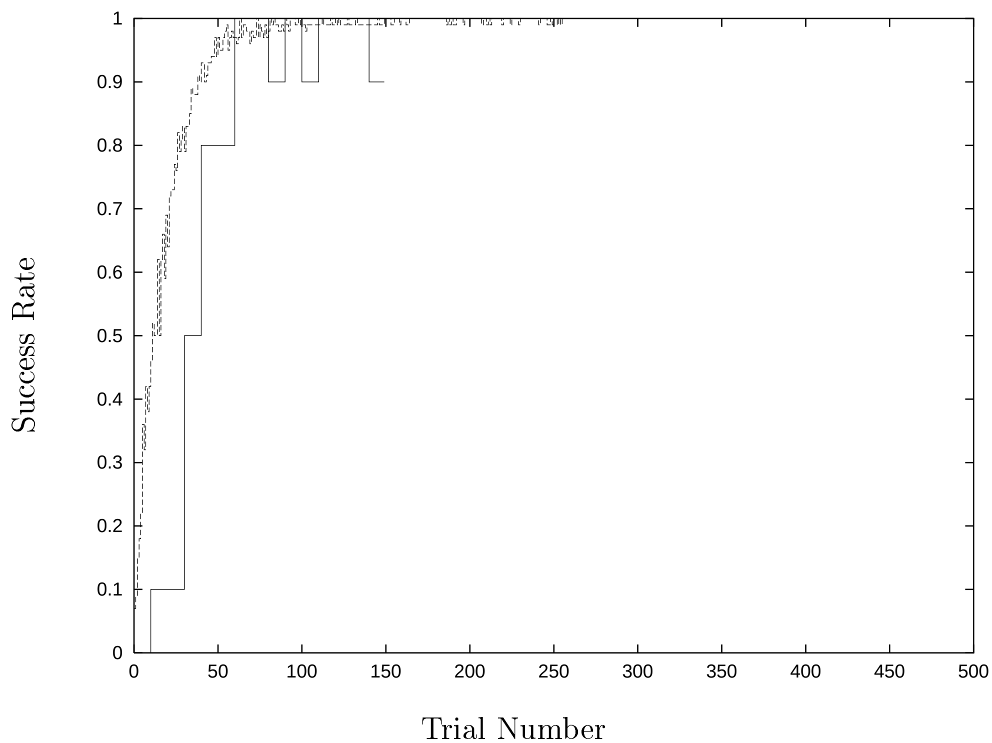


Input Partitioning: Fixed (No Learning). Output Response: Learned Individual.

Dimensionality: 2. Neural units per dimension: 8.

Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.

Figure 5.33: Run number three of learning system version 1. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.

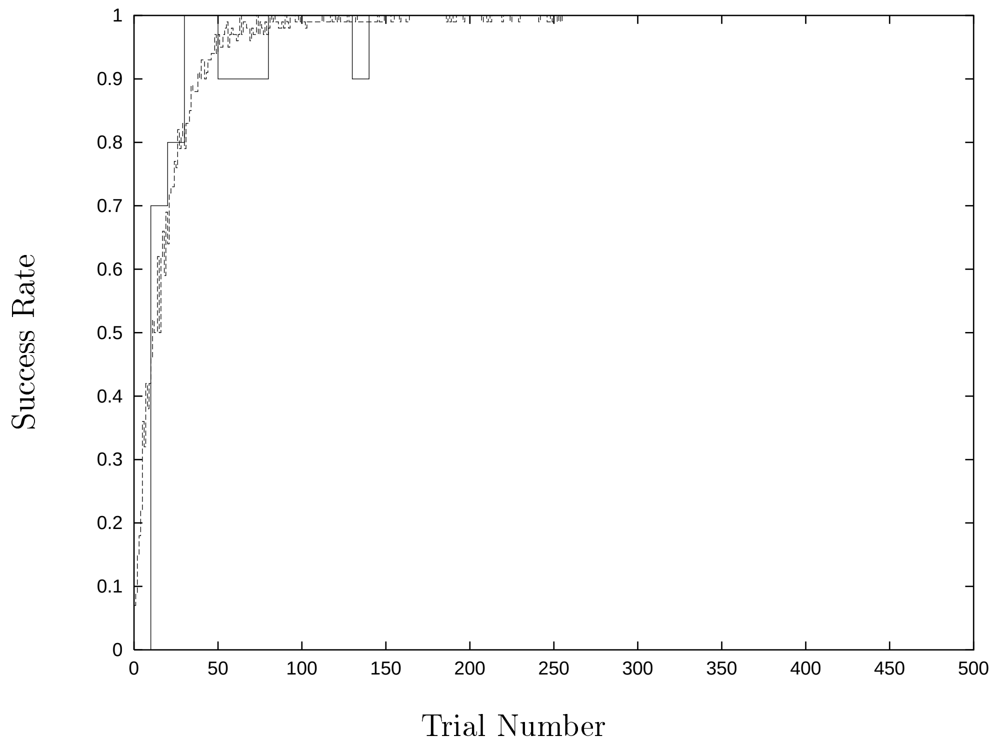


Input Partitioning: Fixed (No Learning). Output Response: Learned Cooperative.

Dimensionality: 2. Neural units per dimension: 8.

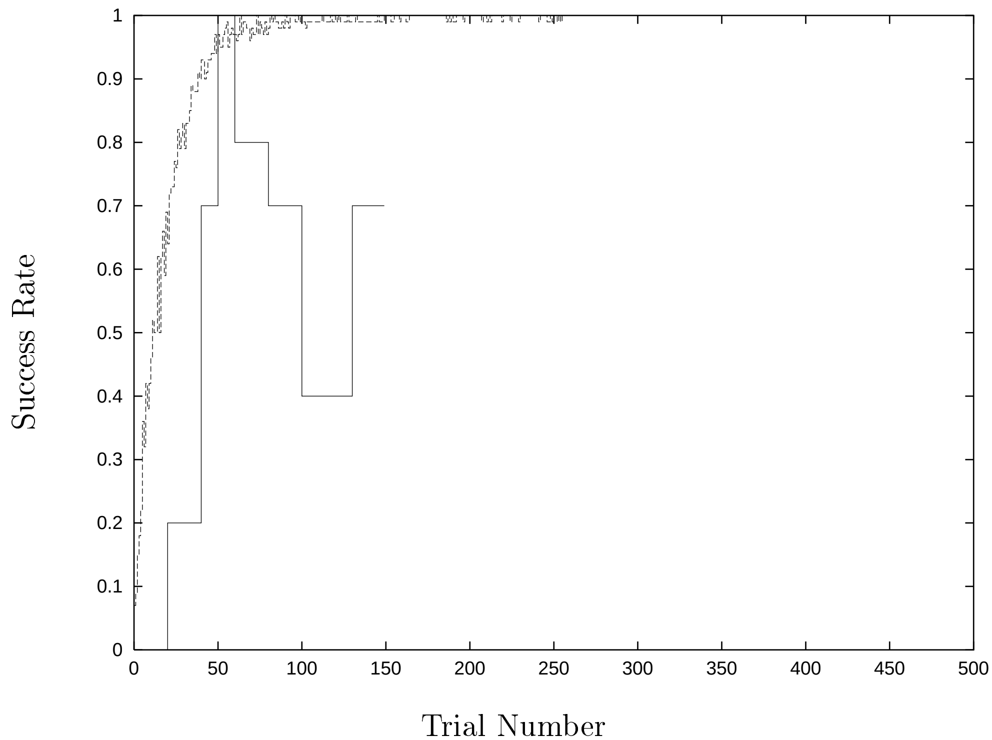
Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.

Figure 5.34: Run number one of learning system version 2. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.



Input Partitioning: Fixed (No Learning). Output Response: Learned Cooperative.
Dimensionality: 2. Neural units per dimension: 8.
Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.

Figure 5.35: Run number two of learning system version 2. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.

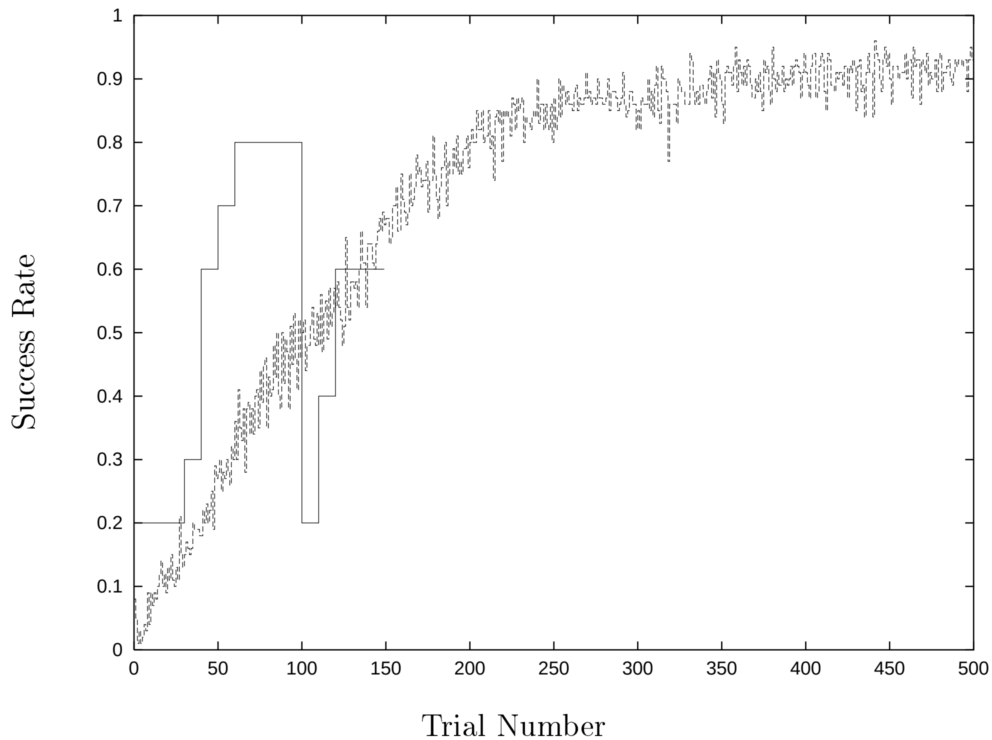


Input Partitioning: Fixed (No Learning). Output Response: Learned Cooperative.

Dimensionality: 2. Neural units per dimension: 8.

Hitch Angle: ±15°. Goal Angle: ±45°.

Figure 5.36: Run number three of learning system version 2. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.

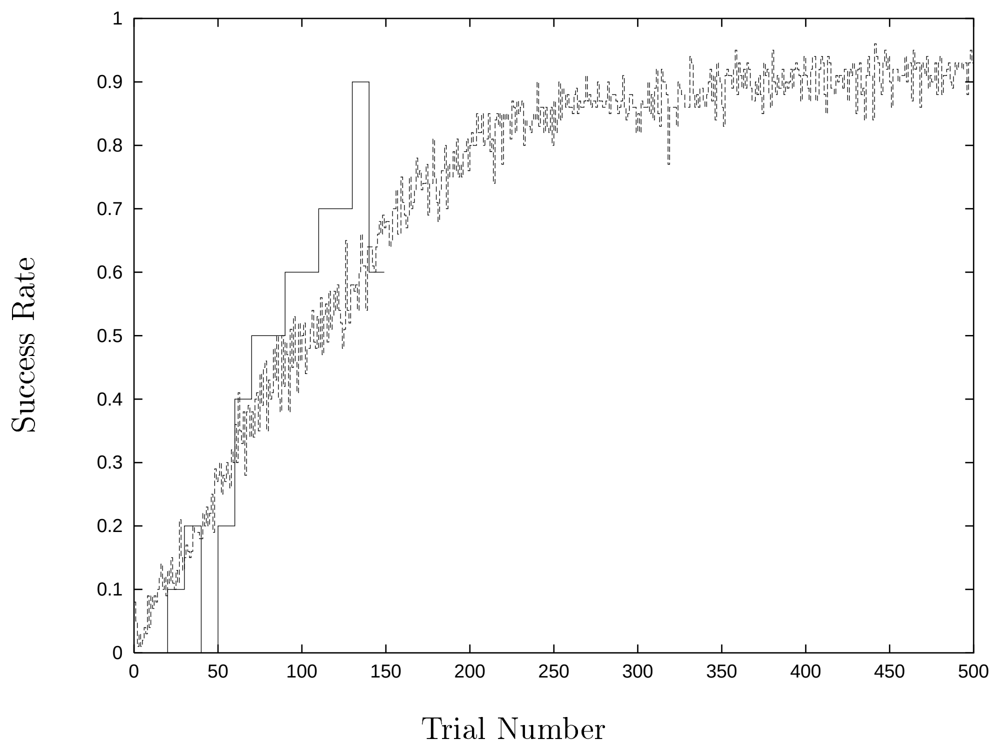


Input Partitioning: Learned Indexed. Output Response: Learned Individual.

Dimensionality: 2. Neural units per dimension: 8.

Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.

Figure 5.37: Run number one of learning system version 5. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.

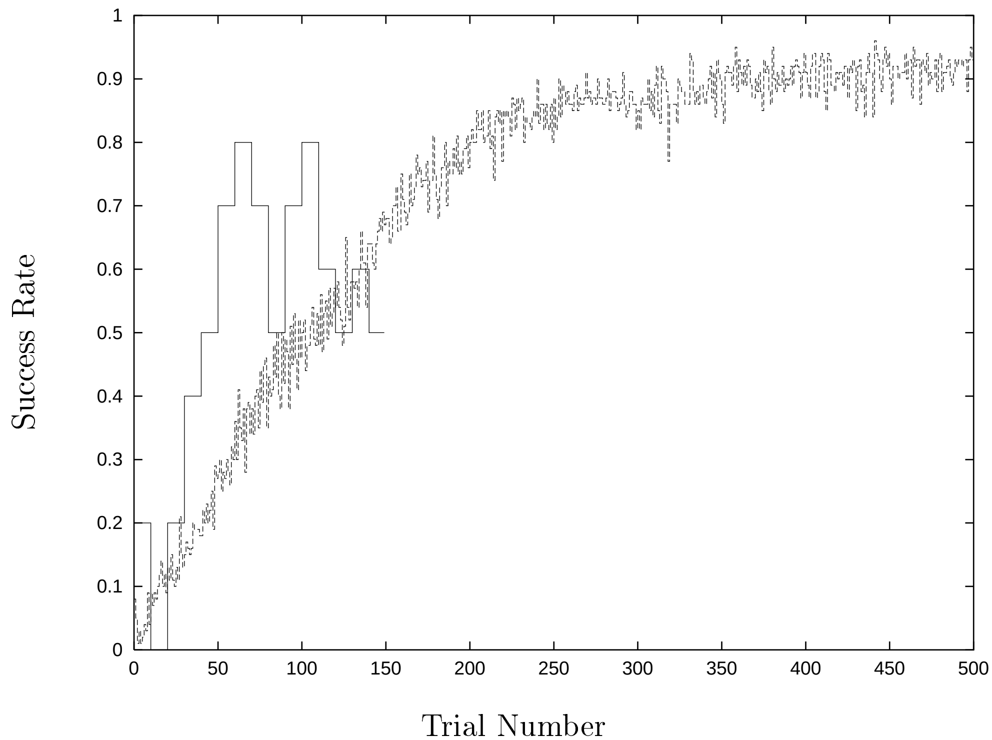


Input Partitioning: Learned Indexed. Output Response: Learned Individual.

Dimensionality: 2. Neural units per dimension: 8.

Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.

Figure 5.38: Run number two of learning system version 5. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.

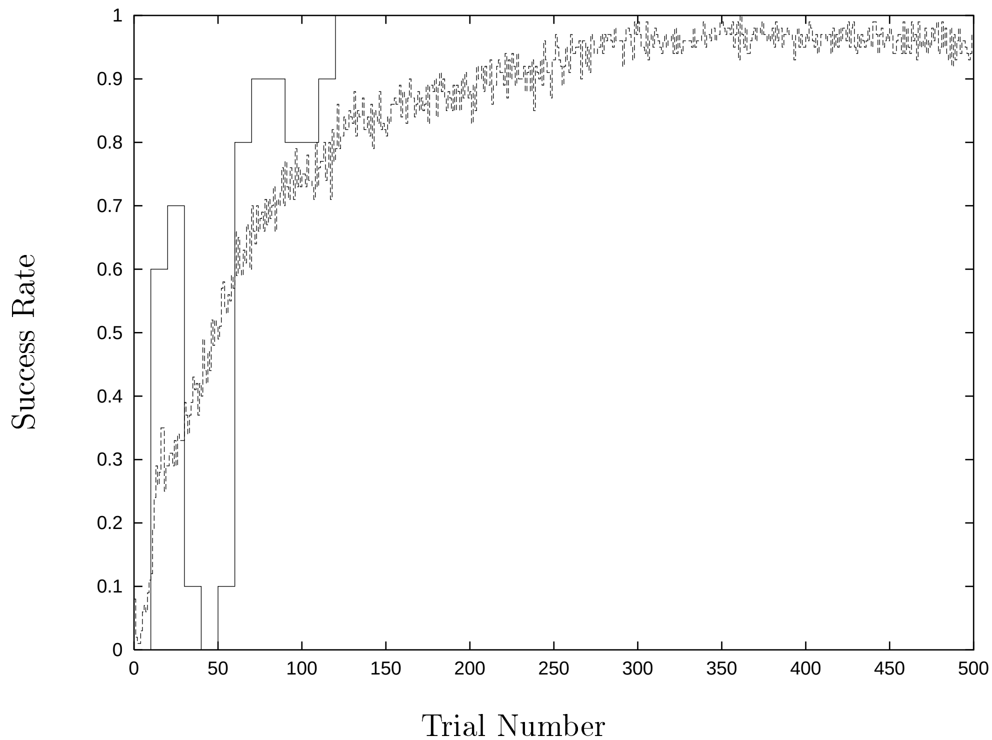


Input Partitioning: Learned Indexed. Output Response: Learned Individual.

Dimensionality: 2. Neural units per dimension: 8.

Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.

Figure 5.39: Run number three of learning system version 5. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.

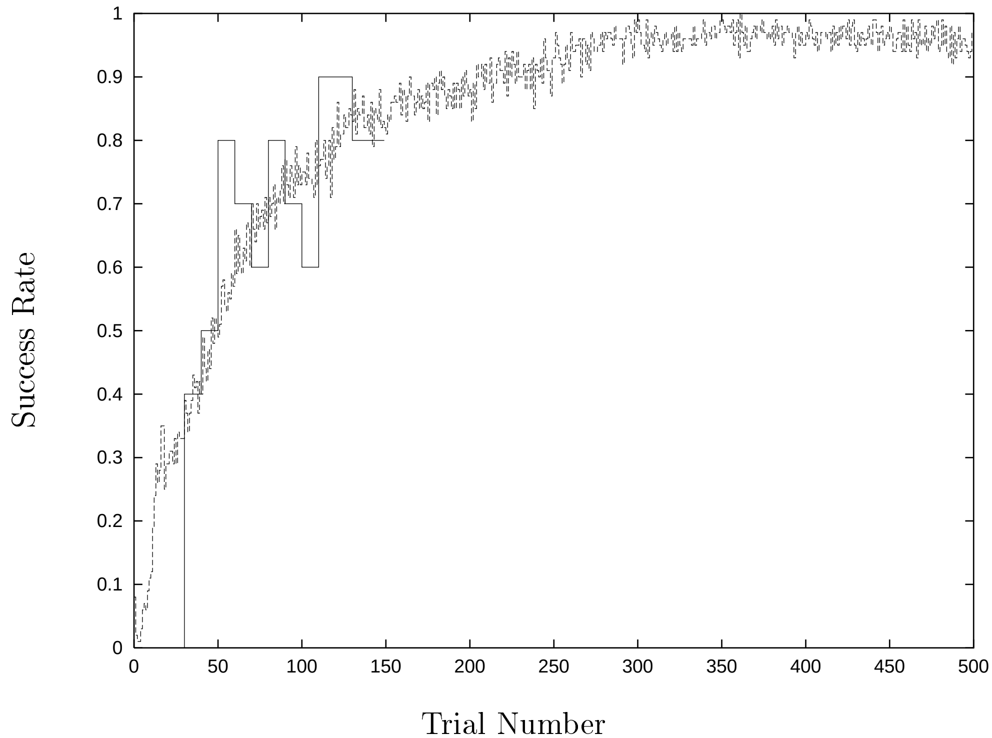


Input Partitioning: Learned Indexed. Output Response: Learned Cooperative.

Dimensionality: 2. Neural units per dimension: 8.

Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.

Figure 5.40: Run number one of learning system version 6. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.

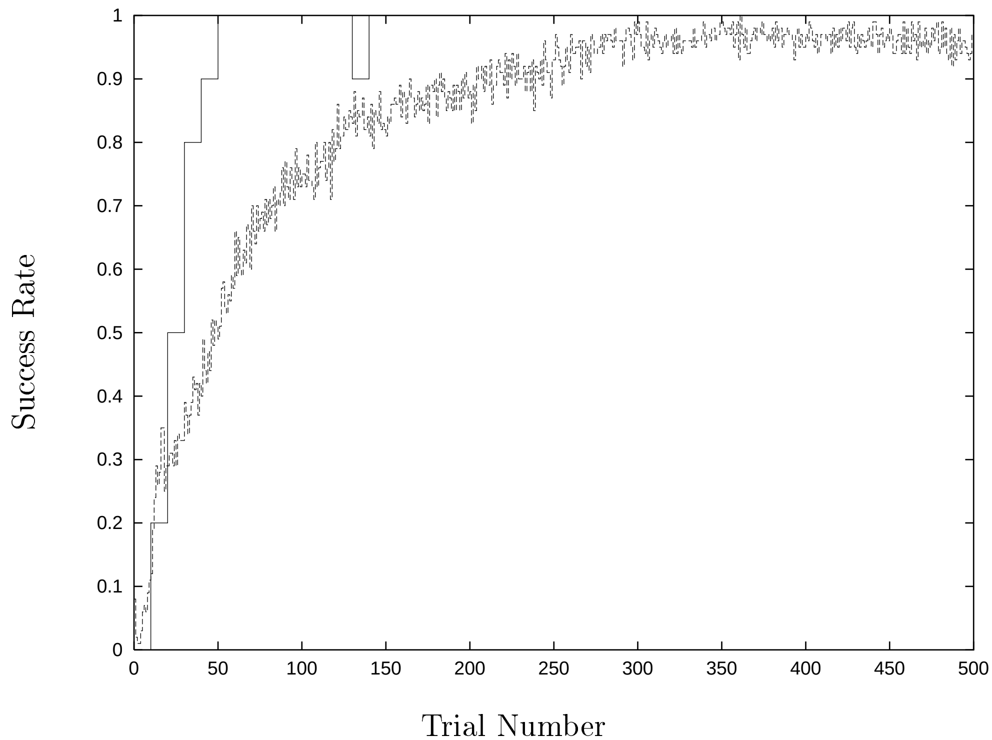


Input Partitioning: Learned Indexed. Output Response: Learned Cooperative.

Dimensionality: 2. Neural units per dimension: 8.

Hitch Angle: ±15°. Goal Angle: ±45°.

Figure 5.41: Run number two of learning system version 6. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.



Input Partitioning: Learned Indexed. Output Response: Learned Cooperative.

Dimensionality: 2. Neural units per dimension: 8.

Hitch Angle: $\pm 15°$. Goal Angle: $\pm 45°$.

Figure 5.42: Run number three of learning system version 6. One run of 150 trials on the real robot plotted against the average success rate of 100 runs of 500 trials each in simulation.

Chapter 6

DISCUSSION

Our discussion of our results includes three sections. In Section 6.1 we discuss other learning methods applied to the truck-backing problem to see what light they can cast on our work. In Section 6.2 we discuss our simulation results. Finally, in Section 6.3 we discuss the results we have obtained on our real robot.

6.1 Comparisons

While it is tempting to directly compare our results with those of other authors who have constructed learning systems for truck-backing, it is important to recognize that simple measures that would allow for such comparisons are not readily available. This is because various authors have formulated the problem differently in terms of the input data supplied to the learning system, the quality of the feedback used in learning, the steering control possible, the physical dimensions of the truck-trailer system, the control objective, etc. So, for example, the fact that the original learning system for truck-backing, developed by Nguyen and Widrow ([99, 100, 101, 160]), took more than 20,000 "backups" to train, certainly does not mean that it is inferior to our learning system. Instead, we must look more closely at the work of those researchers to see how their formulation of the problem differed from ours before we make any comparisons.

What we can easily say from such a figure, however, is that the learning system in question is not suitable for learning on-board real robots — it is only practical for learning in simulation — at least for their version of the problem. If our robot, for example, were to be run through 20,000 trials, it would take roughly two weeks of operation, running 24 hours a day. Further, although our robot is designed to be fully autonomous, it is nonetheless useful to have an operator in the same room (although perhaps engaged in other activities) in case the robot gets into trouble. This means that a more realistic learning time would be well over eight 40-hour work weeks. This is excluding any additional time required for more complex calculations such as those of Nguyen and Widrow's system which are "best suited for off-line computation" [160].

The criterion of suitability for learning on-board real robots is perhaps our most important one and is discussed in more detail in analyzing our results on the real robot (Section 6.3). As far as we are aware, all other researchers investigating systems for learning to back trucks and trailers consider simulated systems only. For this research, we briefly present the formulation of the problem each of these authors use, the learning method they apply, the results they obtain, and some brief comments for comparison.

6.1.1 Woodcock, Hallam, and Picton

Perhaps the most similar learning system applied to the truck-backing problem is presented by Woodcock, Hallam, and Picton [164]. Their formulation of the problem is as follows:

Input Data The distance from the rear of the trailer to the goal, the angle of the ray from the goal to the rear of the trailer relative to some fixed world coordinate system, the angle of the trailer relative to the fixed world coordinate system, and the angle of the truck cab relative to the trailer.

Feedback A performance evaluation dependent on two of the input variables, provided at each time step.

Steering Control Hard-right, hard-left ($\pm 70°$) or continuous.

Objective Back to the goal with trailer at $0°$.

These authors use two similar learning systems: The classic BOXES of Michie and Chambers [90, 91], and a modern variant known as *Fuzzy BOXES*. For both systems the input space is hand-partitioned into 256 non-uniform regions (boxes). The difference between the two systems is that, in the BOXES system the response given by the system on each time step is either a hard-right or hard-left control signal, while in the Fuzzy BOXES system the output is hard-right or hard-left when the input falls near the center of a box but is an intermediate value when the input falls near the border of two or more boxes with different control responses. Note, however, that this difference only comes after training; the systems are always trained using hard-left, hard-right steering.

The failure conditions for these reinforcement learning systems are when jackknife occurs, the distance from the goal exceeds 50 meters (no other indication of the size of any component of the system is given), or the angle of the ray from the goal to the rear of the trailer exceeds $\pm 30°$.

The feedback signal appears to be present on each time step, although this is not explicitly stated, and has a real value in the range of $-2,000$ to $+2,000$ based on the distance from the rear of the trailer to the goal and the angle of the trailer with respect to the fixed world coordinate system. The equation for this feedback signal is

$$f(r, \theta_s) = (50 - r)(50 \cos \theta - r) \tag{6.1}$$

where r is the distance and θ_s is the trailer angle.

Unfortunately, Woodcock, Hallam, and Picton provide no quantitative results in their paper, nor even indicate roughly how many trials they included per run. However, even if results were available, three of the four inputs to their systems are not readily available in our real robot, which would make a meaningful comparison difficult. What we can say is that no attempts are made by these authors to address the primary concerns of our research: learning input partitionings and mitigating the structural credit assignment problem (see Subsections 2.2.2 and 4.1.4). Their 256 non-uniform partition regions are hand-picked and all boxes learn their responses independently. Regardless, this paper provides some answers about possible extensions to our work, and is returned to in Section 7.3.

6.1.2 Koza

Koza [74] provides the only other reference we have found that can be considered to use a form of reinforcement learning applied to truck backing. Koza formulates the truck-backing problem as follows:

Input Data The x and y positions of the rear of the trailer, the angle of the trailer relative to the x axis, and the angle of the truck cab relative to the trailer.

Feedback A *fitness* evaluation based on three of the four input variables, provided at the end of each *fitness case.*

Steering Control Continuous between hard-right and hard-left. (Definition of hard-right and hard-left not given.)

Objective Minimize the fitness value for eight fitness cases.

Koza uses a genetic programming approach for his learning system. This and similar methods from the broad area of evolutionary computation can be seen as reinforcement-learning methods [65], although the literature has remained largely separate.

Koza uses a set of six functions: addition, subtraction, multiplication, a protected version of division in which division by zero returns one, arctangent, and a three argument comparative operator if-less-than-zero which evaluates its first argument and returns either the second or third arguments depending on whether the first is less than zero. Using these six functions, with all four input variables plus random numbers in the range of -1.0 to +1.0 as arguments, he randomly generates a *population* of 1,000 *individuals* (equations). The output of each equation is used as the steering command with values beyond the thresholds of -1 and +1 taken to be hard-left and hard-right steering commands respectively.

Each individual is tested on eight fitness cases (all combinations of two values of x, two of y, and two of trailer angle; the hitch angle was always started at zero). Testing consists of starting at each fitness case location and backing until time expires (3000 time steps — 60 seconds in this simulation), x becomes less than zero (it is started positive in all fitness cases), or the rear of the trailer comes close to the desired final location and orientation. When the test of a fitness case is completed, the fitness value for that individual on that test is measured as the sum of squares of the differences between the desired and actual final location (x and y) and orientation (angle) of the trailer. For each individual, the fitness values for all eight cases are summed to get a total fitness value. Finally, a new generation of 1,000 individuals is created from the fittest individuals in the current population, using *reproduction* (copying an individual) and *crossover* (copying random parts from two individuals). This cycle of testing and generation of a new population is continued until an individual that can successfully arrive at the target location and orientation from all eight starting positions is found, or until 51 generations have been produced.

Koza details one run in which an individual meeting his success criterion was found after 27 generations and indicates that other runs produced similar results. Given one minute for each individual to be tested on each fitness case on a real robot, this would take over one and a half years of 40-hour work weeks to run on a single robot. Of course, if one had multiple robots, this process could be greatly parallelized. With 1,000 robots, one for each individual in a population, this process would take only a little over three and a half hours. Still, this method does not appear appropriate for learning on-board real robots, at least for this version of the truck-backing problem, as 1,000 robots in 1,000 separate testing arenas would be a rather expensive proposition. (Actually, one could get by with only 900 robots after the first generation because, in each generation, 10% of the individuals were direct copies from the last generation and would not need to be re-tested.) Also, since no attempt was made to test the individuals on any cases besides these eight, it is difficult to know how well even the fittest individual could generalize to novel starting positions. We can, however, find some interesting information in this study, and it will be returned to in Section 7.3 with regards to possible future work.

6.1.3 Nguyen and Widrow

The basic formulation of the problem by Nguyen and Widrow is as follows:

Input Data The x and y positions of the rear of the trailer, the x and y positions of the hinge between the trailer and the cab, the angle of the trailer relative to the x axis, and the angle of the truck cab also relative to the x axis.

Feedback An error vector provided at the end of backing, giving the difference between the actual and desired x and y positions of the rear of the trailer and the angle of the trailer to the x axis. (The desired values are constant.)

Steering Control Continuous between hard-right and hard-left. (Definition of hard-right and hard-left not given.)

Objective Minimize the error vector.

The learning method they use to solve this problem is back-propagation through time. Learning proceeds through two phases, one for training a neural-network emulator, and the second for training the actual controller. During the first phase, the truck is backed randomly through "many cycles" until the emulator does a good job of predicting what the next state will be, given a state and a control signal. The emulator is trained using ordinary back-propagation. During the second phase, the controller chooses the steering signal and when the truck jack-knifes or crosses the y axis (it is always started with a positive x value), feedback is given and a new initial starting state is chosen. This training phase is divided into several *lessons* beginning with "easy" positions (close to the target location and with the trailer pointed at it) through increasingly more difficult starting positions.

The result is that, after sixteen lessons, each consisting of from 1,000 to 2,000 backups, the root-mean-square error of the y value of the rear of trailer was about 3% of the truck length and that of the angle of the trailer to the x axis was about 7%. (There was no error in the x value of the rear of the trailer since the rig no longer jack-knifed and all backups were stopped when the rear of the trailer reached the desired value.)

The problem formulation used by Nguyen and Widrow is certainly challenging. For one thing, they have provided redundant information to their system. Only one set of x and y coordinates and the two angles are necessary to fully constrain the position of the rig in the environment. Providing the second x and y only adds to the number of neural units and associated weights that need to be trained in both the emulator and controller.

More importantly, restricting their feedback to a terminal signal means that most supervised learning methods (such as ordinary back-propagation) could not be used to directly solve this problem — the invention of back-propagation through time was an elegant solution. However, for purposes of comparison with our work, it must be noted that back-propagation through time is a supervised-learning method. Like the critic in our system that provides only a terminal signal, the teacher in Nguyen and Widrow's system only provides one feedback response per backup. However, their teacher provides an error vector rather than a boolean success/failure signal (as in our formulation) or other qualitative evaluation. That is, the learning system is not only told how well it did (the type of feedback given by a critic) but also what the direction and magnitude of its mistake was. This additional information greatly simplifies learning, as the system can use it as a gradient to follow towards a better solution. Nonetheless, as discussed above, it does not appear to be well-suited for learning on board a real robot, even when a teacher is available.

6.1.4 Kong and Kosko

Kong and Kosko [73] provide two formulations of the problem, one for backing just a truck cab with no trailer, and the other for backing a truck with a single trailer. The description of the truck with a single trailer follows, as it is more similar to our formulation.

Input Data The x coordinate of the rear of the trailer, the angle of the trailer relative to the x axis, and the angle of the truck cab relative to the trailer.

Feedback An error vector on each time step.

Steering Control Seven discrete values ranging from "Negative Big" to "Positive Big" for the fuzzy controller; continuous values for the neural network controller.

Objective Minimize the error in the feedback signal.

Kong and Kosko investigate two learning systems, both of which are supervised. The first is a backpropagation neural network and the second is an *Adaptive Fuzzy Associative Memory* (AFAM). The supervision comes from an existing fuzzy controller that the authors hand-coded.

For the neural network, 52 training samples (input data matched with correct output response) were chosen and 200,000 iterations were run on this training set. Most training sequences converged to good performance although some failed to converge.

For the adaptive fuzzy controller, the authors partitioned the space into 735 non-uniform regions. Using 6250 training samples and a method known as *Differential Competitive Learning* (DCL) the authors produced new fuzzy controllers that generally functioned similar to the one used as the teacher "except in a few cases, where the DCL-estimated AFAM trajectories were irregular" (p. 223).

Since Kong and Kosko present supervised methods, it is difficult to compare their systems with ours. However, it is worth noting that if a well-distributed subset of correct behaviors is known (such as the 52 training samples used to train the neural network), it may be possible to learn generalizations that function well in similar situations. Also, one interesting note is that these authors choose to have the angles of the truck cab run from $-90°$ to $+270°$, rather than the more common $-180°$ to $+180°$. This point is returned to in Section 6.3.

6.1.5 Geva, Sitte, and Willshire

Geva, Sitte, and Willshire [45] have three formulations of the problem, including backing a rig with a single trailer, backing a rig with a single trailer along the shortest trajectory, and backing a rig with two trailers. The description of the first of these follows:

Input Data The angle of the ray from the goal to the rear of the trailer relative to a fixed world coordinate system, and the angle of the trailer relative to the ray just described.

Feedback A boolean success-failure signal.

Steering Control Either hard-right, hard-left or continuous.

Objective Back to the goal and align the trailer at $0°$ relative to the fixed world coordinate system.

The method presented is barely a learning method at all but it does provide some insights, both into learning methods and into the truck-backing problem. The first thing the authors do is to hand-craft an *override steering signal* to prevent jack-knifing; when the hitch angle grows too large (they have determined in advance what 'too large' means), the override signal kicks in and simply turns the wheels in the proper direction (which they have also determined in advance) and thereby avoids jack-knifing.

With the difficulty of jack-knifing taken care of, these authors do not need to bother their learning system with the hitch angle. But they have done much more than simply reducing the number of inputs that their learning system must deal with — they have taken a highly non-linear control problem and replaced it with a linear one. Having thus created for themselves a linear control problem, they appropriately craft a simple linear equation of the two variables they are giving to their learning system. For the unknown weights in this equation, they choose a small range in which they know good values fall, and then analyze what the chances are of randomly picking good values.

For optimizing the trajectory length, the authors simply add a term to their equation to account for the distance to the goal and weight it by an appropriate amount. They do not discuss having the system learn this weight.

Finally, for backing a truck with two trailer they drop the idea of minimizing trajectory length and add a second override steering controller to prevent the angles of either hitch from getting too large and then show that the original linear equation still works, albeit with different weight values. This is not too shocking, since the addition of the second override steering controller has reduced this problem to a linear one as well. However, they do go on to reduce the "wobble" present in this solution by adding a third term to their original linear equation to account for the angle between the two trailers. They end by claiming that "it is again easy to find good weights, although it is more sensitive to the selection of weights than it is with a single trailer configuration" (p. 855).

While a random choice of weight values is certainly an unsupervised learning method, Geva, Sitte, and Willshire have done so much analysis of the task, and decomposition into smaller sub-tasks, that what remains for learning is minimal. In fact, their approach is similar to the one independently arrived at by Jenkins and Yuhas at about the same time [62]. The primary difference in approach is that Jenkins and Yuhas do not present their system as a learning system at all — they simply hand-code the entire controller through a decomposition of the task into subtasks and skip the part about randomly choosing weights within a small range.

The approaches of both Geva, Sitte, and Willshire, and of Jenkins and Yuhas can most profitably be compared with other non-learning approaches, such as that of Tanaka and Sano [141, 142] who, using *fuzzy control*, give a proof of the stability of their solution and present results from implementing it both in simulation and on a real robot. Nonetheless, Geva, Sitte, and Willshire provide a good reminder that the way we cast a problem, and how much of the problem is solved before it is ever given to an automated system to learn, may make all the difference in the world.

6.2 Discussion of Simulation Results

Because no adequate comparison can be made with the work of other researchers who have investigated the truck-backing problem, we have gathered results for six different versions of our learning system. The first of these versions, which uses a fixed input partitioning and individual response learning, can be seen as a baseline system against which the others may be compared. It is similar to the basic system from which other researchers have started (e.g. Barto, Sutton, and Anderson [12]); perhaps the only thing truly original about our version is the use of the saturating eligibility trace (see Subsection 4.1.3).

As we make comparisons between systems we will be looking at results obtained for each version of the learning system with networks of the same dimensionality, just as they are presented in the pages of graphs in Chapter 5. However, it should be noted that this is not an entirely accurate reflection of the complexity of each system; different versions of the learning system have different requirements in terms of both memory and computations. For example, adding cooperative output learning does not add any memory requirements for storing data, but does require additional computation (Equation 4.10) at the end of each trial. The differences in the systems in terms of their memory and computational requirements are summarized in Table 6.1.

The differences in memory requirements include the number of input weights that need to be stored, the number of output weights that need to be stored, and whether a log of the input states needs to be recorded for input weight adjustment at the end of a trial. Note that for learning system versions 3 through 6 the number of input weights that need to be stored is equal to the number of neural units in the input networks. However, because we are using rectangular partition regions for our fixed partitioning systems (versions 1 and 2), there is no need to store all weight values. Instead, we can treat versions 1 and 2 as though they were fixed versions of learning systems 5 and 6, rather than of 3 and 4, and store only $d_I * \eta$ values, rather than all η^{d_I} weights. If our fixed partition regions were non-rectangular, we would have to store all weights. The length of the state log was set at 1,000 time steps for simulation. For use on the real robot, however, the log contained only a subsample of the inputs, saving only every fifth state, and was therefore only one-fifth as long.

The differences in computational requirements include the time it takes to classify input data, the time it takes to adjust the partition boundaries, and the time it takes to do inter-neural data sharing for cooperative output learning. The classification of the input data takes place during a trial and thus impacts the real-time performance of the system as the robot moves. The adjustment of the partition boundaries and the cooperative output learning take place at the end of a trial and affect the amount of time the robot remains stationary before the next trial begins. As with the storage of input weights, the time for classification for learning system versions 1 and 2 is reduced because we have chosen to use rectangular partition regions. If we used non-rectangular partition regions, the computational requirement for input classification would be equivalent to that for versions 3 and 4, rather than 5 and 6.

	Learning System Version					
	1	2	3	4	5	6
Input	$d_I * \eta$	$d_I\eta$	η^{d_I}	η^{d_I}	$d_I\eta$	$d_I\eta$
Output	η^{d_I}	η^{d_I}	η^{d_I}	η^{d_I}	η^{d_I}	η^{d_I}
Log?	No	No	Yes	Yes	Yes	Yes
Class.	$O(\eta)$	$O(\eta)$	$O(\eta^2)$	$O(\eta^2)$	$O(\eta)$	$O(\eta)$
Part.	N/A	N/A	$O(\eta^{d_I+1})$	$O(\eta^{d_I+1})$	$O(\eta)$	$O(\eta)$
Coop.	N/A	$O(\eta^{d_I}W(T)^{d_I})$	N/A	$O(\eta^{d_I}W(T)^{d_I})$	N/A	$O(\eta^{d_I}W(T)^{d_I})$

Table 6.1: Differences in the six versions of the learning system with respect to memory and computational requirements. *Input* is the number of weights in the input network. *Output* is the number of weights in the output network. *Log?* is the need for a log of input states. *Class.* is the computational time needed for classification of an input vector. *Part.* is the computational time needed for adjusting the input partitioning. *Coop.* is the computational time needed for cooperative response learning. d_I is the number of dimensions in the input space. η is the number of neural units per dimension of the input space. $W(T)$ is the width of the neighborhood used for cooperative response learning and changes with the trial number T.

From this table we can see that learning system versions 3 and 4 require the most memory and that versions 1 and 2 require the least. Versions 3 and 4 also require the greatest computation during each trial, although 1 and 2 would require equal computation for non-rectangular partition regions. It is more difficult to give a full ordering of post-trial computational requirements but we can say that learning system version 4 requires greater computation than both versions 3 and 6, which both require greater computation than versions 2 and 5, which require greater computation than version 1.

6.2.1 Criteria for Evaluation

While these results are from simulation experiments, our goal is to develop a system that can learn on real robotic platforms. For this reason we are primarily concerned with:

1. rapidly reaching high levels of performance, and

2. an ability to function well over a range of conditions.

6.2.2 Evaluation of Learning System Version 1

For version 1 of the learning system (the baseline system) applied to simulation case 1 (backing a single trailer with small angles), there is a general trend towards a better ultimate success rate as network size increases. This is to be hoped for, since we are applying greater computational resources as network size increases; if we allocate more resources to a task, it would be nice to be able to expect a better result. We should probably expect that there will be a point of diminishing returns but we would rather not have wild fluctuations in the ratio of allocated resources to resulting success rate. Unfortunately, there is one such fluctuation. For a size of four neural units per dimension the system achieves an average success rate of roughly 30% by the end of 500 trials, for a size of five neural units the average success rate drops to less than 10%, but with a size of six neural units per dimension the rate has jumped back up to greater than 65%. (See Table 5.2.) This is, most likely, due to the partition boundaries in each case. Because the input space is simply partitioned evenly in each dimension, adding neural units shifts all of the boundaries.

Still looking at learning system version 1 applied to case 1, note that we do, in fact, see a point of diminishing returns — the ultimate success rate hardly improves after we reach a network size of seven neural units per dimension, despite the fact that we are moving from a system with 49 neural units in its input and output networks to one with 100 units in each network. (Again, refer to Table 5.2.) Further, note that as we move from eight to ten neural units per dimension, the increase in successes early on (the slope of the graph in the two hundred time steps) decreases. (See Figures 5.9, 5.10, and 5.11.) That is, it takes longer to improve performance in early trials as we increase network size above eight neural units per dimension. This is most likely due to the structural credit assignment problem.

When looking at learning system version 1 applied to case 2 (backing a single trailer with large angles) we find no evidence of learning with fewer than six neural units per dimension. (See Table 5.3.) This is also true of learning system version 2, but not of any of the other versions which have anywhere up to 75% for an ultimate success rate with smaller networks. This is a good indication that the difficulty is that a uniform gross partitioning simply is not sufficient to solve this problem.

With a finer partitioning, the baseline learning system shows improvements — an ultimate success rate of roughly 35% for six neural units per dimension and over 60% for seven. Unfortunately, this is as good as it gets for this system; as the partitioning is made finer still, the systems performance stays constant (for eight units per dimension) and then progressively declines (for nine and ten). Again, this is probably due to the structural credit assignment problem.

Finally, for simulation case 3 (backing with two trailers), version 1 of the learning system shows expectedly poor results for few neural units per dimension (two to four), a large flat plateau at around 50% for five to nine, and a slight decline to less than 45% for ten. The lack of improvement above five neural units per dimension probably reflects the fact that the structural credit assignment problem is exacerbated by working in three dimensions.

On the whole, the basic learning system is capable of learning to at least a moderate level of success in all three simulation cases, sometimes doing rather well. It does not do well with too few neural units per dimension, as the uniform gross partitioning does not put decision boundaries where they need to be. Nor does it do well with too many neural units, as it succumbs to the structural credit assignment problem.

6.2.3 Evaluation of Learning System Version 2

Like learning system 1, system version 2 does poorly with too few neural units per dimension (three or fewer for simulation cases 1 and 3; five or fewer for case 2). This is to be expected, as both use the same fixed uniform partitioning.

The difference between the systems is in how they learn output responses. Learning system two was developed in order to overcome the structural credit assignment problem found in the basic learning system and so uses cooperative response learning.

Cooperative response learning clearly improves the ability of the system to learn good solutions in simulation case 1. At each network size from four neural units per dimension on up, learning system version 2 achieves a better ultimate success rate than does learning system version 1, sometimes markedly so. Perhaps even more impressive is the rapidity with which learning system 2 progresses in early learning trials. This is most clearly expressed for network sizes of seven and eight neural units per dimension (see Figures 5.8 and 5.9), in which learning system version 2 has achieved over 95% success after only 50 trials. (For these same network sizes, learning system version 1 had achieved roughly 25% and 40% respectively after 50 trials and still had not achieved a success rate of 95% after 500 trials.) This is also seen for network sizes of four, six and ten, although not to the same degree.

Unfortunately, there seems to be some difficulties for network sizes five and nine. It appears that the location of the partitions is causing some neural units to straddle what should be decision boundaries, causing units to share inappropriate experiences with one another. As the trials proceed, however, experience sharing is reduced and the individual neural units are able to learn their own correct responses resulting in high ultimate success rates at those network sizes as well.

The results for learning system version 2 on simulation case 2 are markedly different. Here version 2 achieves somewhat greater success than version 1 for a network size of ten but somewhat less for a network size of six; for all other sizes, the results are similar. It is clear that the primary difficulty in learning on simulation case 2 is not the structural credit assignment problem but the location of the partition boundaries.

For simulation case 3, we are back to having version 2 of the learning system outperform version 1 for networks of size five and greater. However, it is clear that partition boundaries make a big difference for this simulation case as well. The ultimate success rate oscillates between values of somewhat greater than 60% for network sizes five, seven, and nine, and values over 90% for network sizes six and eight. Further, for network sizes six and eight, a success rate of 85% or greater has been achieved in the first 300 trials, whereas for network sizes five, seven, and nine, the success rate is still below 50% at this point.

6.2.4 Evaluation of Learning System Version 3

Learning system version 3 is an attempt to give a system the ability to learn its own partitioning. From the partitioning learned by the input network, selection of an appropriate responding unit from the output network is made directly. Like learning system version 1, version 3 learns output responses individually for all neural units.

It is clear that learning system version 3 succeeds in learning better than uniform partitionings for smaller network sizes. In simulation case 1, learning system version 3 greatly outperforms both learning system version 1 and version 2 for networks of sizes two and three neural units per dimension; in simulation case 2, version 3 outperforms versions 1 and 2 for network sizes of six neural units per dimension or less; and in simulation case 3, version 3 outperforms versions 1 and 2 for network sizes of four or less.

Unfortunately, as network sizes increase beyond these levels, performance soon peaks and then drops (drastically in simulation case 2). This is due to the structural credit assignment problem again. In fact, the structural credit assignment problem may be exacerbated by the system's attempt to learn its own partitioning. Previously there were some few states that were visited infrequently and therefore it was difficult to determine the correct response for them, but there were also many states that were visited regularly and for which there was plenty of experience from which to learn. As the system attempts to learn its own partitioning, however, it is bringing more neural units to regions with high activity levels. This makes the partitioning finer in these regions, allowing for a better differentiation between important states, yet dilutes the experiences by spreading them over more neural units.

6.2.5 Evaluation of Learning System Version 4

Learning system version 4 attempts to combine the best qualities of learning system versions 2 and 3. It combines learning a direct input partitioning with cooperative response learning.

Like learning system version 3, version 4 does significantly better than versions 1 or 2 for smaller network sizes. It also appears that adding cooperative response learning has helped learning system version 4 to mitigate the difficulties that the structural credit assignment problem caused for version 3. Version 4 does at least 5% better (in terms of ultimate success rate) than version 3 in 13 of the 27 experiment sets run, nearly identically in another 13, and worse by more than 5% in only one experiment set. Nonetheless, learning system version 4 still generally does worse than learning system version 2 for larger network sizes. Further, it shows serious mid-trial crashes, with success dropping to near zero, for network sizes eight and larger on simulation case 3. There is a new problem occurring with learning system version 4, which will be returned to in Subsection 6.2.9, after we look at the results for learning system versions 5 and 6.

6.2.6 Evaluation of Learning System Version 5

Learning system version 5 is similar to version 3 in that both learn their own partitionings and do not use cooperative output learning. The difference between them is that version 5 introduces the novel idea of indexed partitioning, whereas version 3 uses direct partitioning.

Up through six neural units per dimension, learning system version 5 generally does better in all three simulation cases as network size increases. In simulation case 1 it plateaus at greater than 90% from six to ten neural units, in case 2 the plateau is rougher and lower (ranging from about 60% to over 65%), and in case 3 it peaks at 73% then declines gradually to just under 65%.

The indexed partitioning of learning system version 5 keeps the borders between the partition regions perpendicular to their coordinate axes. This prevents it from learning as useful a partitioning as learned by versions 3 and 4 for small network sizes (two and three neural units per dimension) on simulation case 1. Generally, however, learning system version 5 out does version 3 and even equals or exceeds version 4 in most cases.

Learning system version 5 also generally equals or exceeds version 1 but is more split in its success versus version 2. Version 5 does better than version 2 in the experiments with small networks in simulation cases 2 and 3, but looses out to version 2 with some large network sizes in simulation case 1 and network sizes six and eight in simulation case 3.

6.2.7 Evaluation of Learning System Version 6

Learning system version 6 uses both indexed partition learning and cooperative output learning. Of all the learning system versions, version 6 shows the smoothest progression in the quality of learning as network size is increased. It does not show the type of oscillations shown by learning system version 2 on simulation case 3. Nor does it show the precipitous drop in ultimate success rate shown by learning system versions 3 and 4 on simulation case 2. Nor does it show the mid-trial crashes of learning system version 4 on simulation case 3. In general, learning system version 6 shows an increase in ultimate success rate as network size increases up to six neural units per dimension. After that, there is a plateau extending through a network size of ten. This is true for all three simulation cases.

In terms of novelty, this version of the learning system is as far from the baseline version as we will see. In terms of performance, it is also different from the baseline version; it achieves significantly better ultimate success rates on all experiments except for very small and very large networks on simulation case 1 and the smallest network on simulation case 2. For very small networks (sizes two and three) on simulation case 1, and networks of size two on simulation case 2, both learning system version 1 and version 6 achieve less than a 5% ultimate success rate. For large networks (sizes eight through ten), version 1 scores in the low 90's while version 6 scores in the high 90's.

On simulation case 1, learning system version 6 looses out to version 2 for network sizes four and five, but otherwise their ultimate success rates are close. Nonetheless, the learning curves for version 2 of the system are better than those of version 6 for network sizes of seven and eight. On simulation case 2, learning system version 6 roundly trumps version 2 at all network sizes. It is clear from this result that the superior partitioning learned by version 6 is important in this simulation case. On simulation case 3, learning system version 6 looses out to version 2 for network sizes six and eight but otherwise learning system version 6 does significantly better.

Compared to learning system versions 3 and 4, version 6 does worse on simulation case 1 for networks of size five and smaller, but does equal or better for the larger networks. On simulation case 2, version 6 does better than versions 3 and 4 for networks of size three and larger (all versions score less than 5% for networks of size two) and on simulation case 3, version six does better than versions 3 and 4 for networks of size six or greater, roughly equaling the others for the smaller networks.

Finally, compared to learning system version 5, version 6 generally does significantly better for larger networks (sizes four and up) on simulation cases 2 and 3. For small networks on these cases, and all of simulation case 1, the performance of the systems is roughly equal, with one exception: For a network size of four neural units per dimension on simulation case 1, learning system 5 has an ultimate success rate over 55% to less than 40% for learning system 6.

6.2.8 Overall Comparison of the Learning System Versions

Looking at the ultimate success rate of each learning system version in each of our 27 experiment sets, we find that no one system always does best. In fact, the only version that does not do best in at least one experiment set is the baseline version — version 1. This is as it should be; version 1 is our starting place for creating better learning systems.

For simulation case 1, learning systems 3 and 4 are roughly equal to each other but far outperform all other versions for networks of four neural units per dimension or less. At five neural units per dimension, learning systems 3 and 4 take a back seat to version 2. At six and seven neural units per dimension it is a fairly tight race between versions 2, 4, 5, and 6, with version 1 joining the pack for eight and nine units per dimension. Significantly, learning system 4 drops out at ten neural units per dimension, leaving versions 1, 2, 5, and 6.

For simulation case 2 with two neural units per dimension, there is no learning system version that is clearly best. However, at three and four units per dimension, learning systems 5 and 6 clearly outperform the others, with learning system 6 being clearly the best version for five units per dimension and up.

For simulation case 3, learning systems 3 and 4 start off strong in the smaller networks, as they did in case 1, but by four units per network version 3 has fallen behind both version 4 and version 6. Version 4 and 6 are roughly even for five neural units per dimension but then version 4 falls out of the running as well. For six and eight neural units per dimension, learning system version 2 gets easily the best ultimate success rate but its oscillations drop it behind the consistent scores of version 6 for seven, nine, and ten neural units per dimension.

Across all simulation cases, then, learning system version 6 has the best results in terms of the number of times it does as well as, or better than, other learning system versions having networks with the same number of neural units per dimension.

What about the worst performance, in terms of ultimate success rate, as the network size increases? It would be nice if we could say that a learning system always did at least as well as the baseline system. After all, what is the use of adding complexity if nothing is gained?

For simulation case 1, the performance of learning system versions 1, 2, 5, and 6 are roughly tied for two and three neural units per dimension. For four through six units per dimension, version 1 is clearly outperformed by all five other versions. However, at seven neural units per dimension, version 1 does as well as version 3. Further, version 1 clearly outperforms version 3 for eight through ten neural units per dimension, and outperforms version 4 for ten neural units per dimension as well. At large network size, the complexity of versions 3 and 4 is not only not helping, it is hurting.

For simulation case 2, learning system versions 1 and 2 are close in performance throughout the size range, with version 1 doing better for a size of six and version 2 doing better for a size of ten. This comparison makes it clear that, without a good partitioning, the cooperative response learning is not a significant benefit.

On the other side of the coin, note that learning system version 5 does no better than version 1 for network sizes seven and eight, although it does better at other sizes. This, tends to indicate that a good partitioning is not sufficient to win out over the baseline system either. Rather, coupled with the fact that learning system version 6 does significantly better than all the others at sizes five and above, this indicates that better partitioning and cooperative response learning are both important.

Also, on simulation case 2, note that learning system versions 3 and 4 do no better than version 1 for a network size of seven, and do significantly worse for all of the larger sizes. Again, versions 3 and 4 are hurt by their complexity at larger sizes.

Finally, on simulation case 3, we see that for small networks, in this case four neural units per dimension or less, learning system versions 1 and 2 are again similar. However, at larger sizes, version 2 improves (at two sizes greatly) over version 1, while version 3 falls behind. Version 3 does significantly worse than version 1 at a size of six or more, while version 4 does significantly worse than version 1 at a size of nine and ten. Once again, versions 3 and 4 are being hurt by their complexity rather than helped.

Why, if versions 3 through 6 of the learning system all learn their own partitionings, do versions 3 and 4 suffer at larger sizes while versions 5 and 6 excel? This was explained in Subsection 6.2.4 for version 3 — learning its own input partitioning was worsening the structural credit assignment problem. However, that answer does not work as well for learning system version 4, since it learns its output responses cooperatively. Nor does it explain the crashes we saw with learning system version 4 in Figures 5.28, 5.29, and 5.30. The answer to both of these questions about learning system version 4 is that there is an important consideration that needs to be taken into account if a system combining input partition learning and cooperative response learning is to function well. This consideration is adjacency relations.

6.2.9 Adjacency Relations

With a fixed input partitioning for cooperative output learning, it is easy to ensure that output units adjacent in their topology space corresponded to adjacent input regions. With the input partition learning but no cooperative output learning, it is unimportant that such adjacency questions be addressed. When combining the systems, however, we wish to ensure that neighboring output units correspond to adjacent input regions, at least as the system begins to stabilize, so that their sharing of response lessons is reasonable. Yet the input partitioning regions themselves are being learned.

We have intentionally chosen to learn the input partition regions using a topology-preserving self-organizing method, rather than a topology-free method, such as vector quantization. However, the Self-Organizing Maps that we use to learn the input partitionings are not initially organized to reflect the topology of the input space. In general, these SOMs untangle themselves rapidly, quickly learning a gross measure of the topology of the input space, then eventually distributing their units with regard to the distribution patterns of the input data. However, the SOMs may develop twists in their arrangement in input space as they learn (see Figure 6.1). Such twists happen early in the learning of the space and generally disappear given time (see [69]). Unfortunately, we are trying to optimize with regard to learning time.

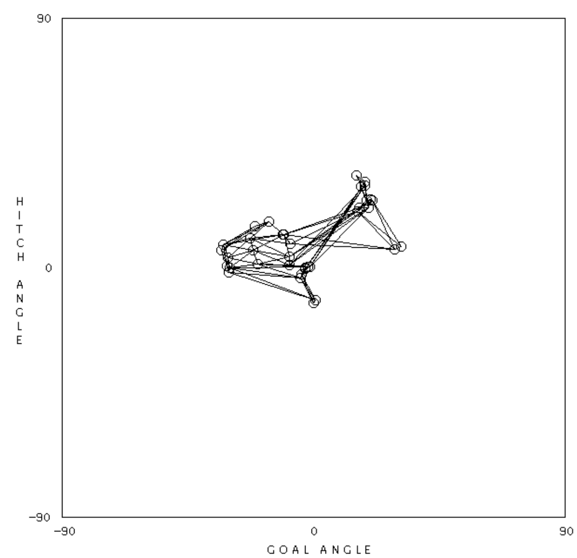


Figure 6.1: A twist in the input network after 25 trials.

If we allow the network to retain enough plasticity long enough for such twists to work themselves out, this will cause an increase in the number of learning trials before good success rates are achieved. This is because, if the regions that output units are responsible for continue to change, the relevance of their previous learning experiences declines.

If, on the other hand, the flexibility is decreased quickly so that the output units are largely responsible for the same input areas through time, the twists may be permanently embedded in the input network, and violate the adjacency relation consideration. This may also cause an increase in the number of learning trials before good success rates are achieved. This is because output units are providing their neighbors with signals that cause updates in the output weights of these neighbors. If the neighbors are not responding to input from adjacent regions of the input space, it is less likely that they should be responding similarly to their respective input. Similarly, folding or overlaps of corner regions often occur in SOMs and can cause similar difficulties for our learning system.

One advantage of learning an indexed partitioning over learning a direct partitioning, is that the indexed system is less subject to both twisting and folding. In simulation cases 1 and 2, with an indexed partitioning there are two one-dimensional maps learning the input space, and the number of units in each map is equal to the number of partitions in one dimension of the input space. With the single two-dimensional input map in cases 1 and 2, when learning a direct partitioning the number of units in the input map is equal to the *square* of the number of partitions in one dimension of the input space. This means that there are many more units in the input map of the learned direct partitioning system that need to untangle themselves before the adjacency relation consideration is met.

Note that after 250 trials in Figures 5.28, 5.29, and 5.30, learning system version 4 recovers. It is not coincidental that this is when cooperative response learning is turned off. It is clear that with 512, 729, and 1,000 neural units arranged in three dimensions, 250 trials is not sufficient to untangle the networks. Therefore, the networks have not sorted themselves out with regard to the adjacency relation and as long as cooperative response learning is taking place, progress cannot be made.

6.3 Discussion of Real-World Results

Restricting their attention to simulated systems for control has freed many researchers from having to deal with the difficulties of real robotic systems, although some researchers have tried to model sources of noise in simulation (e.g., [150]). We feel it is important to actually experiment on board real robots, however, as noise may not assume simple uniform or Gaussian distributions. Further, using real robots in testing brings out other important considerations, such as memory usage.

6.3.1 Criteria for Evaluation

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. compact enough to fit into available on-board memory, and

3. able to give responses in real time.

6.3.2 The Robotic Platform

The details of the robotic platform itself are given in Appendix B. Sources of noise in the sensors during truck-backing include:

- "Jitter" in the light-tracking goal sensor.

- The cord used for data collection passing in front of the light-tracking goal sensor.

- Non-linearity in the potentiometer built into the light-tracking goal sensor.
- Accidental dislocation of sensors on the light-tracking head.
- Non-linearity in the hitch potentiometer.
- Slippage in the hitch attachment.
- Failure to read the hitch potentiometer due to low batter levels.

Sources of noise in the execution of motor commands include:

- Bumps in the floor.
- Slippage of the wheels.
- Slowness of the turning mechanism to respond to commands.
- “Slop” in the turning mechanism.
- Tension in the cord used for data collection.
- Changing power available as the battery discharges.

Sources of noise in the initial positioning mechanism include:

- “Jitter” in both light-tracking heads.
- Accidental dislocation of sensors on either light-tracking head.
- Tension in the cord used for data collection.

6.3.3 The Computing Hardware

Our mini-robot uses a Handyboard for all running all code and storing all data for the learning system, as well as running communication code to transmit data to the workstation. Its primary limitations are its slow processor speed and its limited on-board memory. Processor speed limits what can be computed within a given amount of time. If the robot's speed has to be reduced to allow for computations, the training time may increase, even if the number of trials is reduced. To develop code that could run in real time on this hardware, we used only integers for all of our equations.

We also found it necessary to approximate the simultaneous eligibility decays of the neural units in the output networks. While the eligibility decay is defined to be a continuous exponential decay, a continual update is not necessary. Instead, it is sufficient to update a particular neural unit's eligibility level only when that neural unit is selected. The key is to use a *half-life* of eligibility for this update. The half-life of eligibility is defined to be the number of time steps it would take for any given original value of eligibility to fall approximately in half. For example, if the exponential decay rate is 0.9 then, after multiplying the current eligibility value by 0.9 for 6 time steps, one would get a value of approximately half of the original, so the half-life for this decay rate is 6.

A record is kept of the last update time for each neural unit, and when a neural unit is selected an approximation of the decay that should have taken place since the last firing is computed using the following pair of equations:

$$æ(t) = \frac{e(t-n)}{2^{(\frac{n}{L})}} \tag{6.2}$$

$$e(t) = æ(t) - \frac{æ(t)\ (n\ mod\ L)}{2L} \tag{6.3}$$

where t is the current time, $e(t)$ is the eligibility, $æ(t)$ is the approximate eligibility, n is the number of time steps since the last update, and L is the half-life for eligibility.

Use of this approximation means that only one neural unit need have its eligibility values updated on any given time step, as only a single output neural unit is selected. Since the number of neural units updated per time step remains constant (at one), regardless of network size, this adaptation (like indexed input partitioning) becomes even more useful as network size increases.

The need for real-time code was our main motivation for using eligibility traces rather than an Adaptive Heuristic Critic (AHC, see Subsection 2.1.3). AHC's, such as Barto, Sutton, and Anderson's Adaptive Critic Element [12], require exponential decays for all regions to be computed on each time step, regardless of which region contains the new input. This greatly increases the computational requirements of the algorithm. Perhaps, as computing speed continues to grow, such a computational cost will be minimal and AHC's will become practical for learning on board real robots. However, it should be noted that robots will likely to be asked to learn many things at once, as people do, so minimizing the computation required for each task may still be necessary.

The 32 kilobytes of random access memory on the Handyboard also seriously limit our algorithm choices. In fact, there was not enough memory available to run learning system versions 3 and 4 (see Table 6.1). For this reason, they were excluded from testing on the real robot.

6.3.4 Analysis of Results

The runs made on the real robot demonstrate that these learning systems are capable of learning on board a real robot despite all the noise present in the system. Most runs seem to roughly follow the average performance of the same learning system version in simulation. The large fluctuations shown in the individual graphs simply reflect the all-or-nothing measure of success inherent in our formulation of the truck-backing problem combined with the random nature of the starting positions.

The only runs in which a learning system does much better overall on the real robot than in simulation, are run two of learning system version 1 and run three of learning system version 6. However, by the end of the run 2 of version 1, the success rate has fallen back right in line with the average simulation performance and probably simply represents a random fluctuation due the initial weight values and the starting positions given. The anomaly in run three of version 6 is probably no more significant.

The only run that falls greatly below the average achieved in simulation is run three of learning system version 2. Again, we should probably not make too much of a single abnormality in such a small sample set. It is, however, interesting to compare the dip shown in Figure 5.36 with that found for the same learning system in Figure 5.10 in which the dip is found in the average results for 100 runs. Could it be that, due to a sensor miscalibration, the partitioning was off somewhat for this run on the real robot, causing it behave as the simulation version did with a poor partitioning?

6.4 Final Analysis

It is clear in the final analysis that learning system version 6, combining indexed partition learning with cooperative response learning is highly successful under a wide variety of conditions and is capable of learning on board real robotic systems. If a good partitioning is known, version 2, which uses a fixed partitioning along with cooperative response learning, is impossible to beat. It is not as widely applicable as version 6, however, since it cannot be expected to do well in cases where a good partitioning is not available beforehand.

Chapter 7

CONCLUSION

7.1 Conclusions

The problem of autonomous control learning in real robots has been addressed. Reinforcement learning allows systems to autonomously improve their performance through interactions with their environment but has previously not been directly applicable to learning in real robots. The problem of continuous input has been addressed by having the system learn a partitioning. The problem of long learning times has been ameliorated through cooperative response learning.

7.2 Contributions

The contributions of this thesis are in the development of reinforcement learning systems and their components appropriate for learning with limited feedback on board real robots.

1. The most original, and also the most successful, learning system developed is version 6, which uses the novel concepts of indexed partition learning and cooperative response learning. This learning system does consistently well over a large range of network sizes and on all simulation cases studied. It benefits both from its components and from their coherent integration.

2. Indexed partition learning itself is a significant contribution. It allows the learning system to be applied to control problems with continuous input spaces with-

out requiring that a good partitioning be known in advance, yet it minimizes the overhead required.

3. Cooperative response learning is also a significant contribution. It directly addresses the problem of long learning times by mitigating the structural credit assignment problem found in reinforcement learning systems with many partition regions. It does so by making use of the underlying continuous nature of the input space of control problems.

4. Direct partition learning for reinforcement learning using Self-Organizing Maps is also a contribution. These mappings can provide useful partitionings with a minimum of neural units.

5. The method for approximating eligibility traces greatly reduces the computational requirements of all of these systems and allows for real-time use of reinforcement learning.

6. The saturating eligibility trace provides a tool that has proven useful in developing integer versions of reinforcement learning systems.

7.3 Future Research

Future work will seek to build on this work in the following ways:

1. Find a way to combine multiple two- or three-dimensional learning systems into a single system. While it might be interesting to investigate individual learning systems with a higher number of dimensions in the output network, it should be noted that brains have been found to use numerous two- and three-dimensional sensory and motor maps which are, presumably, serving useful purposes. It

makes sense, then, to explore ways in which these maps can be combined to accomplish complex tasks.

2. Add the ability to learn continuous output. One approach that has been attempted to extend a similar reinforcement learning system is to add fuzzy decision boundaries to the partition regions after learning has been completed. Unfortunately, this has proven to be a less than satisfactory solution in some cases [164]. However, it might be possible to learn with fuzzy decision boundaries in place. The primary difficulty to learning continuous output is that, if the system fails, it is not clear in which direction to change the output. However, with fuzzy decision boundaries but underlying discrete output values, it might be possible to determine the direction for change as with the present system but use fuzzy rules to determine the extent of change. It should be noted, however, that continuous output may not be superior in all cases, even if it can be learned. Using genetic programming, the solution learned by Koza's system used hard-right or hard-left steering roughly 90% of the time, even though it was possible for it to learn intermediate values [74]. Whether this is because intermediate values are not useful in these cases or is simply an artifact of his learning system remains to be seen.

3. Add the ability to learn the network topology. While the indexed partitioning learned by the two independent networks works extremely well in the cases we studied, it is worth considering whether a different topology might not work better in some cases.

Appendix A

THE TRUCK-BACKING SIMULATION

There are two versions of the simulation; a single-trailer version and a double-trailer version. The presentation here is similar to that of Anderson and Miller [5] for a similar single-trailer simulation.

A.1 Single-Trailer Simulation

State Variables

x, y	coordinates of the rear of the trailer
θ_τ	angle of the trailer, measured from positive x with counterclockwise being positive
θ_c	angle of the truck cab, measured from positive x with counterclockwise being positive

Control

u	steering angle of front wheels relative to truck cab orientation with counterclockwise being positive, $\pm 30°$

Parameters

l_τ	length of the trailer, 11 inches
l_c	length of the cab, 7 inches
r	distance moved by the front wheels of the truck per time step, 0.25 inches
	trailer width, 4.4 inches
	target radius, 1.5 inches

Equations of Motion

$$
\begin{aligned}
A &= r\cos u(t) \\
B &= A\cos(\theta_c(t) - \theta_\tau(t)) \\
C &= A\sin(\theta_c(t) - \theta_\tau(t)) \\
x(t+1) &= x(t) - B\cos\theta_\tau(t) \\
y(t+1) &= y(t) - B\sin\theta_\tau(t) \\
\theta_c(t+1) &= \tan^-1\left(\frac{l_c\sin\theta_c(t) - r\cos\theta_c(t)\sin u(t)}{l_c\cos\theta_c(t) + r\sin\theta_c(t)\sin u(t)}\right) \\
\theta_\tau(t+1) &= \tan^-1\left(\frac{l_\tau\sin\theta_\tau(t) - C\cos\theta_\tau(t)}{l_\tau\cos\theta_\tau(t) + C\sin\theta_\tau(t)}\right)
\end{aligned}
$$

A.2 Double-Trailer Simulation

State Variables

x, y coordinates of the rear of the second trailer

θ_τ angle of the first trailer, measured from positive x with counterclockwise being positive

θ_π angle of the second trailer, measured from positive x with counterclockwise being positive

θ_c angle of the truck cab, measured from positive x with counterclockwise being positive

Control

u steering angle of front wheels relative to truck cab orientation with counterclockwise being positive, $\pm 30°$

Parameters

l_τ length of the first trailer, 16 inches

l_π length of the second trailer, 16 inches

l_c length of the cab, 8 inches

r distance moved by the front wheels of the truck per time step, 0.25 inches

trailer width, 4.4 inches

target radius, 1.5 inches

Equations of Motion

$$
\begin{aligned}
A &= r \cos u(t) \\
B &= A \cos(\theta_c(t) - \theta_\tau(t)) \\
C &= A \sin(\theta_c(t) - \theta_\tau(t)) \\
D &= B \cos(\theta_\tau(t) - \theta_\pi(t)) \\
E &= B \sin(\theta_\tau(t) - \theta_\pi(t)) \\
x(t+1) &= x(t) - D \cos \theta_\pi(t) \\
y(t+1) &= y(t) - D \sin \theta_\pi(t) \\
\theta_c(t+1) &= \tan^- 1 \left(\frac{l_c \sin \theta_c(t) - r \cos \theta_c(t) \sin u(t)}{l_c \cos \theta_c(t) + r \sin \theta_c(t) \sin u(t)} \right) \\
\theta_\tau(t+1) &= \tan^- 1 \left(\frac{l_\tau \sin \theta_\tau(t) - C \cos \theta_\tau(t)}{l_\tau \cos \theta_\tau(t) + C \sin \theta_\tau(t)} \right) \\
\theta_\pi(t+1) &= \tan^- 1 \left(\frac{l_\pi \sin \theta_\pi(t) - E \cos \theta_\pi(t)}{l_\pi \cos \theta_\pi(t) + E \sin \theta_\pi(t)} \right)
\end{aligned}
$$

Appendix B

THE TRUCK-BACKING ROBOT

The Self-Positioning Trailer-Backing Mini-Robot, SPTBMin[1], seen in Figure B.1, consists of two parts, a truck cab unit and a trailer unit. The cab is an Ackerman-steering 4-wheeled vehicle with rear-wheel differential drive. The front of the cab has a binary bumper switch that lets the robot know when it has contacted something from the front. The trailer is attached to the cab by a hitch constructed from a single-turn potentiometer. A servo with a set of photoresistors (CdS cells) is mounted on the back of the trailer so that the robot can keep track of the goal (a light bulb). The trailer also has a binary bumper switch on its rear to tell the robot when it is in contact with the goal.

The potentiometer on the hitch measures the angle that the trailer makes with the cab. Valid angles that the trailer can make are within a 180° arc due to the physical constraints of the system. The light tracking servo on the trailer is also limited to a 180° arc.

SPTBMin uses LEGO for its construction base and the Motorola 68HC11-based Handyboard microcontroller [86] as its on-board computer. LEGOs were chosen as the construction platform because of their ease of use and rapid prototyping facility. The Handyboard and Interactive-C [165] were chosen as the hardware and software components of the development environment because of their ease of programming

---

[1] Paul Rybski devised the self-positioning algorithm and designed, built, programmed, and documented SPTBMin. Most of the text and figures in this appendix are his.

Figure B.1: SPTBMin, the self-positioning trailer-backing mini-robot.

and hardware interfacing capabilities. A Miniboard [85] is connected to the Handyboard via its SPI (fast synchronous serial) port and serves as a device controller. The Miniboard manages all of the lifting mechanisms and drive motors on the robot. An additional light-tracker servo is placed on the front of the cab to assist with the self-positioning process (see Subsection B.1). Top-view diagrams of the layouts of the segments are seen in Figure B.2.

B.1 Training the Robot

The environment in which the robot conducts its experiments consists of a circular arena 6 feet in radius with a light bulb placed at the center (Figure B.3). The robot is connected to a workstation via its serial port for the purpose of data collection only. The robot can be untethered from the workstation for stand-alone operation.

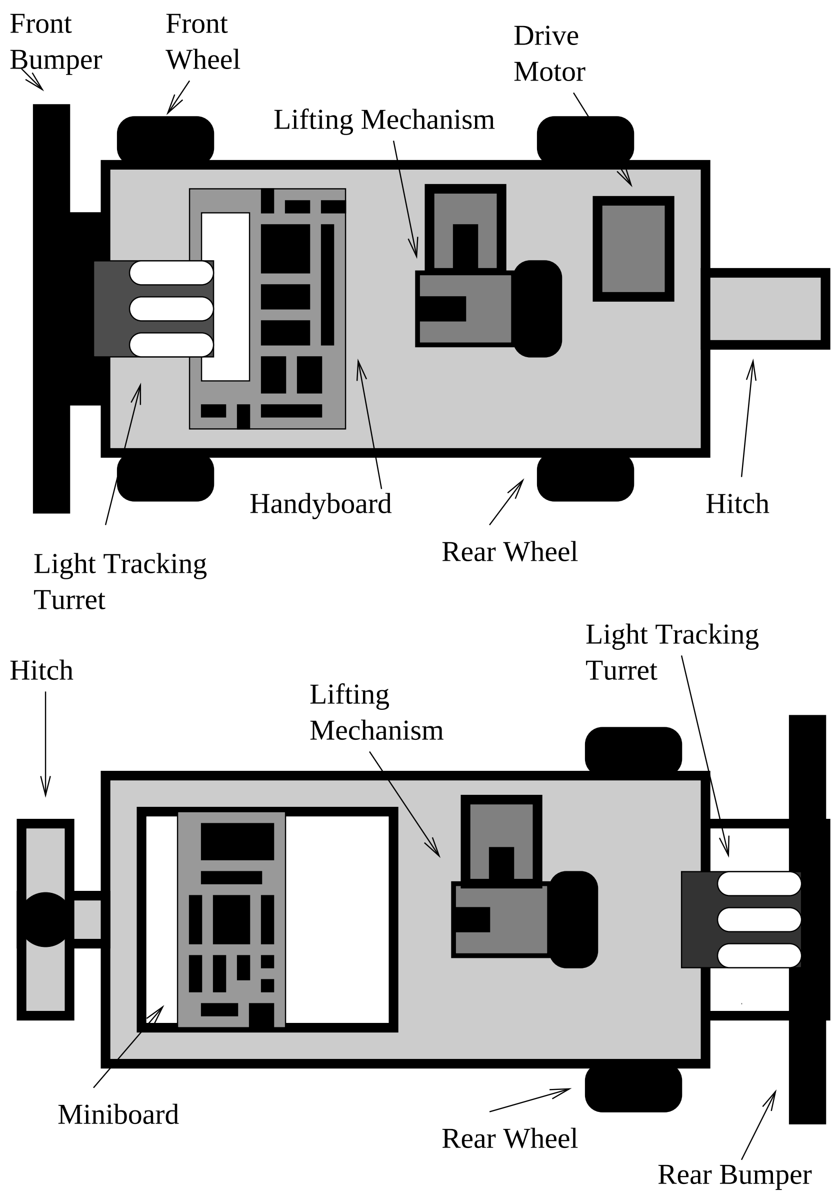


Figure B.2: Top views of the cab and trailer segments.

The entire learning system (data structures and supporting source code) fits into the on-board RAM of the Handyboard.

At the beginning of a trial, the robot starts out touching the perimeter of the arena with its front bumper. The goal-light tracker and the trailer hitch are given some initial starting angles (within a valid range) and the robot backs up, attempting to touch the goal with its rear bumper, until it either generates a failure or success signal.

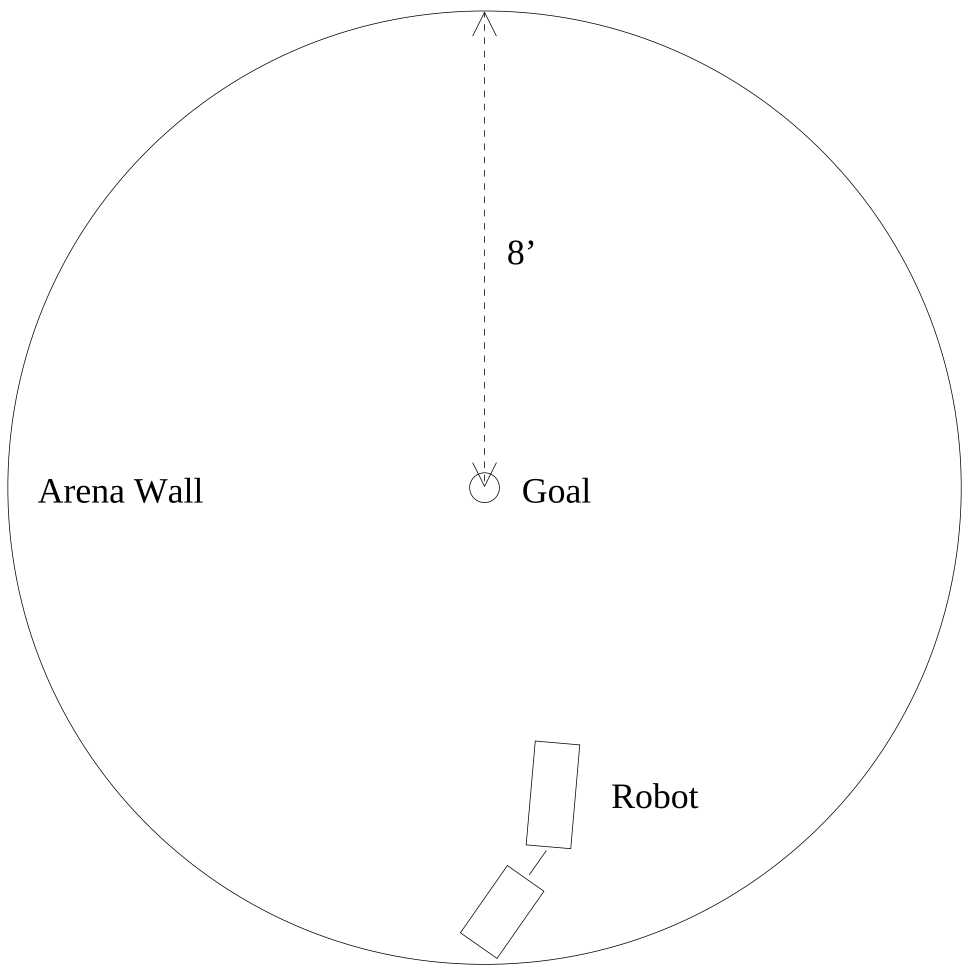


Figure B.3: The arena used for the training experiments.

The failure signal is produced when the trailer jack-knifes. This occurs when the measured hitch angle comes close to its physical limits of $\pm 90^\circ$ with respect to the cab. The robot is physically unable to recover from this condition without pulling forward to straighten itself out, and so the system halts. The failure signal is also generated when the light tracker reaches its physical limits of $\pm 90^\circ$ with respect to the trailer. This failure condition represents the case when the robot has missed the light entirely and has no hope of hitting it without pulling forward or turning around completely. Unfortunately, neither of these are guaranteed to succeed because the tracker has lost track of the light and may not necessarily re-acquire it.

The success signal is generated when the trailer's bumper touches the light bulb, since the light bulb is the only obstacle in the environment. The robot cannot contact the arena with its rear bumper without the light tracker turning to $\pm 90^\circ$ with respect to the trailer and producing the failure signal.

When either a success or failure signal is produced, the robot stops its motion and then applies the learning algorithm. After the system completes a training iteration, the robot is moved back to the edge of the arena for another run.

B.2 Self-Positioning

To position the robot at some random initial angles at the edge of the arena, each segment of the robot is outfitted with a lifting mechanism consisting of a linear actuator which lifts that segment's 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. Figures B.4 and B.5 show images of the actual robot lifting its trailer and cab segments along with diagrams showing the internal construction of the mechanism.

Using this linear actuator and drive system, each segment can pivot about a point

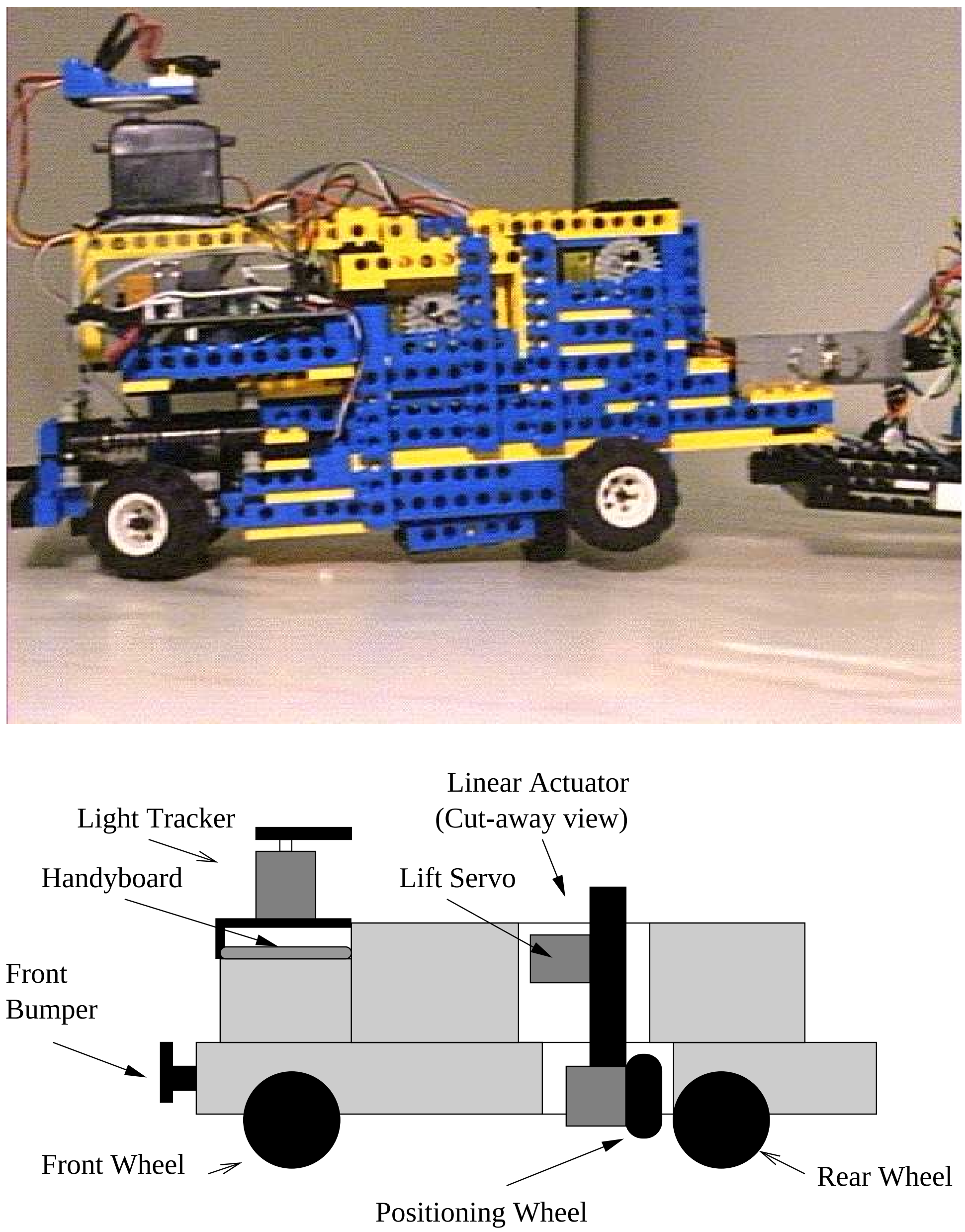


Figure B.4: Side view and cut-away descriptions of cab lifting mechanism.

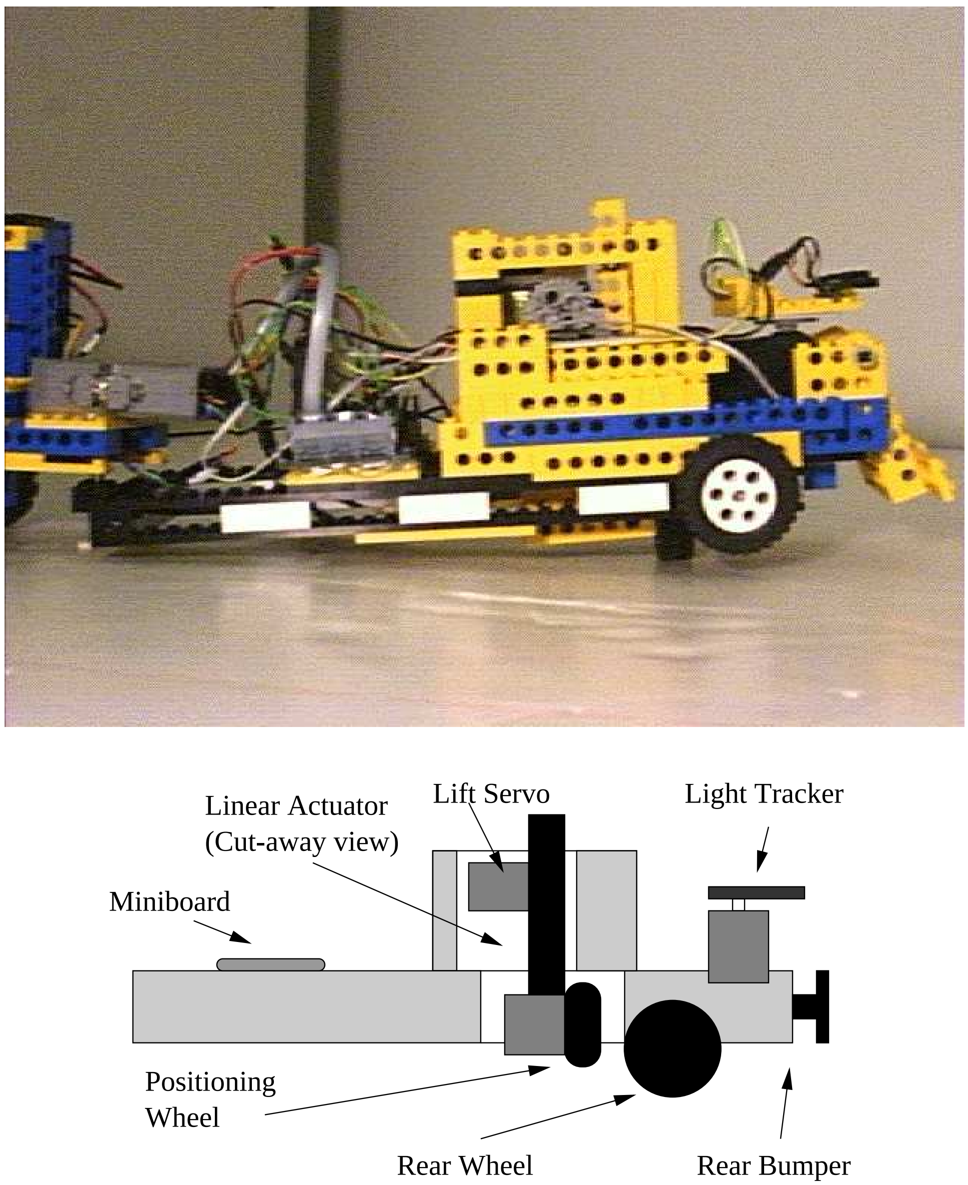


Figure B.5: Side views and cut-away descriptions of trailer lifting mechanism.

at its front. For the trailer, this point is the hitch and in the case of the cab, this point is between its front wheels. Figure B.6 shows the arc that each segment can rotate about when using the self-positioning mechanism.

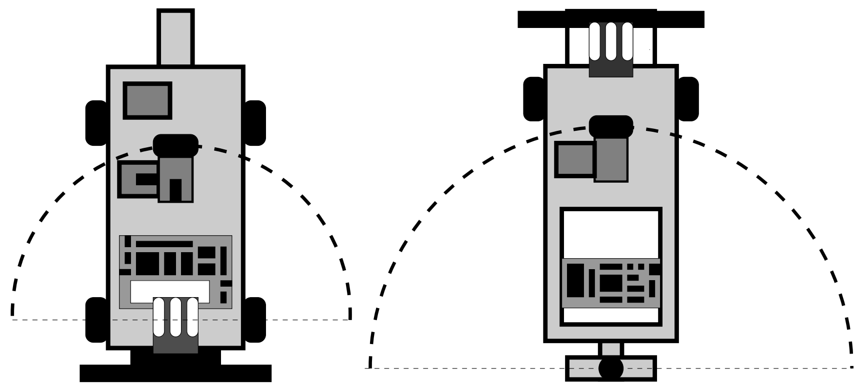


Figure B.6: The arcs that the cab and trailer segments can rotate about when using the self-positioning system.

The workstation generates a set of initial hitch and goal angles from a uniformly-distributed random number generator and downloads them to the robot.

Once the angles have been received, the robot uses them to determine how to position itself properly. While the robot is given hitch and rear light tracker angles that it must match up, its physical construction does not allow it to directly move to those angles. 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 move the trailer segment. This position is determined by the angle that the front of the cab makes to a radius line from the goal to the arena wall where the cab is touching. 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 goal angles.

B.2.1 Modeling the Robot

The physical configuration of the system is modeled in the following way to solve for the appropriate angles. Figure B.7 shows a diagram that represents the configuration of the robot with respect to the light:

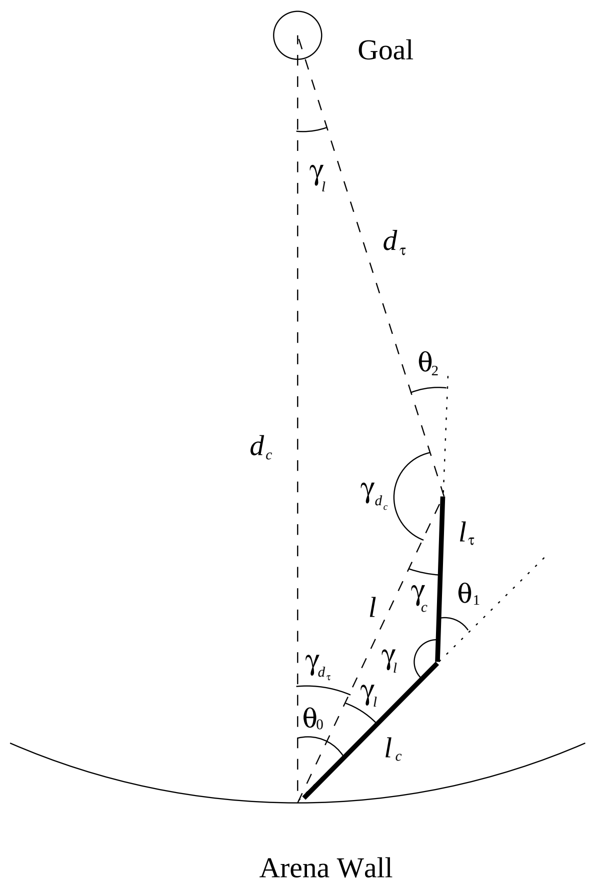


Figure B.7: Determining angles for self-positioning.

- θ_2 is the angle of the light tracker with respect to the light.
- θ_1 is the angle of the trailer with respect to the cab.
- θ_0 is the angle of the cab with respect to a radius line from the goal to the arena wall where the cab is touching. This is the value that is needed to place the

cab into its correct position.

- d_c is the distance of the front of the cab to the light.
- d_t is the distance from the back of the trailer to the light.
- l_c is the length of the cab segment.
- l_τ is the length of the trailer segment.
- l is the distance between the front of the cab and the rear of the trailer.
- γ_l, γ_{d_τ} and γ_{d_c} are the angles formed by the triangle $\triangle d_c d_\tau l$.
- γ_τ, γ_c and γ_l are the angles formed by the triangle $\triangle l_c l_\tau l$.

θ_2 and θ_1 are measured quantities which are determined when the robot stops at the wall of the arena. d_c is simply the radius of the arena (8 feet for this set of experiments). l_c and l_τ are known a priori and obviously do not change. From these known values, the remaining values can be solved for through repeated use of the law of cosines and other simple geometric formulas. Finally, the value of θ_0 can be generated.

This value represents the angle that the cab must move to with respect to tangent line to the wall of the arena. The cab's light-tracking servo head is mounted directly over its pivot point (directly between the front wheels). This 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 jack-knifes 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 tolerance, then the robot knows that an error has taken place in its self-positioning procedure and can attempt to position itself again or summon human assistance.

BIBLIOGRAPHY

[1] Shun-Ichi Amari. Topographic organization of nerve fields. *Bulletin of Mathematical Biology*, 42:339–364, 1980.

[2] Charles W. Anderson. Strategy learning with multilayer connectionist representations. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 103–114, Irvine, CA, June 1987.

[3] Charles W. Anderson. Strategy learning with multilayer connectionist representations. Technical Report TR87-509.3, GTE Labs, Waltham, MA, May 1988.

[4] Charles W. Anderson. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37, 1989.

[5] Charles W. Anderson and W. Thomas Miller, III. Challenging control problems. In W. Thomas Miller, III, Richard S. Sutton, and Paul J. Werbos, editors, *Neural Networks for Control*, pages 475–510. MIT Press, Cambridge, MA, 1990.

[6] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosada. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, pages 279–303, 1996.

[7] C. Ray Asfahl. *Robotics and Manufacturing Automation*. John Wiley and Sons, Inc., 1992.

[8] Shumeet Baluja. Evolution of an artificial neural network based autonomous land vehicle controller. *IEEE Transactions on Systems, Man, and Cybernetics*, 26, Part B(3), 1996.

[9] Jérôme Barraquand and Jean-Claude Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2328–2335, Sacramento, CA, 1991.

[10] Jérôme Barraquand and Jean-Claude Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1996.

[11] Andrew G. Barto. Connectionist learning for control. In W. Thomas Miller, III, Richard S. Sutton, and Paul J. Werbos, editors, *Neural Networks for Control*, chapter 1, pages 5–58. MIT Press, Cambridge, MA, 1990.

[12] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

[13] Nazareth S. Bedrossian. Approximate feedback linearization: The cart-pole example. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1987–1992, Nice, France, 1992.

[14] Geoge A. Bekey. The future of intelligent machines: Charting the path. *IEEE Robotics and Automation Magazine*, 4(3):12–16, 1997.

[15] Geoge A. Bekey. Needs for robotics in emerging applications: A research agenda. *IEEE Robotics and Automation Magazine*, 4(4):12–14, 1997.

[16] Geoge A. Bekey and Kenneth Y. Goldberg, editors. *Neural Networks in Robotics*. Kluwer Academic Publishers, Boston, MA, 1993.

[17] George A. Bekey. Robotics and neural networks. In Bart Kosko, editor, *Neural Networks for Signal Processing*, pages 161–187. Prentice Hall, 1992.

[18] Richard Ernest Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[19] H. Benhrahim, J.S. Doleac, J.A. Franklin, and O.G. Selfridge. Real-time learning: A ball on a beam. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 98–103, 1992.

[20] P. Bolzern, R. DeSantis, A. Locatelli, and S. Togno. Dynamic model of a two-trailer articulated vehicle subject to nonholonomic constraints. *Robotica*, 14:445–450, 1996.

[21] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. Wadsworth and Brooks, Montery, CA, 1984.

[22] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[23] Robert H. Cannon, Jr. *Dynamics of Physical Systems*. McGraw-Hill, New York, 1967.

[24] David Chapman and Leslie Pack Kaelbling. Learning from delayed reinforcement in a complex domain. Technical Report TR-90-11, Teleos Research, Palo Alto, CA, December 1990.

[25] Paul R. Cohen and Edward A. Feigenbaum. *The Handbook of Artificial Intelligence*, volume III. HeurisTech Press, Stanford, CA, 1982.

[26] Jonathan H. Connell and Sridhar Mahadevan. Introduction to robot learning. In Jonathan H. Connell and Sridhar Mahadevan, editors, *Robot Learning*, pages 1–17. Kluwer Academic Publishers, Boston, MA, 1993.

[27] Jonathan H. Connell and Sridhar Mahadevan, editors. *Robot Learning*. Kluwer Academic Publishers, Boston, MA, 1993.

[28] M. Connell and P. Utgoff. Learning to control a dynamic physical system. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 456–460, Seattle, WA, 1987.

[29] Neil E. Cotter, Thierry M. Guillerm, Jerome B. Soller, and Peter R. Conwell. Prejudicial searches and the pole balancer. In *Proceedings of the International Joint Conference on Neural Networks*, volume II, pages 689–694, 1991.

[30] Jill D. Crisman and George Bekey. Grand challenges for robotics and automation: The 1996 icra panel discussion. *IEEE Robotics and Automation Magazine*, 3(4):10–16, 1997.

[31] Elmer P. Dadios and David J. Williams. Multiple fuzzy logic systems: A controller for the flexible pole-cart balancing problem. In *Proceedings of the IEEE*

International Conference on Robotics and Automation, pages 2276–2281, Minneapolis, MN, 1996.

[32] José del R. Millán. Rapid, safe, and incremental learning of navigation strategies. *IEEE Transactions on Systems, Man, and Cybernetics*, 26, Part B(3), 1996.

[33] S. Dominic, R. Das, D. Whitley, and C. Anderson. Genetic reinforcement learning for neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 71–76, Seattle, WA, 1991.

[34] P. E. K. Donaldson. Error decorrelation: a technique for matching a class of functions. In *Proceedings III International Conference on Medical Electronics*, pages 173–178, 1960.

[35] James Doran. An approach to automatic problem-solving. In N. L. Collins and Donald Michie, editors, *Machine Intelligence 1*, pages 105–123. Oliver and Boyd, Edinburgh, 1967.

[36] Leslie Pack Kaelbling (Ed.). Special issue on reinforcement learning. *Machine Learning*, 22(1), 1996.

[37] Richard S. Sutton (Ed.). Special issue on reinforcement learning. *Machine Learning*, 8(3–4), 1992.

[38] David J. Finton and Yu Hen Hu. An application of importance-based feature extraction in reinforcement learning. In *Proceedings of the 1994 Workshop on Neural Networks for Signal Processing IV*, pages 52–60, 1994.

[39] Adriana Fiorentini, Günter Baumgartner, Svein Magnussen, Peter H. Schiller, and James P. Thomas. The perception of brightness and darkness: Relations to neuronal receptive fields. In *Visual Perception: The Neurophysiological Foundations*, chapter 7, pages 129–161. Academic Press, Inc., San Diego, 1990.

[40] Dario Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics*, 26, Part B(3), 1996.

[41] Shlomo Geva and Joaquin Sitte. Is the broom-balancer a useful test case for training methods. In *Proceedings of the IEEE International Workshop on Emerging Technologies and Factory Automation*, pages 283–289, 1992.

[42] Shlomo Geva and Joaquin Sitte. BOXES revisited. In *Proceedings of the International Conference on Artificial Neural Networks*, page 777, Amsterdam, Netherlands, September 1993.

[43] Shlomo Geva and Joaquin Sitte. A cartpole experiment benchmark for trainable controllers. *IEEE Control Systems Magazine*, pages 40–51, October 1993.

[44] Shlomo Geva and Joaquin Sitte. Performance of temporal differences and reinforcement learning in the cart-pole experiment. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2835–2838, 1993.

[45] Shlomo Geva, Joaquin Sitte, and Geoff Willshire. A one neuron truck backer-upper. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 850–856, 1992.

[46] A. Guez and J. Selinsky. A neuromorphic contoller with a human teacher. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 2, pages 595–602, 1988.

[47] A. Guez and J. Selinsky. A trainable neuromorphic controller. *Journal of Robotic Systems*, 5(4):363–388, 1988.

[48] David A. Handelman and Stephen H. Lane. *Fast sensorimotor skill acquisition based on rule-based training of neural networks*, pages 255–270. Kluwer Academic Publishers, Boston, MA, 1993. Geoge A. Bekey and Kenneth Y. Goldberg (Eds.).

[49] Janbin Hao, Shaohua Tan, and Joos Vandewalle. A rule-based neural controller for inverted pendulum system. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 534–539, 1993.

[50] J. K. Hawkins. Self-organizing systems — a review and commentary. *Proceedings of the IRE*, 49(1):31–48, January 1961.

[51] Robert Hecht-Nielsen. Counterpropagation networks. *Applied Optics*, 26(23):4979–4984, 1987.

[52] Henry Hexmoor, Lisa Meeden, and Robin R. Murphy. Is robot learning a new subfield: The robolearn–96 workshop. *AI Magazine*, 18(4):149–152, 1997.

[53] Geoffrey E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

[54] Dean F. Hougen. Use of an eligibility trace to self-organize output. In *Science of Artificial Neural Networks II, Proceedings SPIE*, volume 1966, pages 436–447, 1993.

[55] Dean F. Hougen, John Fischer, Maria Gini, and James Slagle. Fast connectionist learning for trailer backing using a real robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1917–1922, April 1996.

[56] Dean F. Hougen, John Fischer, and Deva Johnam. 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*, pages 73–80, 1994.

[57] Dean F. Hougen, Maria Gini, and James Slagle. Partitioning input space for reinforcement learning for control. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 755–760, June 1997.

[58] Dean F. Hougen, Maria Gini, and James Slagle. Rapid, unsupervised connectionist learning for backing a robot with two trailers. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2950–2955, April 1997.

[59] H. Ichihashi and M. Tokunaga. Neuro-fuzzy optimal control of backing up a trailer truck. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 306–311, 1993.

[60] John Jameson. A neurocontroller based on model feedback and the adaptive heuristic critic. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 37–44, 1990.

[61] Yashvant Jani. Application of fuzzy logic-neural network based reinforcement learning to proximity and docking operations: Special approach/docking test-case results. Technical Report NASA-CR-192294, Research Institute for Computing and Information Systems, University of Houston, Clear Lake, TX, January 1993.

[62] Robert E. Jenkins and Ben P. Yuhas. A simplified neural network solution through problem decomposition: The case of the truck backer-upper. *IEEE Transactions on Neural Networks*, 4(4):718–720, July 1993.

[63] T. Jervis and F. Fallside. Pole balancing on a real rig using a reinforcement learning controller. Technical Report CUED/F-INFENG/TR 115, Cambridge University Engineering Department, Cambridge, England, 1992.

[64] Joseph L. Jones and Anita M. Flynn. *Mobile Robots: Inspiration to Implementation.* A. K. Peters, 1993.

[65] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[66] J. A. Kangas, T. K. Kohonen, and J. T. Laaksonen. Variants of self-organizing maps. *IEEE Transactions on Neural Networks*, pages 93–99, 1990.

[67] Fred S. Keller. *Learning: Reinforcement Theory.* Random House, 1954.

[68] A. Klopf. Brain function and adaptive systems – a heterostatic theory. In *Proceedings of the International Conference on Systems, Man, and Cybernetics*, 1974.

[69] Teuvo K. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

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

[71] Teuvo K. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, September 1990.

[72] Teuvo K. Kohonen. Exploration of very large databases by self-organizing maps. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages PL1–PL6, 1997.

[73] Seong-Gon Kong and Bart Kosko. Adaptive fuzzy systems for backing up a truck-and-trailer. *IEEE Transactions on Neural Networks*, 3(2):211–223, 1992.

[74] John R. Koza. A genetic approach to finding a controller to back up a tractor-trailer truck. In *Automatic Control Conference*, pages 2307–2311, 1992.

[75] R. Matthew Kretchmar and Charles W. Anderson. Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 834–837, Houston, TX, 1997.

[76] Ulf Larsson, Caj Zeli, Kalevi Hyyppä, and Åke Wernersson. Navigating an articulated vehicle and reversing with a trailer. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2398–2404, 1994.

[77] Jean-Paul Laumond. Controllability of a multibody mobile robot. *IEEE Transactions on Robotics and Automation*, 9(6):755–763, December 1993.

[78] Chuen-Chien Lee and Hamid R. Berenji. An intelligent controller based on approximate reasoning and reinforcement learning. In *Proceedings of the IEEE International Symposium on Intelligent Control*, pages 200–205, Albany, NY, 1989.

[79] Chun-Shin Lin and Hyongsuk Kim. CMAC-based adaptive critic self-learning control. *IEEE Transactions on Neural Networks*, 2(5):530–533, September 1991.

[80] Chun-Shin Lin and Hyongsuk Kim. Use of CMAC neural networks in reinforcement self-learning control. In *Proceedings of the 1991 International Conference on Artificial Neural Networks*, pages 1285–1288, Espoo, Finland, 1991.

[81] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.

[82] Sridhar Mahadevan and Leslie Pack Kaelbling. [draft] report of the 1996 workshop on reinforcement learning. Available from URL: http://www.csee.usf.edu/ mahadeva/nsf-workshop/report/last-report.ps.gz, September 1996.

[83] Sridhar Mahadevan and Leslie Pack Kaelbling. The National Science Foundation workshop on reinforcement learning. *AI Magazine*, 17(4):89–97, 1996.

[84] Borut Maričić. Genetically programmed neural network for solving pole-balancing problem. In *Proceedings of the 1991 International Conference on Artificial Neural Networks*, pages 1273–1276, Espoo, Finland, 1991.

[85] Fred G. Martin. *The Mini Board Technical Reference*. MIT Media Laboratory, Cambridge, MA, 1995.

[86] Fred G. Martin. *The Handy Board Technical Reference*. MIT Media Laboratory, Cambridge, MA, 1998.

[87] Thomas M. Martinetz, Helge J. Ritter, and Klaus J. Schulten. 3d-neural-net for learning visuomoter-coordination of a robot arm. In *Proceedings of the International Joint Conference on Neural Networks*, volume II, pages 351–356. Erlbaum, 1989.

[88] Thomas M. Martinetz and Klaus J. Schulten. A neural network for robot control: Cooperation between neural units as a requirement for learning. *Computers and Electrical Engineering*, 19(4):315–332, 1993.

[89] J. M. Mendel and R. W. McLaren. Reinforcement-learning control and pattern recognition systems. In J. M. Mendel and K. S. Fu, editors, *Adaptive, Learning and Pattern Recognition Systems*, chapter 8, pages 287–318. Academic Press, New York, 1970.

[90] Donald Michie and R.A. Chambers. Boxes: an experiment in adaptive control. In E. Dale and Donald Michie, editors, *Machine Intelligence*. Oliver and Boyd, Edinburgh, 1968.

[91] Donald Michie and R.A. Chambers. 'boxes' as a model of pattern-formation. In C. H. Waddington, editor, *Towards a Theoretical Biology: 1. Prolegomena*, pages 206–215. Edinburgh University Press, Edinburgh, 1968.

[92] W. Thomas Miller, III, Richard S. Sutton, and Paul J. Werbos. *Neural Networks for Control*. MIT Press, Cambridge, MA, 1990.

[93] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, January 1961.

[94] Tom M. Mitchell. *Machine Learning*. WCB/McGraw-Hill, Boston, MA, 1997.

[95] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[96] Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:199–233, 1995.

[97] David E. Moriarty and Risto Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32, 1996.

[98] Michiyo Nakamura and Shin'ici Yuta. Trajectory control of trailer type mobile robots. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2257–2263, Yokohama, Japan, July 1993.

[99] Derrick Nguyen and Bernard Widrow. The truck backer-upper: an example of self-learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume II, pages 357–363. Erlbaum, 1989.

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

[101] Derrick Nguyen and Bernard Widrow. Neural networks for self-learning control systems. *International Journal of Control*, 54(6):1439–1451, 1991.

[102] Nils J. Nilson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1980.

[103] Katsuhiko Ogata. *System Dynamics*. Prentice-Hall, Englewood Cliffs, NJ, 1978.

[104] Katsuhiko Ogata. *Discrete-Time Control Systems*. Prentice-Hall International, London, 1995.

[105] Sigeru Omatu, Marzuki Khalid, and Rubiyah Yusof. *Neuro-Control and its Applications*. Springer, London, 1996.

[106] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, Reading, MA, 1989.

[107] Ramon Parra-Loera and David J. Corelis. Expert system controller for backing-up a truck-trailer system in constrained space. In *Proc. of the 37th Midwest Symposium on Circuits and Systems*, pages 1357–1361, 1995.

[108] Ajay Patrikar and John Provence. A self-organizing controller for dynamic processes using neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 359–364, 1990.

[109] Sameer M. Prabhu and Devendra P. Garg. Artificial neural network based robot control: An overview. *Journal of Intelligent and Robotic Systems*, 15:333–365, 1996.

[110] John Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[111] John Ross Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[112] H. Ritter, T. Martinetz, and K. Schulten. Topology conserving maps for learning visio-motor-coordination. *Neural Networks*, 2(3), 1989.

[113] H. Ritter, T. Martinetz, and K. Schulten. *Neural computation and self-organizing maps: an introduction*. Addison-Wesley, Reading, MA, 1992.

[114] H. Ritter, T. Martinetz, and K. Schulten. *Neural computation and self-organizing maps: an introduction*. Addison-Wesley, Reading, MA, 1992.

[115] H. Ritter and K. Schulten. Extending Kohonen's self-organizing mapping algorithm to learn balistic movements. In R. Eckmiller and C. von der Malsberg, editors, *Neural Computers*, volume F41, pages 393–406. Springer, Heidelberg, 1987.

[116] Bruce E. Rosen, James M. Goodwin, and Jacques J. Vidal. Machine operant conditioning. In *Proceedings of the IEEE Engineering in Medicine and Biology Society 10th Annual International Conference*, 1988.

[117] Bruce E. Rosen, James M. Goodwin, and Jacques J. Vidal. State recurrence learning. In *INNS First Annual Meeting: Advance Program*, page 48, Boston, MA, 1988.

[118] Bruce E. Rosen, James M. Goodwin, and Jacques J. Vidal. Process control with adaptive range coding. *Biological Cybernetics*, 66(5):419–428, 1992.

[119] Brian H. Rudall. Reoprts and surveys: World industrial robots — statistical data to 1998. *Robotica*, 15(2):247–249, 1997.

[120] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McCelland, and the PDP Research Group, editors, *Parallel Distributeed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations, chapter 8, pages 318–362. MIT Press, Cambridge, MA, 1986.

[121] David W. Russell. Further studies in AI-augmented process control using the BOXES methodology. In *Artificial Intelligence in Real-Time Control: Proceedings of the Third IFAC Workshop*, pages 75–79, Sonoma Valley, CA, September 1991. Proceedings reprinted as *Annual Review in Automatic Programming*, Volume 16, Part I, Pergamon Press: Oxford, 1992.

[122] David W. Russell. AI-augmented goal achievement. In *Proceedings of the Seventh International Conference on Applications of Artificial Intelligence in Engineering*, pages 971–982, Waterloo, Canada, July 1992.

[123] David W. Russell. A critical assessment of the BOXES paradigm. *Applied Artificial Intelligence*, 7:383–395, 1993.

[124] David W. Russell and S. J. Rees. System control — a case study of a statistical learning automaton. In Robert Trappl and F. de P. Hanika, editors, *Progress in Cybernetics and Systems Research*, volume II. John Wiley and Sons, New York, 1975.

[125] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1995.

[126] Marcos Salganicoff, Lyle H. Ungar, and Ruzena Bajcsy. Active learning for vision-based robot grasping. *Machine Learning*, 23:251–278, 1996.

[127] C. Sammut. Experimental results from an evaluation of algorithms that learn to control dynamic systems. In *Proceedings of the International Conference on Machine Learning*, pages 437–443, San Mateo, CA, 1988. Morgan Kaufman.

[128] A.L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.

[129] A.L. Samuel. Some studies in machine learning using the game of checkers. II — recent progress. *IBM Journal of Research and Development*, 11(6):601–617, November 1967.

[130] John F. Schafer and Robert H. Cannon, Jr. On the control of unstable mechanical systems. In *Automatic and Remote Control III: Proceedings of the Third Congress of the International Federation of Automatic Control*, page Paper 6C, 1966.

[131] S. Sekhavat and J.P. Laumond. Topological property of trajectories computed from sinusoidal inputs for nonholonomic chained form systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3383–3388, Minneapolis, MN, April 1996.

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

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

[134] Herbert A. Simon. Why should machines learn? In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Michell, editors, *Machine Learning: An Artificial Intelligence Approach*, chapter 2, pages 25–37. Tioga Publishing Company, Palo Alto, CA, 1983.

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

[136] Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[137] O.J. Sordalen. Conversion of the kinematics of a car with /em n trailers into a chained form. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1993.

[138] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[139] Richard S. Sutton and Andrew G. Barto. Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review*, 88(2):135–170, 1981.

[140] Petr S vestka and Jules Vleugels. Exact motion planning for tractor-trailer robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2445–2450, 1995.

[141] Kazuo Tanaka. Advanced fuzzy control of a trailer type mobile robot: Stability analysis and model-based fuzzy control. In *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence*, pages 205–211, November 1994.

[142] Kazuo Tanaka. A robust stabilization problem of fuzzy control systems and its application to backing up control of a truck-trailer. *IEEE Transactions on Fuzzy Systems*, 2(2):119–134, 1994.

[143] Kazuo Tanaka. Model-based fuzzy control of a trailer type mobile robot. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, pages 65–70, 1995.

[144] Chen K. Tham and Richard W. Prager. Reinforcement learning for multi-linked manipulator control. Technical Report CUED/F-INFENG/TR 104, Cambridge University Engineering Department, Cambridge, UK, June 1992.

[145] Chen K. Tham and Richard W. Prager. Reinforcement learning for multi-linked manipulator control. In *Proceedings of the Sixth International Conference on Systems Research, Informatics and Cybernetics*, Baden-Baden, Germany, August 1992.

[146] D. Tilbury, J-P. Laumond, R. Murray, S. Sastry, and G. Walsh. Steering car-like systems with trailers using sinusoids. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1993–1998, Nice, France, May 1992.

[147] Dawn Tilbury, Ole Jakob Sordalen, Linda Bushnell, and S. Shankar Sastry. A multisteering trailer system: Conversion into chained form using dynamic

feedback. *IEEE Transactions on Robotics and Automation*, 11(6):807–818, December 1995.

[148] V. Tolat and B. Widrow. An adaptive "broom balancer" with visual inputs. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 2, pages 641–647, San Diego, CA, 1988.

[149] Julius T. Tou and Rafael C. Gonzalez. *Pattern Recognition Principles.* Addison-Wesley, Reading, MA, 1974.

[150] T. Troudet and W. Merrill. Neuromorphic learning of continuous-valued mappings from noise-corrupted data. *IEEE Transactions on Neural Networks*, 2(2):294–301, 1991.

[151] John G. Truxal. Computers in automatic control systems. *Proceedings of the IRE*, 49(1):305–312, January 1961.

[152] Marc M. Van Hulle. Topology-preserving map formation achieved with a purely local unsupervised competetive learning rule. *Neural Networks*, 10(3):431–446, April 1997.

[153] V. Rao Vemuri, editor. *Artificial Neural Networks: Concepts and Control Applications.* IEEE Computer Society Press, Los Alamitos, CA, 1992.

[154] G. Walsh, D. Tilbury, S. Sastry, R. Murray, and J.P. Laumond. Stabalization of trajectories for systems with nonholonomic constraints. *IEEE Transactions on Automatic Control*, 39(1):216–222, January 1991.

[155] Gou-Jen Wang and Denny K. Miu. Unsupervising adaption neural-network control. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 421–428, 1990.

[156] Philip D. Wasserman. *Neural Computing: Theory and Practice*. Van Nostrand Reinhold, New York, 1989.

[157] Bernard Widrow. Adaptive inverse control. In *Adaptive Systems in Control and Signal Processing, 1986: Proceedings of the 2nd IFAC Workshop*, pages 1–5, Lund, Sweden, 1986.

[158] Bernard Widrow. The original adaptive neural net broom-balancer. In *International Symposium on Circuits and Systems*, pages 351–357, 1987.

[159] Bernard Widrow, Narendra K. Gupta, and Sidhartha Maitra. Punish/reward: learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(5):455–465, 1973.

[160] Bernard Widrow and Michael A. Lehr. 30 years of adaptive neural networks: perceptron, madaline and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, September 1990.

[161] Bernard Widrow and Fred W. Smith. Pattern-recognizing control systems. In *COINS (Computer and Information Sciences Symposium)*, pages 288–317, Washington, DC, 1964.

[162] Bernard Widrow and Eugene Walach. *Adaptive Inverse Control*. Prentice Hall, Upper Saddle River, NJ, 1996.

[163] Victor Williams and Kiyotoshi Matsuoka. Learning to balance the inverted pendulum using neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 214–219, 1991.

[164] N. Woodcock, N. J. Hallam, and P. D.Picton. Fuzzy BOXES as an alternative to neural networks for difficult control problems. *Artificial Inteligence in Engineering*, pages 903–919, 1991.

[165] Anne Wright, Randy Sargent, and Carl Witty. *Interactive C User's Guide*. Newton Research Labs, Cambridge, MA, 1996.

[166] A. M. S. Zalzala and A. S. Morris, editors. *Neural Networks for Robotic Control*. Ellis Horwood, London, 1996.