TECHNICAL UNIVERSITY OF CRETE

Learning Continuous-Action Control Policies

by

Jason Pazis

Thesis committee:
Michail G. Lagoudakis, Supervisor
Michalis Zervakis
Nikos Vlassis

A thesis submitted in partial fulfillment
for the Diploma degree

in the
Department of Electronic and Computer Engineering

October 13, 2008

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Μάθηση Πολιτικών Ελέγχου με Συνεχείς Χώρους Ενεργειών

Ιάσων Πάζης

Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών

Abstract

Reinforcement Learning for control in stochastic processes has received significant attention in the last few years. Several data-efficient methods, even for continuous state spaces, have been recently proposed, however most of them assume a small and discrete action space. While continuous action spaces are quite common in real-world problems, the most common approach still employed in practice is crude discretization of the action space. This thesis presents a novel, computationally-efficient method for realizing continuous-action policies, using binary decisions corresponding to adaptive increment or decrement steps of the continuous action variables. The proposed approach essentially approximates any continuous action space to arbitrary resolution and can be combined with any known discrete-action reinforcement learning algorithm for learning continuous-action policies. It is coupled with three well-known reinforcement learning algorithms (LSPI, Q-learning, and Fitted Q-iteration) and its use and properties are thoroughly investigated and demonstrated on the Inverted Pendulum and the Bicycle domains.

Abstract

Τα τελευταία χρόνια έχει αφιερωθεί σημαντική ερευνητική δραστηριότητα στην ενισχυτική μάθηση για εφαρμογές ελέγχου σε στοχαστικές διεργασίες. Αν και έχουν προταθεί πολλές μέθοδοι που επιτυγχάνουν αποτελεσματική χρησιμοποίηση των δεδομένων μάθησης, ακόμα και για συνεχείς χώρους καταστάσεων, οι περισσότερες προϋποθέτουν έναν μικρό, διακριτό χώρο ενεργειών. Παρόλο που οι συνεχείς χώροι ενεργειών είναι συνήθεις σε πολλά προβλήματα πρακτικού ενδιαφέροντος, η πιο κοινή τεχνική αντιμετώπισής τους εξακολουθεί να παραμένει η πρόχειρη διακριτοποίηση του χώρου σε λίγες δυνατές ενέργειες. Στην παρούσα διπλωματική εργασία παρουσιάζεται μια προτότυπη, υπολογιστικά αποδοτική μέθοδος για την υλοποίηση πολιτικών ελέγχου με συνεχείς χώρους ενεργειών, χρησημοποιώντας δυαδικές αποφάσεις για την προσαρμοστική αύξηση ή μείωση των τιμών των συνεχών μεταβλητών ελέγχου. Η προτεινόμενη μέθοδος ουσιαστικά προσεγγίζει οποιοδήποτε συνεχή χώρο ενεργειών σε αυθαίρετο βαθμό ακριβείας και μπορεί να συνδυαστεί με όλους τους γνωστούς διακριτούς αλγορίθμους ενισχυτικής μάθησης για τη μάθηση πολιτικών ελέγχου με συνεχείς ενέργειες. Η μέθοδος συνδυάζεται με τρεις γνωστούς αλγορίθμους ενισχυτικής μάθησης (LSPI, Q-learning, και Fitted Q-iteration) και η χρήση και οι ιδιότητές της ερευνούνται σε βάθος και επιδεικνύονται στα προβλήματα του ανάστροφου εκκρεμούς και της ισορρόπησης και οδήγησης ποδηλάτου.

Acknowledgements

This is usually one of the first sections to read in a thesis. It is also among the last to write. For the author, it is a bit like looking back and recalling the people and events that led to this point.

The people that have helped and influenced me, throughout the last few years are far too many to mention by name. Nevertheless, I thank them all, those that are still here and those that have left to pursue their own dreams.

First of all, I would like to thank my project advisor, Michail G. Lagoudakis, for guiding me and believing in me. He is the main reason I became interested in the area of Reinforcement Learning and has always been there, to provide valuable insight to ideas, others would have not so open mindedly considered. He is much more than a usual professor, caring deeply about his students as people and teaching more than a narrow field of science.

Throughout my years at the Technical University of Crete, a number of professors have influenced my decisions and challenged me to think "outside the box". Their ideas and advice have been a tremendous help, not only when we agreed, but also when we held opposite views.

Of course no one can go on this journey alone. I owe a debt of gratitude to all my friends over the years, who have been with me, through my best and worst. I would not be the same person, if it wasn't for them. What I've learned from them, cannot be taught from books, or expressed in words.

Contents

List of Figures

Chapter 1

Introduction

Autonomous agents that can act and learn from experience in the real world, has been the subject of scientific research for many years. It has been a topic of human imagination even longer. Despite the obvious appeal of the field, it is only recently that advances have allowed us to produce results that could be usefull in a real wold setting. This is directly related to the complexity of the task.

Reinforcement Learning is an approach to the problem, that makes limited assumptions about available knowledge. Reinforcement Learning for control is a relatively young and promising field. It has received great interest in the last few years, with researchers bringing ideas from related fields and trying to solve problems previously only seen in optimal control theory.

Despite its tremendous success in laboratory settings, Reinforcement Learning is yet to reach the quality of solutions required by commercial standards. When faced with the real world and its complexity, the performance of many theoretical solutions degrades significantly, or even breaks down altogether. Tackling complexity in real world problems, in order to make learning possible and efficient, will be the main focus throughout this thesis.

1.1 Decisions, Control and the Real World

Most examples in introductory texts on Reinforcement Learning, start with simple discrete models of a world, with a small number of states, one of them being the "winning" state and another being the "losing" state. The agent has a small number of actions available to him, in order to solve or "navigate" on the problem.

For example in a maze world, a possible set of actions could be: move up, move down, move left, or move right. The state of the world would then be the current position of the agent and the goal a particular block that when reached yields a reward (a chest of gold coins). Our problem is then to find a *policy* that maps every possible state (position) to an action.

When the number of states and actions is limited, finding a successfull policy is relatively easy. If we visit each state a number of times and record the results of each action, we can find a mapping of states to actions that brings us to the goal, while avoiding undesired states. As the number of states and actions increases, this simple approach is no longer applicable. Visiting every possible state and trying every possible action, could take a prohibitive amount of time. Even if we manage to gather all this information, processing and storing it becomes a problem.

Even from this simple example, it becomes obvious that exact solutions come at a cost, both in computational time and storage space. When the number of states becomes too large, exact solutions are abandoned and approximate methods are employed. The obvious drawback of this approach is that approximation may lead to errors and suboptimal decisions.

Despite the dangers associated with using approximate methods, there are a number of benefits that makes their use worthwile:

- The first and most obvious benefit is the decrease in the computational resources required.
- Storing and transferring the policy, is now a matter of storing a small number of parameters, rather than a large amount of tabular data.
- One positive side effect of using approximate methods, is their ability to generalize. A number of samples at one part of the state space, may provide insight at other parts of the state space where samples are scarce. This, of course, can equally be a drawback, when generalization leads to mistakes.

When the state space is continuous, approximation is a necessity, since there is no tabular way to associate an action with every possible combination of values of the state variables.

Current approximation methods, have reached a point where they work quite well for a number of continuous variables in the state space. The same cannot be said in cases where the actions are continuous.

Approximating continuous state variables, works because for most problems, similar states share properties that require similar actions to be performed. On the other hand, action spaces do not nessesarily behave well enough, to allow an efficient interpolation amongst the value function (activation) of each discrete action.

1.2 Discrete agents in a continuous world

Events around us are continuous[1]. The force applied by a muscle, the speed of a moving car, or the position of an object relative to a coordinate system, are described by continuous variables. This is obvious to a human, an animal, or even an insect. Computers on the other hand are digital. Algorithms available in Reinforcement Learning are able to make a decision amongst discrete actions and run into trouble when the number of actions grows beyond a certain point.

A number of efforts have been made in the context of Reinforcement Learning to find a way around this problem. Some have had more success than others, yet none of the available methods can be characterized as a comprehensive solution.

Variations of this problem is found in a number of different disciplines and has been addressed with varying degrees of success. Some examples familiar to most of us are:

- given an analog voice signal, what is the most efficient method to transmit it over a digital communications channel?
- given an audio recording, how can we store it with the minimum possible loss of quality, while at the same time using storage space effectively?
- given a picture, how can we utilize the similarities between pixels to achieve a more compact representation?
- given a video recording, how can the similarity between successive frames be utilized in order to achieve compression?

This list could go on for pages, but the point would be the same. What is the common property all these problems share? The answer to these questions and others, most, if not all of the time, utilizes some form of locality (temporal, spacial). Locality is a recurring theme in many scientific disciplines and harnessing its properties can offer some very significant advantages.

[1]The theory of Quantized Universe debates this, but for practical purposes, most events can be considered to be taking place in continuous space and time.

1.3 Contribution

This thesis addresses the problem of learning successfull control policies, in domains where actions (and state) are continuous. Efficiency and scalability are carefully considered during every step of the process. In particular a constant time, sample efficient algorithm is contributed, targeted for control problems where temporal locality is a common characteristic.
The proposed algorithm has a number of desirable properties that make it very suitable for use in real, low cost, high performance systems:

- Can be combined with any Reinforcement Learning algorithm (or other form of learning) that uses discrete decisions.
- Does not depend on the form of approximation used for the state variables.
- Comes with very low computational requirements. A small number of additions, subtractions, shift operations, and logical statements are all that is required. This makes it perfect for low cost microcontrollers with limited resources.
- Exhibits a "minimum required effort" behavior, making it a good choice for applications where it is importand not to waste resources, such as mobile robotics.

1.4 Thesis outline

Chapter 2 provides the necessary background of the ideas this thesis builds upon. A brief introduction is given to Reinforcement Learning, Markov Decision Processes and Policies. Q-learning, a classical Reinforcement Learning algorithm for control, along with Fitted Q iteration, a new, batch version of this successfull algorithm are also visited. LSPI, the main Reinforcement Learning algorithm used in the experiments throughout this thesis, is summarized. The basic principles behind Adaptive Delta Modulation, the algorithm used in telecommunication systems that provided the inspiration for our approach, are explained.

Chapter 3 discusses the motivation behind this work. The need for continuous actions is explained, along with a summary of related work in the area. The strengths and weaknesses of existing approaches are discussed, along with usual practices.

Chapter 4 introduces the Continuous Action Control algorithm, a novel approach to realizing policies in continuous action domains. Its efficiency is analyzed and its scalability is discussed. Finally a number of practical considerations is taken into account.

Chapter 5 demonstrates the applicability of the new algorithm, in the Inverted Pendulum and the Bicycle Domains. It is coupled with LSPI, Q-learning and Fitted Q-iteration.

Finally, Chapter 6 discusses the strengths and weaknesses of the approach, gives a number of guiding directions for future work, and concludes this thesis with a brief summary.

Chapter 2

Background

2.1 Agents and Environments

An agent can be anything that has the ability to perceive some aspect(s) of its environment and act. The environment is what our agent perceives as the "outside" world. It can be the real world, a room, a simulated labyrinth or even an actuator. Even though something like an actuator would appear to be part of the agent and not the environment, an agent will usually have no prior knowledge about the results of his own actions. Therefore he has to treat everything as part of the environment and try to learn how to interact with it, in a beneficial way.

Environments are in most cases stochastic. What this means for our agent, is that even when the environment is the same, an action may have different outcomes. For example the action of tossing a coin, may produce heads or tails.

2.2 Markov Decision Process

A Markov Decision Process (MDP) is a discrete-time mathematical decision-making modeling framework, particularly usefull when the outcome of a process is in part a result of the agents actions and in part random. They have found extensive use in areas such as economics, control, manufacturing, and Reinforcement Learning.

An MDP can be described as a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where:

- $\mathcal{S} = \{s_1, s_2, ..., s_n\}$ is the (finite) state space of the process. The state s is a description of the status of the process at a given time.

- $\mathcal{A} = \{a_1, a_2, ..., a_m\}$ is the (finite) action space of the process. The set of actions are the possible choices an agent has at a particular time.
- $\mathcal{P}$ is a Markovian transition model, where $\mathcal{P}(s, a, s')$ is the probability of making a transition to state s' when taking action a in state s. A Markovian transition model, means that the probability of making a transition to state s' when taking action a in state s, depends only on s and a and not on the history of the process.
- $\mathcal{R}$ is the reward function (scalar real number) of the process. It is Markovian as well and can be the immediate, or the expected immediate reward (for stochastic rewards) at each time step. The expected reward for a state-action pair (s, a), is defined as:

$$\mathcal{R}(s, a) = \sum_{s' \in S} \mathcal{P}(s, a, s')\mathcal{R}(s, a, s') \ .$$

An MDP is often augmented to include γ and $\mathcal{D}$ as $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{D})$, where:

- $\gamma \in (0, 1]$ is the discount factor. When $\gamma = 1$ a reward retains its full value independently of when it is received. As γ becomes smaller, the importance of rewards in the future is diminished exponentially by γ^t.
- $\mathcal{D}$ is the initial state distribution. It describes the probability that each state in $\mathcal{S}$ will be the initial state. On some problems most states have a zero probability, while few states (possibly only one) are candidates for being an initial state.

The optimization objective in an MDP is the maximization (or minimization depending on the problem) of the expected total discounted reward, which is defined as:

$$E_{s \sim \mathcal{D} \ ; \ a_t \sim ? \ ; \ s_t \sim \mathcal{P}} \left(\sum_{t=0}^{\infty} \gamma^t r_t \ \middle| \ s_0 = s \right) \ .$$

2.3 Policies

A policy π is a mapping from states to actions. It defines the responce (which may be deterministic or stochastic) of an agent in the environment for any state and it is sufficient to completely determine its behavior. In that sense $\pi(s)$ is the action chosen by the agent following policy π.

An optimal policy π^* also known as an "undominated optimal policy" is a policy that yields the highest expected utility. That is, it maximizes the expected total discounted reward under all conditions (over the entire state space). For every MDP there is at least

one such policy although it may not be unique (multiple policies can be undominated; hence yielding equal expected total discounted reward through different actions).

The state-action value function $Q^\pi(s,a)$ for a policy π is defined over all possible combinations of states and actions and indicates the expected, discounted, total reward when taking action a in state s and following policy π thereafter:

$$Q^\pi(s,a) = E_{a_t \sim \pi \,;\, s_t \sim \mathcal{P}} \left(\sum_{t=0}^{\infty} \gamma^t r_t \;\middle|\; s_0 = s,\ a_0 = a \right) .$$

2.4 Reinforcement Learning

Reinforcement Learning is learning in an environment by interaction [1–3]. It is usually assumed that the agent knows nothing about how the environment works (has no model of the underlying MDP) or what the results of its actions are. In addition the environment can be stochastic, yielding different outcomes for the same situation. In contrast to supervised learning there is no teacher to provide examples of correct or bad behavior. It is very similar to unsupervised learning, except that one of the percepts is "hardwired" to be recognized as a reward.

The goal of an agent in such a setting is to learn from the consequences of its actions, in order to maximize the total reward over time. Rewards are in most cases discounted, in the sense that a reward early in time is "more valuable" than a reward later. This is done mainly in order to give an incentive to the agent, to go towards a solution fast, rather than wasting time in areas of the state space where there is no large negative reward.

Two related problems fall within Reinforcement Learning: Prediction and Control. In prediction problems, the goal is to learn to predict the total reward for a given policy, whereas in control the agent tries to maximize the total reward by finding a good policy. These two problems are often seen together, when a prediction algorithm evaluates a policy and a control algorithm subsequently tries to improve it.

The learning setting is what characterizes the problem as a Reinforcement Learning problem. Any method that can successfully reach a solution, is considered as a Reinforcement Learning method. This means that very diverse algorithms coming from different backgrounds can be used; and that is indeed the case. Most of the approaches can be distinguished into Model Based learning and Model Free learning.

In Model Based learning, the agent uses its experiences in order to learn a model of the process and then find a good decision policy through planning. Model Free learning

on the other hand tries to learn a policy directly without the help of a model. Both approaches have their strengths and drawbacks (guarantee of convergence, speed of convergence, ability to plan ahead, use of resources).

2.5 Q-learning

Q-learning [4] is a model-free, off-policy, Reinforcement Learning control algorithm that can be used in an online or off-line setting. It uses samples of the form (s, a, r, s') to find a state-action value function. The simple temporal difference update equation for Q-learning is:

$$Q(a, s) = Q(a, s) + \alpha(\mathcal{R}(s) + \gamma \max_{a'} Q(a', s') - Q(a, s))$$

Essentially it is an incremental version of dynamic programming techniques, which imposes limited computational demands at every step, with the drawback that it usually takes a large number of steps to converge. It learns an action-value representation, allowing the agent using it to act optimally (under certain conditions) in domains satisfying the Markov property.

2.6 Fitted Q Iteration

Fitted Q Iteration [5], is a batch training version of the popular Q-learning algorithm. It uses the full sample set, to create desired input-output pairs which are later used by a supervised learning algorithm to approximate the state-action value function. Figure 2.1 summarizes the Fitted Q algorithm.

2.7 LSPI

LSPI [6] (Least-Squares Policy Iteration) is a relatively new, model-free, approximate policy iteration Reinforcement Learning algorithm for control. It is an offline, off-policy, batch training method that exhibits good sample efficiency and offers stability of approximation. LSPI has met great success in the last few years, applied in domains with continuous or discrete states and discrete actions. In the heart of the LSPI algorithm lies LSTDQ. LSTDQ learns the weighted least-squares fixed-point approximation of the state-action value function of a fixed policy π, by solving the $(k \times k)$ linear system

$$\mathbf{A}w^{\pi} = b \,,$$

Fitted Q $(D,\ N)$

// D : Source of samples (s, a, r, s')
// N : Number of iterations

$k \leftarrow 0$
$\hat{Q}_k \leftarrow 0$

repeat
Build the training set $TS \leftarrow \{(input^l, target^l), l = 1, ..., \#D\}$ where
$input^l = s^l, a^l,$
$target^l = r^l + \gamma\ \max_{a'} \hat{Q}_k(s'^l, a')$
$\hat{Q}_{k+1} \leftarrow$ use regression on TS
$k \leftarrow k + 1$
while $(k < N)$

return Q_N

FIGURE 2.1: The Fitted Q algorithm.

LSTDQ $(D,\ k,\ \phi,\ \gamma,\ \pi)$ // Learns $\widehat{Q}^\pi$ from samples

// D : Source of samples (s, a, r, s')
// k : Number of basis functions
// ϕ : Basis functions
// γ : Discount factor
// π : Policy whose value function is sought

$\widetilde{\mathbf{A}} \leftarrow \mathbf{0}$ // $(k \times k)$ matrix
$\widetilde{b} \leftarrow \mathbf{0}$ // $(k \times 1)$ vector

for each $(s, a, r, s') \in D$
$\widetilde{\mathbf{A}} \leftarrow \widetilde{\mathbf{A}} + \phi(s, a)\Big(\phi(s, a) - \gamma\phi\big(s', \pi(s')\big)\Big)^\intercal$
$\widetilde{b} \leftarrow \widetilde{b} + \phi(s, a)r$

$\widetilde{w}^\pi \leftarrow \widetilde{\mathbf{A}}^{-1}\widetilde{b}$

return $\widetilde{w}^\pi$

FIGURE 2.2: The LSTDQ algorithm.

where w are k weights of linearly independent basis functions. Figure 2.2 summarizes the LSTDQ algorithm and figure 2.3 summarizes the LSPI algorithm.

2.8 Adaptive Delta modulation

Delta modulation is a special case of differential pulse-code modulation (DPCM) [7]. Very often a data stream contains samples that are highly correlated. A voice/sound

LSPI $(D,\ k,\ \phi,\ \gamma,\ \epsilon,\ \pi_0)$ // Learns a policy from samples

// D : Source of samples (s, a, r, s')
// k : Number of basis functions
// ϕ : Basis functions
// γ : Discount factor
// ϵ : Stopping criterion
// π_0 : Initial policy, given as w_0 (default: $w_0 = 0$)

$\pi' \leftarrow \pi_0$ // $w' \leftarrow w_0$

repeat
  $\pi \leftarrow \pi'$ // $w \leftarrow w'$
  $\pi' \leftarrow$ **LSTD**Q $(D,\ k,\ \phi,\ \gamma,\ \pi)$ // $w' \leftarrow$ **LSTD**Q $(D,\ k,\ \phi,\ \gamma,\ w)$
until $(\pi \approx \pi')$ // **until** $(\|w - w'\| < \epsilon)$

return π // **return** w

FIGURE 2.3: The LSPI algorithm.

signal is a common example of such a stream. Representing and transmitting each sample separately in such cases can be very costly. A more efficient method, is to code the difference of each sample from the previous one. Using a high enough sample rate, (much higher than the Nyquist rate) can ensure that successive samples differ little enough to be coded with few bits. In the limit, this leads to Delta modulation, where the difference between successive samples is encoded into 1-bit. The quantizer has two levels, $\pm\Delta$, therefore each sample differs $\pm\Delta$ from its previous (and next) one. An advantage of Delta modulation is the simplicity of the system. At the receiver the stream can be recovered using the following formula:

$$\hat{X}_n - \hat{X}_{n-1} = \hat{Y}_n$$

Where $\hat{X}_n$ is the (estimated) original data sample at time step n, and $\hat{Y}_n$ is the sample received at time step n.

For zero initial conditions $\hat{X}_n$ is:

$$\hat{X}_n = \sum_{0}^{n} \hat{Y}_i$$

The last formula essentially means that in order to find $\hat{X}_n$ we just need the (signed) sum of all previous $\hat{Y}_i$.

The magnitude of Delta is a very important design factor. Large values for Delta allow the system to follow fast changes of input, but cause too much quantization noise for slow varying signals. This is known as granular noise. The opposite is known as slope-overload distortion. It is the situation when Delta is too small to follow a rapid changing input. A thing to note, is that both of these problems can occur for the same Delta at different points on a stream.

A very simple, yet effective, solution to the above problems, is to change the magnitude of Delta adaptively, depending on the input. One -and possibly the simplest- rule for adjusting delta is:

$$\Delta_n = \Delta_{n-1} K^{e_n \times e_{n-1}}$$

where e_n is the output of the quantizer and K is a constant greater than one.

As a last note, it is very easy to see that these simple forms of modulation are very sensitive to noise. A single reversed bit does not only affect the current sample but all the successive ones as well. With just a few reversed bits, the rest of the received stream can be rendered unrecognizable. For this reason more sophisticated forms of modulation are usually employed over noisy channels. As we will see, these restrictions do not apply in our case.

Chapter 3

Problem statement

3.1 The need for continuous actions in control

Reinforcement Learning methods have met great success and significant development in the past years. Methods have been developed to deal with continuous states and as a result, more practical problems can be negotiated. A significant amount of effort has been put into applying these methods to control problems. Although the results have been satisfactory for lab tests, they have a number of disadvantages, that prevents wide spread commercial use. This is in a large part due to the fact that many real world control problems have a characteristic that has yet to be adequately addressed; continuous actions.

The approach adopted in most experiments is a coarse discretization of the action space. For example, in the inverted pendulum domain three actions are usually allowed: left force LF (-50 Newtons), right force RF ($+50$ Newtons), or no force NF (0 Newtons), whereas ideally we would like to be able to choose any action in [-50,50]. Such schemes do succeed at balancing the pendulum (or some other control task), but do so at a very steep price.

The first point to note, is that most experiments are computer simulations, not actual implementations. This makes it very easy to overlook the fact, that a physical system cannot instantaneously go from an output of -50 Newtons to an output of $+50$ Newtons. This poses a question as to whether all these -successfull in simulation- controllers would be as successfull in the real world.

The second (and very important) problem associated with the discrete approach, is that the resulting controllers are very abrupt on their actions, resulting in choppy control. If they succeed in keeping the plant within acceptable limits, their performance is deemed

adequate. In a commercial application, more often than not, these acceptable limits are not adequate at all. For example, in a servo motor control application, staying within an acceptable margin from the desired position is not enough on its own. Ideally one has to accomplish that, while at the same time making sure that:

- energy consumption stays as low as possible; especially so in mobile platforms, such as autonomous robots.
- mechanical stresses are minimized; this helps improve reliability as repeated mechanical stresses will wear out the system.
- noise induced to the power lines is kept within acceptable limits; in battery controlled applications and platforms where sophisticated noise filters are impractical, a single badly behaved device can create problems to other parts of the system (like CMOS logic).
- motion is smooth; coordinated movement (as in the links of a robotic arm) needs to be smooth. For example, choppy motion could destabilize a walking biped. Abrupt movement of a camera with pan and tilt capability may produce blurry images.

Apart from the fact that selecting among a very small number of actions limits the abilities of discrete controllers, there is a subtletry inherent to policy methods and discrete decisions in general, that significantly complicates things. One would hope that smooth changes in state should incur for smooth changes in action. Yet with discrete actions that is not the case. For states close to the decision surface, miniscule changes in state can result in very different actions.

3.2 Related work

A number of recent attempts [8–16] have been made to find ways to extend Reinforcement Learning, to domains with continuous actions. Although they represent steps to the right direction, each suffers from one or more disadvantages.

Most approaches try to tackle the problem, using methods reliant on specialized cases of neural networks. While for some applications this may not be a problem, there are cases where neural networks are not easily applicable. The first concern, are the computational resources required. Many classical -low cost- control applications run either on simple analog circuits or on humble 8-bit microcontrollers. To be competitive at a commercial level, successfull Reinforcement Learning methods would have to be

able to run on comparable hardware. The exponent needed to evaluate each neuron, makes them too expensive for such cases. Another issue is the unlearning effect often seen when the inputs change slowly. Since a controller is expected to spend most of its time close to the goal state, where the inputs do change slowly, this becomes an important problem.

A number of implementations, attempt to find an optimal set of discrete actions, or a set of actions that can slowly "move" when the state changes slowly. One common problem with these approaches, is their reliance on specific-for-the-task learning methods. These methods, most of the time suffer from inefficient use of samples and/or need for online, on-policy learning. This fact makes them hard to use on a real system, where samples are expensive. Another concern with these approaches, is that more often than not, they are very computationally demanding, even more so than the neural network based implementations. They also scale badly when the desired resolution or number of controlled variables increases.

High computational complexity during training could be tolerated, if samples were collected locally and some form of batch training took place at a proper system. With online, on-policy training methods, this is not an option. Computational complexity during control is even worse and forfeits the use of very low cost hardware.

A factor that should not be overlooked is the conceptual complexity of each approach. An approach that is too difficult to implement, deters researchers from trying it and chances are, will not find widespread use.

The following list summarizes, a number of existing approaches and their weaknesses. Of course this list is meant only to provide some perspective and is not exhaustive.

- Gaskett et al. [8] describe an online neural network based approach, coupled with a novel interpolator. Although its computational requirements are not quite as high as some of the other approaches, they are still out of range of most low cost microcontrollers. The approach relies exclusively on neural networks. One of the side effects is the unlearning that occurs when the inputs change slowly. The authors propose storing examples of state, action and next state transitions and replaying them to combat this. Although this works, it further increases the computational and memory requirements.

- Sallans and Hinton [15] present a stochastic method for approximating the value function and selecting good actions for Markov decision processes with large state and action spaces. The method approximates state-action values as negative free energies in an undirected graphical model called a product of experts. Actions can

be found even in large factored action spaces by the use of Markov Chain Monte Carlo sampling (MCMC). The large computational requirements, needed to get a good approximation, make this method unsuitable for most practical applications. Also the conceptual complexity of this approach, makes it difficult to adapt to existing projects.

- Strösslin and Gerstner [12] propose a biologically plausible, neural network based model, for spatial learning and navigation based on reinforcement learning. Like most, it relies on a particular method tied to neural networks and suffers from the unlearning problem. It is also unclear how it can be extended to domains that do not exhibit the same structure.

- Gross et al. [9] present a neural field approach to distributed Q-learning. Their results seem quite promising, for the task they tested it on. Of course since this is a specialized neural network based approach, it comes with all the negative side effects associated with such approaches. Its applicability on fast changing, inherently unstable domains is also questionable.

- Lazaric et al. [16] propose an on-policy, actor-critic approach, in which the stochastic policy of the actor is estimated through sequential Monte Carlo methods. This method attempts to find an optimal discrete subset of the continuous action space, by recognizing which areas of the action space are the most promising and exploring them. In order to achieve competitive performance, a relatively large number of discrete actions is required. This creates problems as the number of controlled variables increases.

- Touzet [11] proposes two neural implementations: one with a competitive multilayer perceptron and the other with a self-organizing map. Each action variable is coded into one neuron output, which varies from -1 to $+1$. For their experiments the output is partitioned into 10 areas, corresponding from speed 0 to speed 9. While this seems to work well for a few values, it is questionable whether it could differentiate between a larger number of possible values for each action, such as 2^8 or more.

- Esogbue and Hearnes [14] present an approximate dynamic programming approach to reinforcement learning, for continuous action set-point regulator problems. This is another approach trying to find an optimal discrete subset of the continuous action space. Its sample efficiency is rather limited and it does not scale well, as the number of controlled variables increases.

- Santamaría et al. [10] describe a method to select actions, using sparse-coded function approximators and a one step search. This works relatively well for a coarse

discretization, but the search step can become very expensive for fine discretizations and more than one controlled variable.

Chapter 4

The proposed approach

4.1 Taking advantage of temporal locality on actions

Usually when learning a policy on a small number of possible actions, the decision is the action that will be performed. However, when dealing with action spaces that can take a range of values, successive decisions exhibit a great deal of temporal locality. So, instead of making a decision about what particular action to perform, one can make a decision on how to modify the current action. In its simplest form this could be a decision whether to increase or decrease the output by a certain amount.

Suppose that we decide to partition an action space to 2^8 (256) uniformly-distributed discrete values. Now, the problem is to decide the size of the step (Δ) by which the current value will be increased or decreased at every time step. If we choose a large value like 64, we are essentially limiting the possible discrete values to 256/64=4. This is clearly a problem. We would have to choose the smallest possible step (1) in order to take advantage of the full range of possible values. If we do that though, our resulting controller will be unable to respond to fast changes. It would take 256 time steps to move from one end of the spectrum to the other. Figures 4.1 and 4.2 illustrate that point for the small and large Deltas respectively.

There is a very simple way to get the best of both worlds that has long been used in telecommunication systems: change the size of the step adaptively. All we need is a way to decide how to change the size of the step. A very simple, yet effective, rule is the sign of the decision. If for two successive time steps, the sign remains the same (with increase of the output represented as positive and decrease as negative) the step-size Δ is increased by a factor K, to $K \times \Delta$, while when the sign of two successive time steps is different, the size of the step is decreased by a factor K, to $\frac{\Delta}{K}$, where K is a real number > 1. Figure 4.3 illustrates that point.

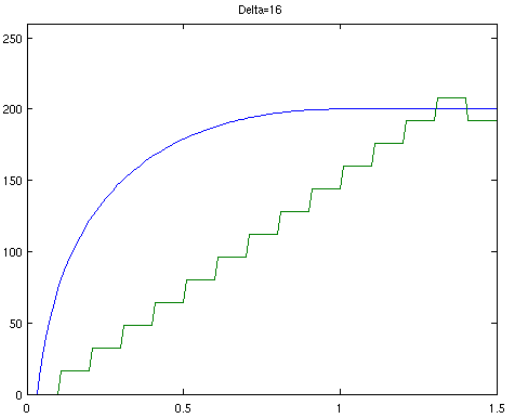

FIGURE 4.1: Desired action output and discrete approximation with constant $\Delta = 16$

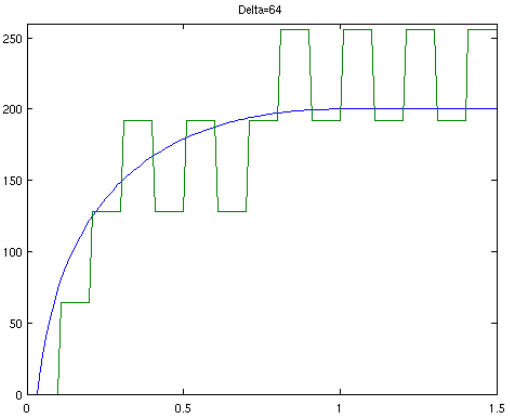

FIGURE 4.2: Desired action output and discrete approximation with constant $\Delta = 64$

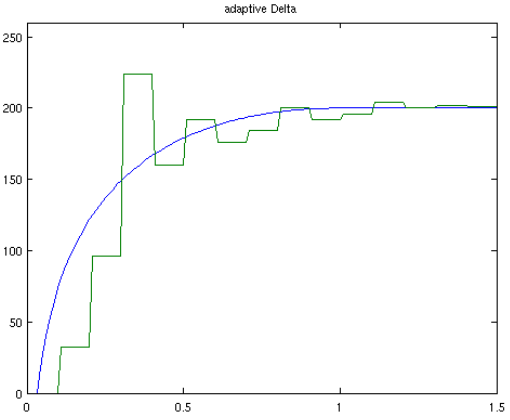

FIGURE 4.3: Desired action output and discrete approximation with adaptive Δ

controller $(\pi, s, e_{n-1}, \Delta_{n-1}, \Delta_{min}, \Delta_{max}, A_{n-1}, A_{res}, A_{min}, A_{max})$ // Uses a discrete policy to control a continuous plant

// π : A policy on states s with decisions +1, -1
// s : The current state
// e_{n-1} : The last decision
// Δ_{n-1} : The last delta
// Δ_{min} : Minimum allowed delta (usually 1)
// Δ_{max} : Maximum allowed delta (usually $\Delta_{max} = A_{res}$)
// A_{n-1} : The last continuous action
// A_{res} : The desired resolution of the continuous action
// A_{min} : Minimum allowed A
// A_{max} : Maximum allowed A

$e_n \leftarrow \pi(s)$ // make a decision based on current state
$\Delta_n \leftarrow \Delta_{n-1} K^{e_n \times e_{n-1}}$ // update Δ based on the decision

$if(\Delta_n > \Delta_{max})$ // make sure Δ stays within acceptable limits
    $\Delta_n \leftarrow \Delta_{max}$
$else\ if(\Delta_n < \Delta_{min})$
    $\Delta_n \leftarrow \Delta_{min}$

// update A according to the new Δ_n and the decision e_n
$A_n \leftarrow A_{n-1} + (e_n \times \Delta_n \times (A_{max} - A_{min}) \div A_{res})$

$if(A_n > A_{max})$ // make sure A stays within acceptable limits
    $A_n \leftarrow A_{max}$
$else\ if(A_n < A_{min})$
    $A_n \leftarrow A_{min}$

return A_n, Δ_n, e_n

FIGURE 4.4: The continuous controller algorithm.

4.2 Algorithm

With the previous observations in mind, it is now easy to implement an algorithm that takes advantage of temporal locality on actions. The resulting algorithm uses a discrete, binary policy that chooses whether A, the continuous action, will be increased or decreased at each step. The step-size Δ is adjusted appropriately, according to a scaling factor K, and A is updated. The whole process is repeated at each time step. Figure 4.4 summarizes the proposed algorithm.

The steps of the algorithm are explained below.

1. use the policy to decide whether A will be increased or decreased
2. update Δ according to the decisions in the last two steps. If the decisions are the same (for the two time steps), scale up Δ by a factor K to keep up with rapid

changes. If the decisions are different, scale down Δ by a factor K to enable fine adjustments. K is a constant greater than one.

3. keep Δ within its predefined limits. (where usually Δ_{min}=1 and $\Delta_{max} = A_{res}$)

4. update the continuous action, based on its previous value, the current decision and the current Δ.

5. keep the continuous action A within its predefined limits.

6. return the resulting continuous action A along with the updated Δ and e, that will be required on the next step. These last two variables could be declared static and stay "inside" the algorithm.

4.3 Efficiency

One of the main goals in the development of this algorithm was computational efficiency and ease of implementation on low cost microcontroller hardware. The total cost during execution for each basic operation is explained below.

1. $e_n \leftarrow \pi(s)$: The cost of this operation is the time needed to evaluate the policy. It is directly dependent on the policy representation used.

2. $\Delta_n \leftarrow \Delta_{n-1} K^{e_n \times e_{n-1}}$: For $K = 2$ this operation could be implemented as follows:

   $if((e_n + e_{n-1}) == 0)$
   $\quad \Delta_n \leftarrow \Delta_{n-1} >> 1$
   $else$
   $\quad \Delta_n \leftarrow \Delta_{n-1} << 1$
   (where $>>$ and $<<$ denote the shift right and shift left operations respectively)

   The operations performed are: one addition, one zero equality, one conditional and one shift operation (one for each execution path).

3. $if(\Delta_n > \Delta_{max})$
   $\quad \Delta_n \leftarrow \Delta_{max}$
   $else\ if(\Delta_n < \Delta_{min})$
   $\quad \Delta_n \leftarrow \Delta_{min}$

   One to two comparisons, one to two jumps and zero to one assignments are required.

4. $A_n \leftarrow A_{n-1} + (e_n \times \Delta_n \times ((A_{max} - A_{min}) \div A_{res}))$
   Note that $((A_{max} - A_{min}) \div A_{res}))$ is a constant and can be precomputed and replaced. This leaves us with two multiplications. If multiplication is expensive, the multiplication with e_n can be replaced with a conditional. Also, if K is a power of 2, Δ will be a power of 2 too. Thus, step (2) can be replaced with:

   $if((e_n + e_{n-1}) == 0)$
   $\quad \Delta_n \leftarrow \Delta_{n-1} - 1$
   $else$
   $\quad \Delta_n \leftarrow \Delta_{n-1} + 1$

   And this step becomes:

   $if(e_n == 1)$
   $\quad A_n \leftarrow A_{n-1} + (((A_{max} - A_{min}) \div A_{res})) >> \Delta_n)$
   $else$
   $\quad A_n \leftarrow A_{n-1} - (((A_{max} - A_{min}) \div A_{res})) << \Delta_n)$

   The cost of this step is then: one conditional, one equality test, one addition (or one subtraction) and a shift operation.

5. $if(A_n > A_{max})$
   $\quad A_n \leftarrow A_{max}$
   $else\ if(A_n < A_{min})$
   $\quad A_n \leftarrow A_{min}$

   Once again, one to two comparisons, one to two jumps and zero to one assignments are required.

From the analysis above, we can conclude that the algorithm has very low computational requirements. Excluding the policy evaluation step, especially when K is a power of two, only a small number of additions, subtractions, shift operations and conditional statements are required. No divisions or multiplications are necessary. The total computation needed is less than would be required for the exponent of a single neuron. Also bare in mind that the policy has to choose one out of two possible actions. Usually this makes it a lot faster to evaluate, than the full discrete partitioning, or other continuous methods. To our knowledge, this is the least computationally intensive implementation of continuous actions for Reinforcement Learning.

4.4 Scalability

If we have to take simultaneously a decision for n continuous action variables we need a policy with 2^n discrete joint action choices. For example at the bicycle domain the 4 joint actions are ({-1,-1}, {-1,1}, {1,-1}, {1,1}). Admittedly this is a number that grows fast. Having said that, it is worth to note that it grows slower than the discrete case. If a discrete controller has m choices per action variable, it will need m^n joint actions in a domain with n controlled variables. Since usually $m > 2$ and 2 is the smallest base possible, it has the best scalability possible when taking decisions on all variables at each time step. For example, at the bicycle domain, if we have to control the torque applied to the handlebar, with three discrete actions (({-2}, {0}, {2})) and the displacement of the rider with another three discrete actins ({-0.02}, {0}, {0.02}), we need $3^2 = 9$ discrete actions, whereas our approach would require only $2^2 = 4$ discrete joint actions.

For special cases, if we only need to decide for one of the continuous actions at each time step, then the number of discrete actions required drops to $2n$, as opposed to 2^n, with an appropriate delay penalty proportional to n.

4.5 A note on continuity

"In theory, theory and practice are the same. In practice, they are not."

Lawrence Peter Berra

One could argue, that this technique does not truly offer continuous actions, the way some neural network based approaches do. Partitioning an action space to 2^8 or even 2^{16} actions, does provide a large number of actions, but is still discrete. While in principle that statement is true, one has to take into consideration how everything will be implemented on real hardware. ADCs and DACs have finite precision, which for the majority of cases on modern microcontrollers is at most 2^{16}. Also the speed of the control loop, will be limited by the time it takes to complete one iteration of the algorithm on real (limited) hardware. Therefore, from the authors perspective, it is best to have that in mind when designing and simulating a new algorithm, rather than striving for the best theoretical results that cannot be realized in practice.

4.6 Integration with Reinforcement Learning algorithms

Most existing Reinforcement Learning algorithms can be used with this technique. There are only two requirements: The algorithm of choice needs to support continuous state spaces, and be able to produce policies with two actions for each controlled variable. The original problem has a state space $\mathcal{S}$. Depending on whether all the state variables are observable -or hidden-, we are dealing with an MDP (Markov Decision Process) or a POMDP (Partially Observable Markov Decision Process). Using the proposed algorithm the state is now dependent on the original state variables, plus A, Δ and e_{n-1}. We can enhance the state description, so that the Reinforcement Learning algorithm can observe them, or we can choose to hide any number of them and treat the resulting system as a POMDP. As was observed during our experiments, the best results where produced when A was taken into account and Δ and e_{n-1} where ignored. Depending on the particular application, a different set of state variables may produce the best results.

4.7 Practical considerations

The fundamental principle of this technique is taking advantage of temporal locality of actions. In domains where the action has to change fast we may have to decrease the time step (increase the rate at which our controller makes decisions). This has two potentially negative consequences:

1. More samples are required to cover the same amount of real time. Depending on the situation, this may or may not be an issue.

2. The ill effects of the Temporal Credit Assignment Problem in Reinforcement Learning are amplified. This is a consequence, of the fact that actions and the rewards that result from them, become separated by more samples, as the time steps become shorter. This can be alleviated using more direct reward functions, or shaping rewards.

Chapter 5

Experimental results

5.1 Inverted Pendulum

The inverted pendulum problem [17] requires balancing a pendulum of unknown length and mass, at the upright position by applying forces to the cart it is attached to. Usually the state space (vertical angle θ and angular velocity $\dot{\theta}$) is continuous and just three actions are allowed: left force LF (-50 Newtons), right force RF ($+50$ Newtons), or no force NF (0 Newtons). The initial action space for our experiment consisted of 2^8 equally spaced actions in range [-50 Newtons, 50 Newtons]. As we will see later, both the action and state space where further augmented for our experiments. All actions are noisy; uniform noise in [-10, 10] is added to the chosen action. The transitions are governed by the nonlinear dynamics of the system (Wang et al., 1996) and depend on

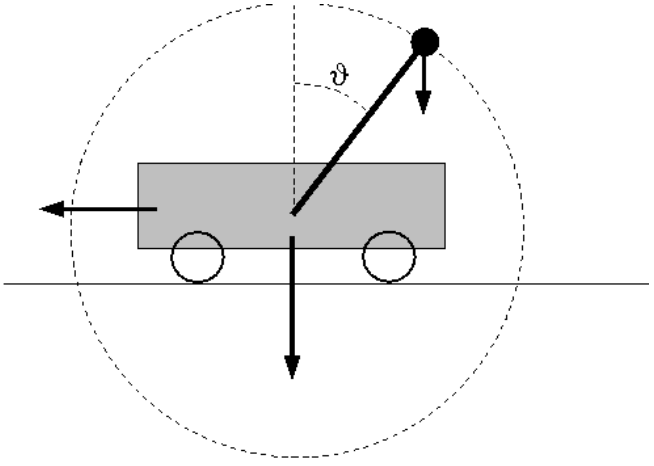

FIGURE 5.1: The Inverted Pendulum

the current state and the current (noisy) control u:

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml(\dot{\theta})^2 \sin(2\theta)/2 - \alpha\cos(\theta)u}{4l/3 - \alpha ml\cos^2(\theta)} \ ,$$

where g is the gravity constant ($g = 9.8m/s^2$), m is the mass of the pendulum ($m = 2.0$ kg), M is the mass of the cart ($M = 8.0$ kg), l is the length of the pendulum ($l = 0.5$ m), and $\alpha = 1/(m+M)$. The simulation step is set to 0.02 seconds. Thus, the control input is given at a rate of 50 Hz, at the beginning of each time step, and is kept constant during any time step. A reward of $-|(2\theta/\pi)|$ is given as long as the angle of the pendulum does not exceed $\pi/2$ in absolute value (the pendulum is above the horizontal line). An angle greater than $\pi/2$ signals the end of the episode and a reward (penalty) of -1. The discount factor of the process is set to 0.95. The value of K for all the experiments is set to 2. Note that for many experiments a K close to 1.5 produced slightly better results, but since choosing a value for K that is not a power of 2 makes calculations more expensive, the choice of $K = 2$ was preferred.

5.1.1 First attempt

For this first experiment, the original 2-dimensional state space $\mathcal{S}$ where $s = (\theta, \dot{\theta})$ was used. We applied LSPI with a set of 10 basis functions for each of the 2 actions, thus a total of 20 basis functions, to approximate the value function. These 10 basis functions included a constant term and 9 radial basis functions (Gaussians) arranged in a 3×3 grid over the 2-dimensional state space. In particular, for some state $s = (\theta, \dot{\theta})$ and some action a, all basis functions are zero, except the corresponding active block for action a which is

$$\left(1,\ e^{-\frac{\|s-\mu_1\|^2}{2\sigma^2}},\ e^{-\frac{\|s-\mu_2\|^2}{2\sigma^2}},\ e^{-\frac{\|s-\mu_3\|^2}{2\sigma^2}},\ \ldots,\ e^{-\frac{\|s-\mu_9\|^2}{2\sigma^2}} \right)^\intercal \ ,$$

where the μ_i's are the 9 points of the grid $\{-\pi/4,\ 0,\ +\pi/4\} \times \{-1,\ 0,\ +1\}$ and $\sigma^2 = 1$.

Figures 5.2 through 5.4 [1] show the results for a single run of the learned policy for 1000 training episodes. The angle of the pendulum which is the most important, stays at very low values for the duration of the experiment. Looking at the histogram for applied force and the use of deltas, we can see that the controller does not make very good use of its potential. The applied force seems to be concentrated at a small number of values and only the top range of deltas has been used (that results in large steps in the action space). The mean force magnitude (sum of force magnitude for each time step divided

[1] For an explanation on how to read the diagrams see Appendix A.

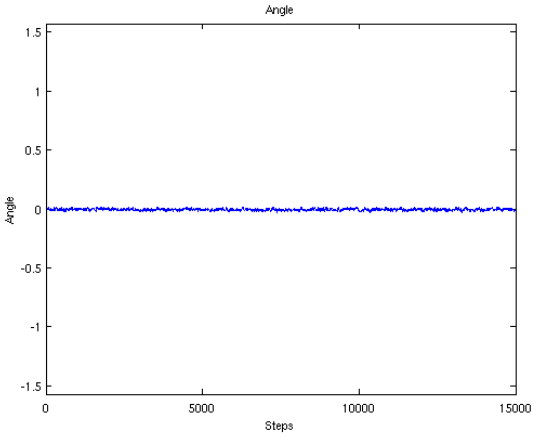

FIGURE 5.2: Inverted pendulum (LSPI): Angle over time.(50N)

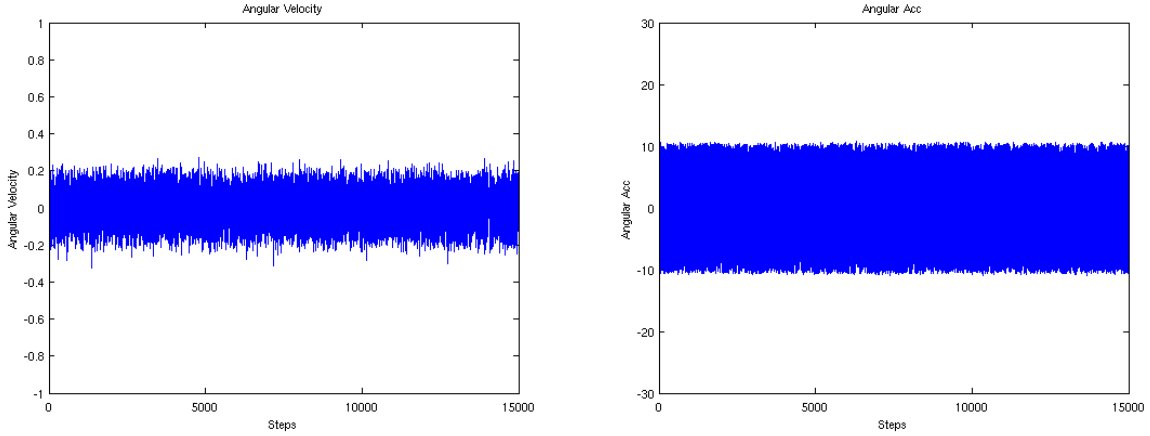

FIGURE 5.3: Inverted pendulum (LSPI)(50N):
Angular velocity and Acceleration over time.

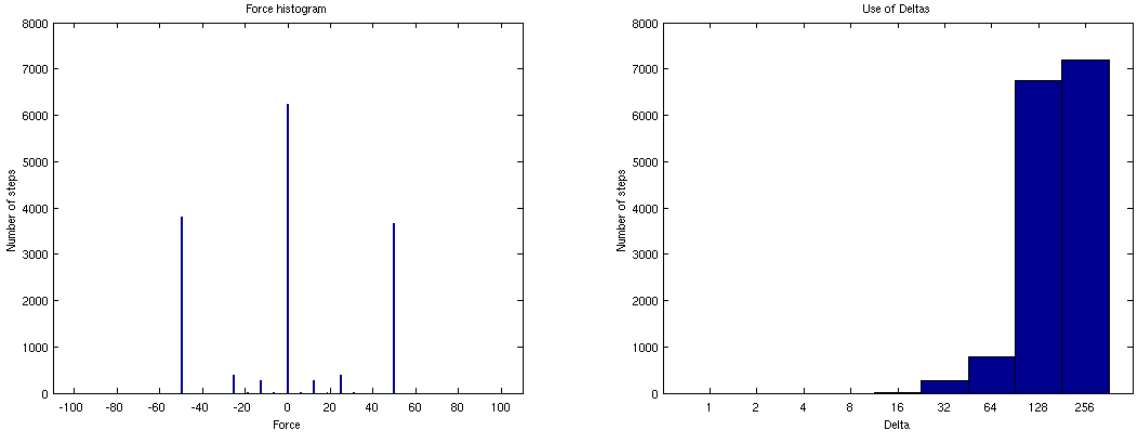

FIGURE 5.4: Inverted pendulum (LSPI)(50N):
Force histogram and use of Deltas. Mean force magnitude: 26.5996

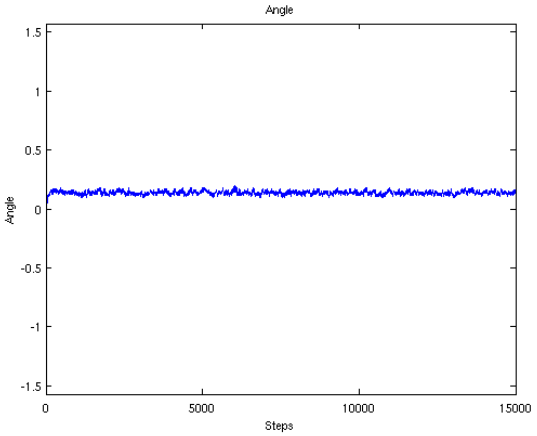

FIGURE 5.5: Inverted pendulum (LSPI): Angle over time.(100N)

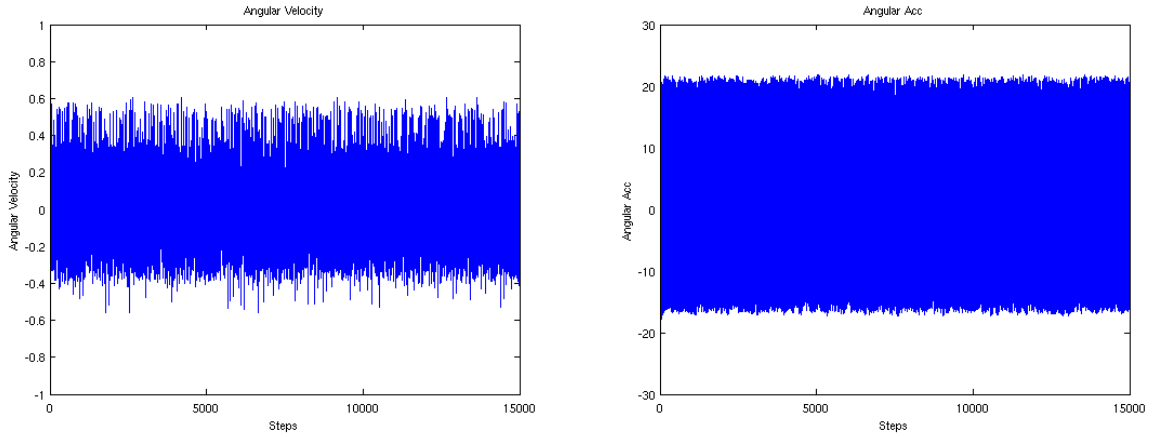

FIGURE 5.6: Inverted pendulum (LSPI)(100N):
Angular velocity and Acceleration over time.

by the number of steps) for the experiment is 26.5996 This is a rather large value and as we will see later, the controller can perform better. The significance of this metric is that in a real implementation it would have a direct correlation to power consumption and mechanical stresses. Obviously the smaller this value is, the better.

As figures 5.5 through 5.7 show, increasing the force range to [-100 Newtons, 100 Newtons] the system becomes more erratic, with the top end of the force range been used too often, as well as angular velocity and acceleration reaching unnecessary high values.

Increasing force range further, destabilized the system even more, up to the point where controlling it became impossible for any policy. Since from a mechanical perspective we are enhancing the system it seemed obvious that this was a result of poor ability of the control policy to capture the underlying principles of the system.

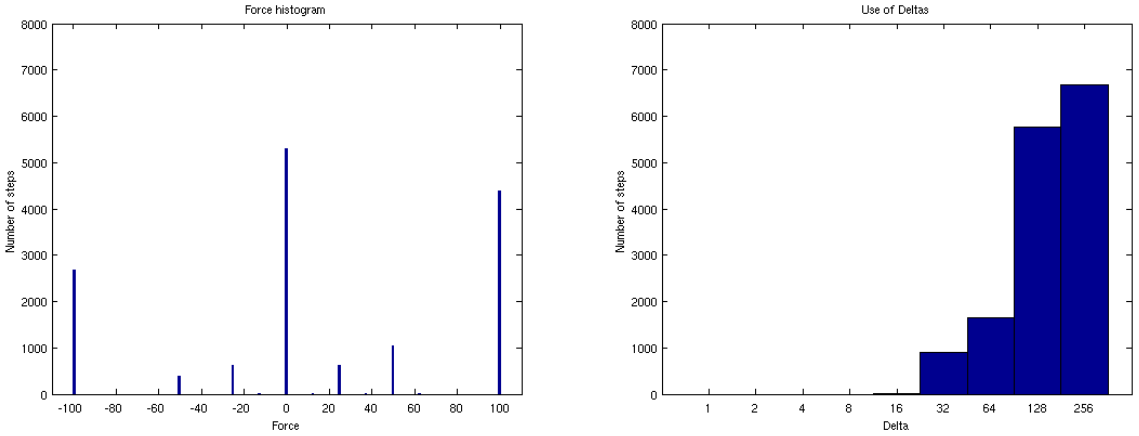

FIGURE 5.7: Inverted pendulum (LSPI)(100N):
Force histogram and use of Deltas. Mean force magnitude: 53.8042

It was the authors belief that if we take into account the applied force, the situation would be greatly improved.

5.1.2 A robust controller

The state is now s $= (\theta, \dot{\theta}, F)$ and for each action a there is a total of 28 basis functions including a constant term and 27 radial basis functions (Gaussians) arranged in a $3\times3\times3$ grid over the 3-dimensional normalized state space:

$$(1, e^{-\frac{(sx-\theta_1)^2/(\pi/4)^2+(sy-\dot{\theta}_1)^2/2.5^2+(sz-F_1)^2/50^2}{2\sigma^2}}, e^{-\frac{(sx-\theta_2)^2/(\pi/4)^2+(sy-\dot{\theta}_2)^2/2.5^2+(sz-F_2)^2/50^2}{2\sigma^2}},$$

$$\cdots, e^{-\frac{(sx-\theta_3)^2/(\pi/4)^2+(sy-\dot{\theta}_3)^2/2.5^2+(sz-F_3)^2/50^2}{2\sigma^2}})^\intercal$$

where θ_i's are $\{-\pi/4,\ 0,\ +\pi/4\}$, $\dot{\theta}_i$'s are $\{-2.5,\ 0,\ +2.5\}$ F_i's are $\{-50,\ 0,\ +50\}$ and $\sigma^2 = 1$.

Figures 5.8 through 5.10 show the results of the new policy for the [-50 Newtons, 50 Newtons] range. Both the angular velocity and acceleration stay at much lower values than before. Small values for deltas are preferred, making good use of the action space resolution and the applied force is concentrated around zero. The mean force magnitude for the experiment is 2.6562. This is quite good and a tenfold improvement over 26.5996 of the previous controller.

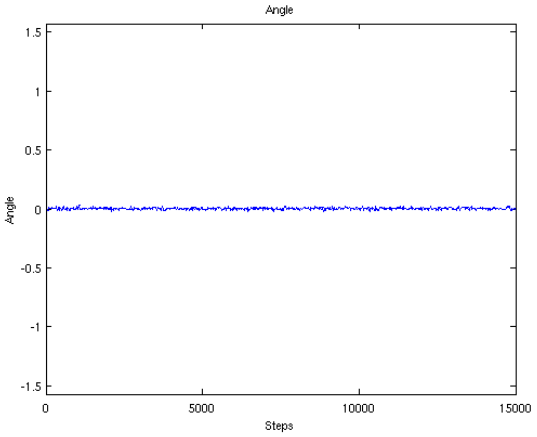

FIGURE 5.8: Inverted pendulum (LSPI) s = $(\theta, \dot{\theta}, F)$: Angle over time.(50N)

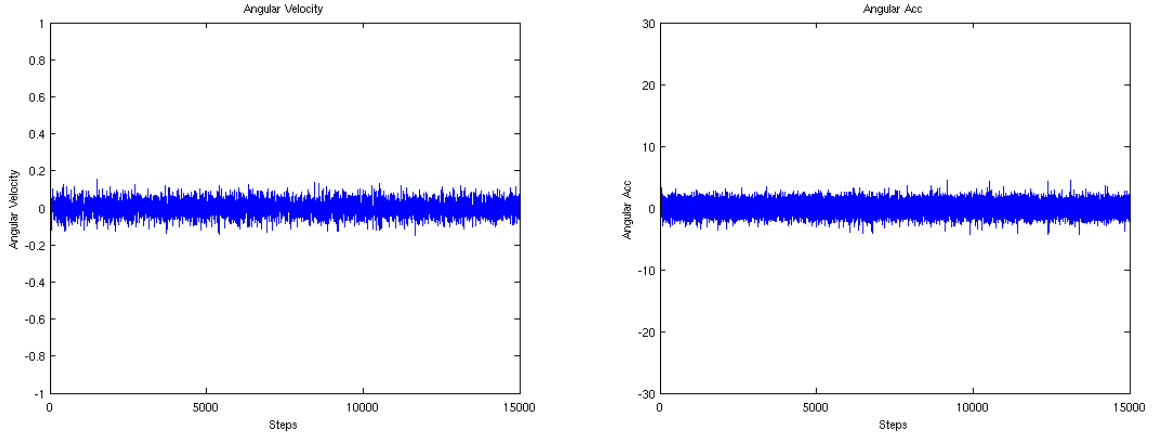

FIGURE 5.9: Inverted pendulum (LSPI)(50N) s = $(\theta, \dot{\theta}, F)$:
Angular velocity and Acceleration over time.

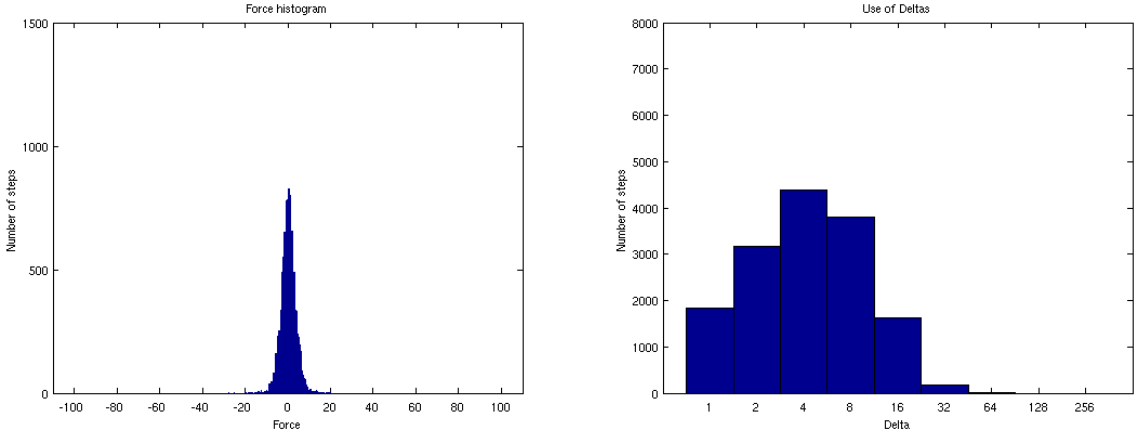

FIGURE 5.10: Inverted pendulum (LSPI)(50N) s = $(\theta, \dot{\theta}, F)$:
Force histogram and use of Deltas. Mean force magnitude: 2.6562

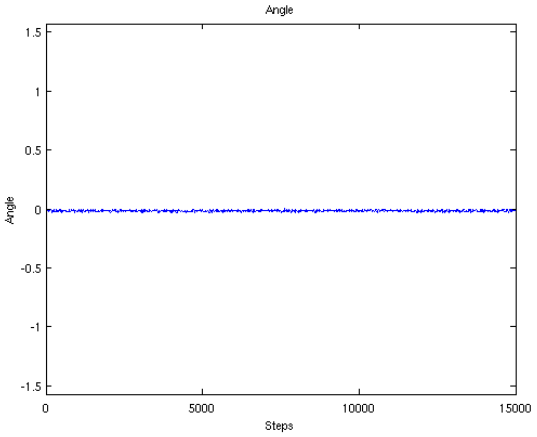

FIGURE 5.11: Inverted pendulum (LSPI) s = $(\theta, \dot{\theta}, F)$: Angle over time.(100N)

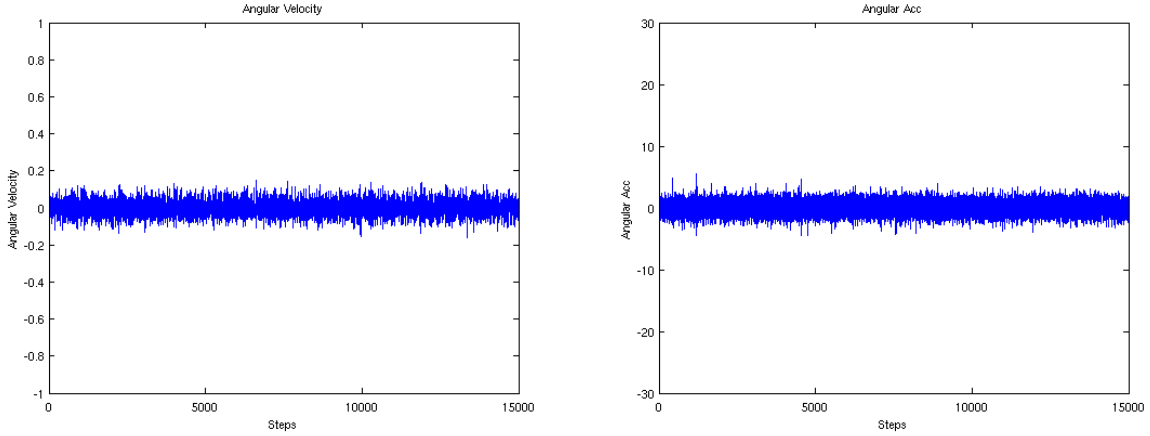

FIGURE 5.12: Inverted pendulum (LSPI)(100N) s = $(\theta, \dot{\theta}, F)$:
Angular velocity and Acceleration over time.

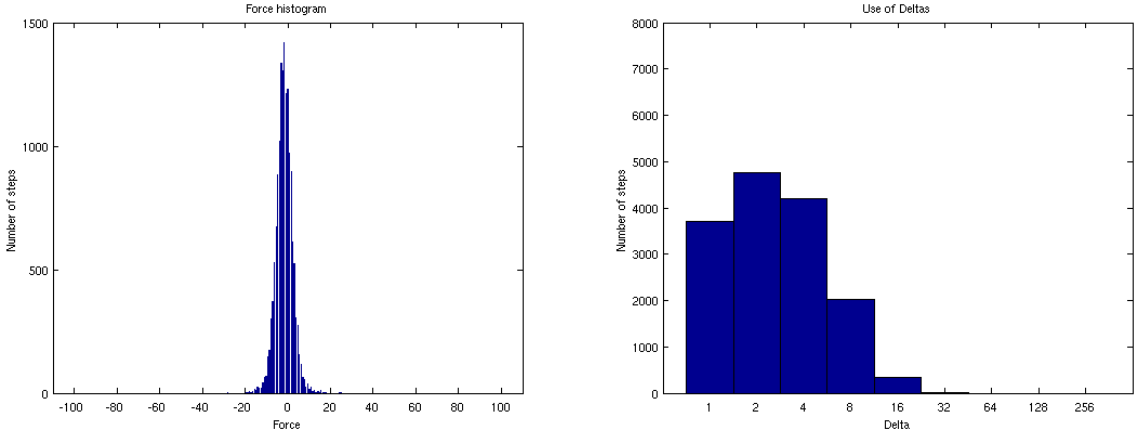

FIGURE 5.13: Inverted pendulum (LSPI)(100N) s = $(\theta, \dot{\theta}, F)$:
Force histogram and use of Deltas. Mean force magnitude: 3.2352

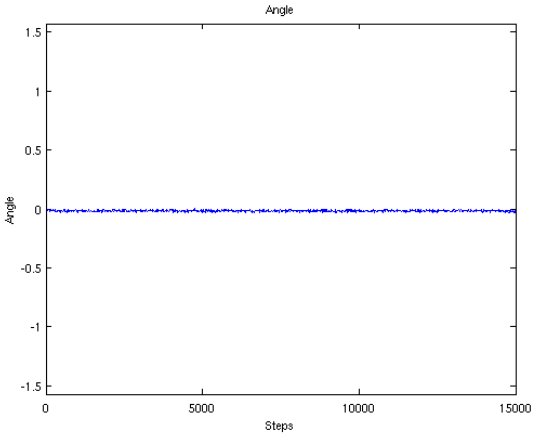

FIGURE 5.14: Inverted pendulum (LSPI): Angle over time.(25600N)

For the [-100 Newtons, 100 Newtons] experiment, the state space is normalized for the new force range, resulting in the following basis functions:

$$(1, e^{-\frac{(sx-a1)^2/(\pi/4)^2+(sy-b1)^2/2.5^2+(sz-c1)^2/100^2}{2\sigma^2}}, e^{-\frac{(sx-a2)^2/(\pi/4)^2+(sy-b2)^2/2.5^2+(sz-c2)^2/100^2}{2\sigma^2}},$$

$$\cdots, e^{-\frac{(sx-a3)^2/(\pi/4)^2+(sy-b3)^2/2.5^2+(sz-c3)^2/100^2}{2\sigma^2}})^\intercal$$

where a_i's are $\{-\pi/4,\ 0,\ +\pi/4\}$, b_i's are $\{-2.5,\ 0,\ +2.5\}$ c_i's are $\{-100,\ 0,\ +100\}$ and $\sigma^2 = 1$. Figures 5.11 through 5.13 show the results for the [-100 Newtons, 100 Newtons] range. No significant loss in performance can be seen. Deltas stay at even lower values, due to the fact that each step in this action space is larger. The mean force magnitude for the experiment is 3.2352, only marginally higher than 2.6562 of the 50 newtons case. Compared to 53.8042 when the applied force was not taken into account, it is a great improvement.

5.1.3 Increasing force range

Continuing to increase force range, results in increasingly more unstable controllers, up to the point where the system once again becomes impossible to control for all resulting policies; even if we increase the action space resolution. The main reason for this behavior is the degradation of sample sessions. If we train a controller using a moderate force range and then for testing we increase both the force range and the resolution by the same factor, the controller performs exactly as in the moderate force range case. Figures 5.14 through 5.16 show how a controller trained at $[-100\ Newtons, 100\ Newtons]$ with 2^8 resolution, performed at $[-25600\ Newtons, 25600\ Newtons]$ with 2^{16} resolution.

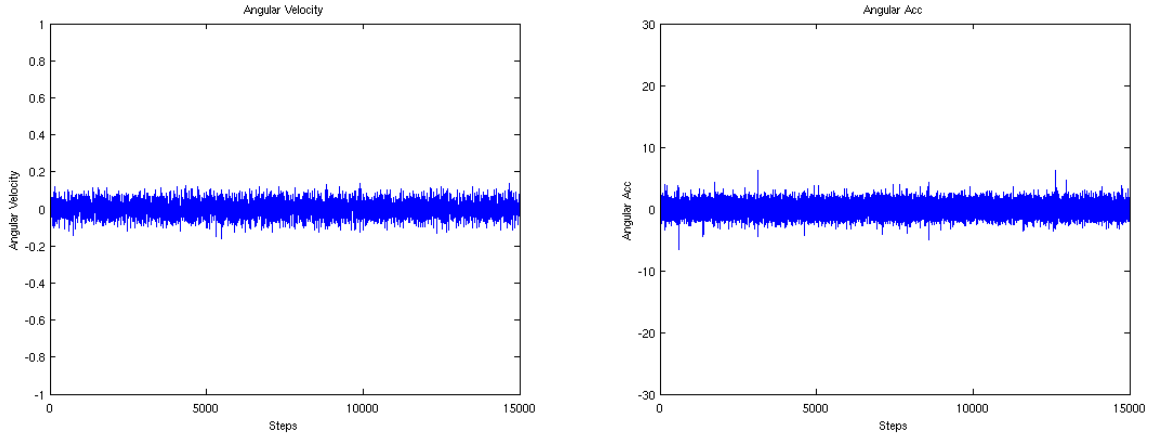

FIGURE 5.15: Inverted pendulum (LSPI)(25600N):
Angular velocity and Acceleration over time.

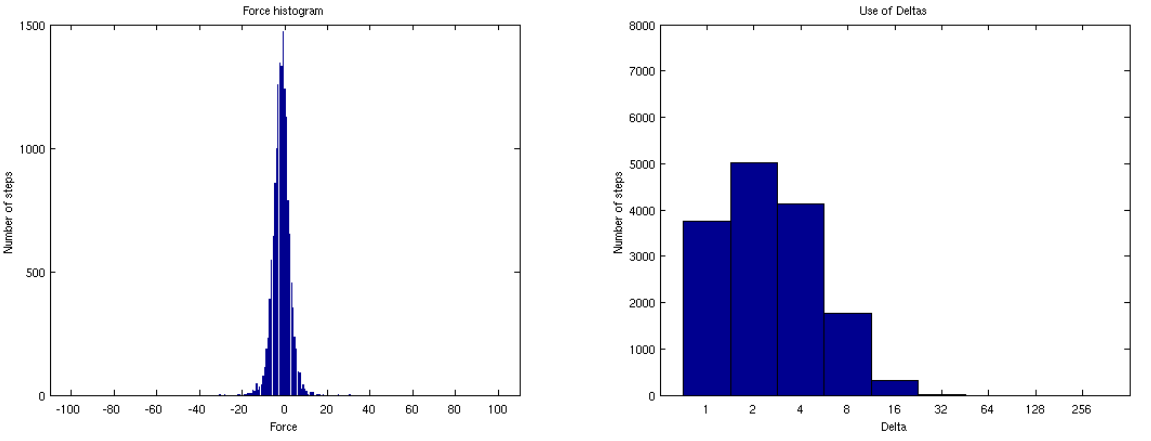

FIGURE 5.16: Inverted pendulum (LSPI)(25600N):
Force histogram and use of Deltas. Mean force magnitude: 3.1367

5.1.4 Performance under increasing noise

In all the above experiments, deltas as well as force magnitude, are concentrated at very low values. Although this is generally a good thing, a question arises whether the resulting controllers can use the rest of the range when the situation requires it. Figures 5.17 through 5.22 show the results for the [-100 Newtons, 100 Newtons] controller, when noise is increased to [-20, 20], [-50, 50] and [-100, 100] newtons (additive uniform noise).

We can deduce from these results, that the controller uses its available range of deltas and applied force, making a very good tradeoff between persision and controlability.

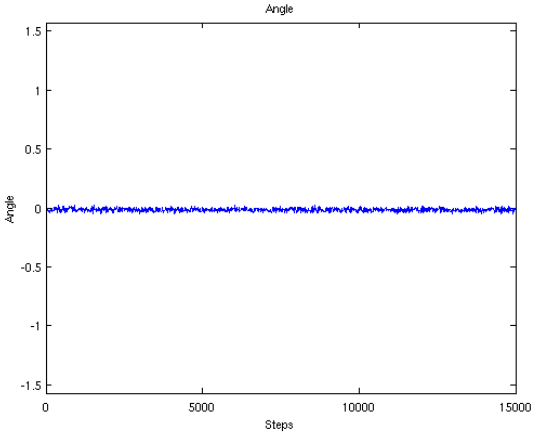

FIGURE 5.17: Inverted pendulum (LSPI) noise[-20, 20]: Angle over time.(100N)

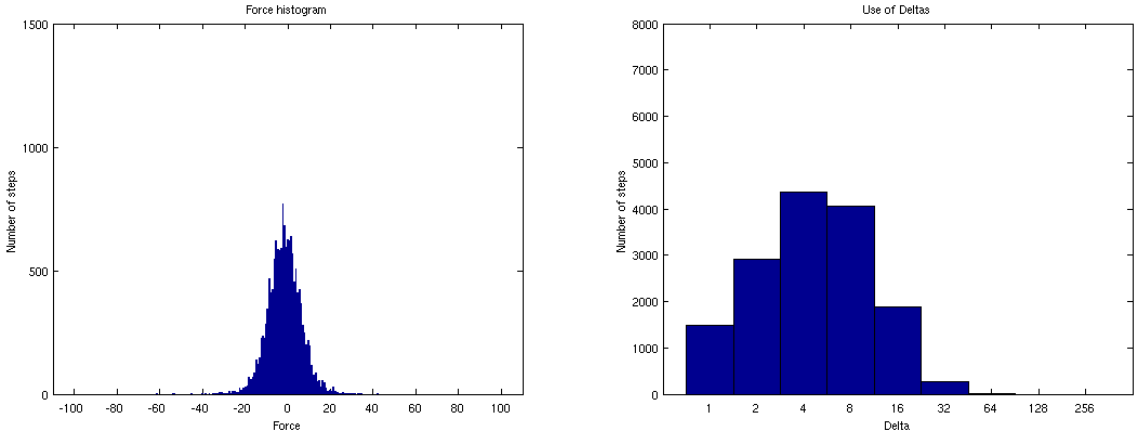

FIGURE 5.18: Inverted pendulum (LSPI)(100N) noise[-20, 20]:
Force histogram and use of Deltas. Mean force magnitude: 6.0767

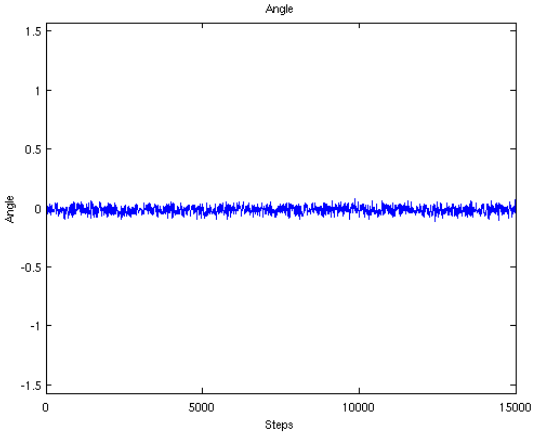

FIGURE 5.19: Inverted pendulum (LSPI) noise[-50, 50]: Angle over time.(100N)

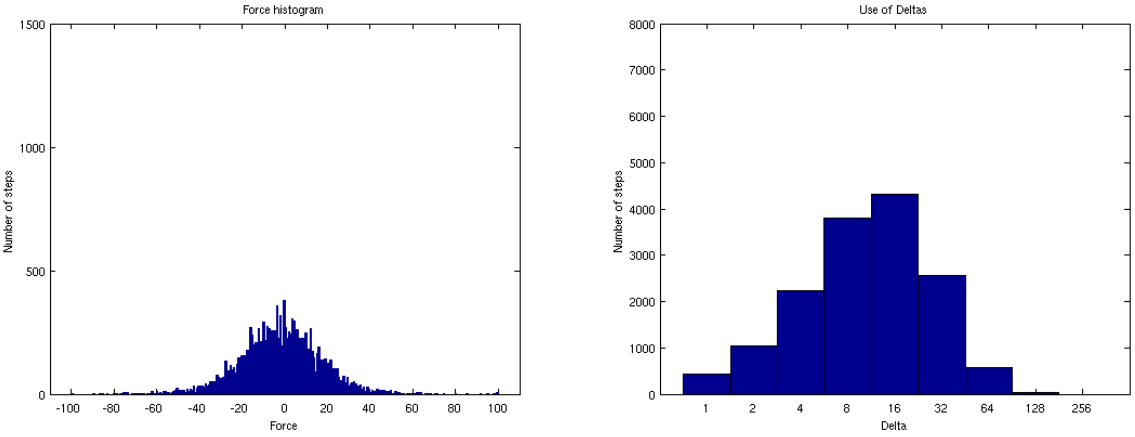

FIGURE 5.20: Inverted pendulum (LSPI)(100N) noise[-50, 50]:
Force histogram and use of Deltas. Mean force magnitude: 14.6652

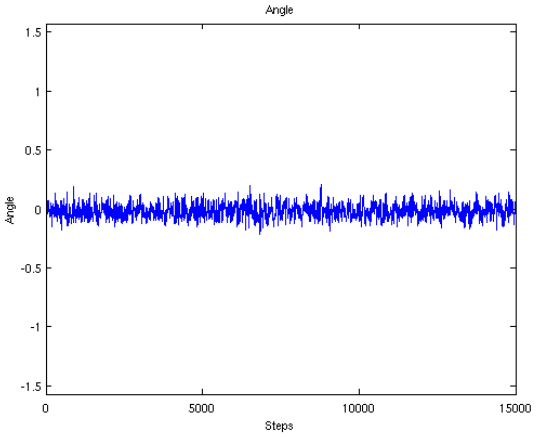

FIGURE 5.21: Inverted pendulum (LSPI) noise[-100, 100]: Angle over time.(100N)

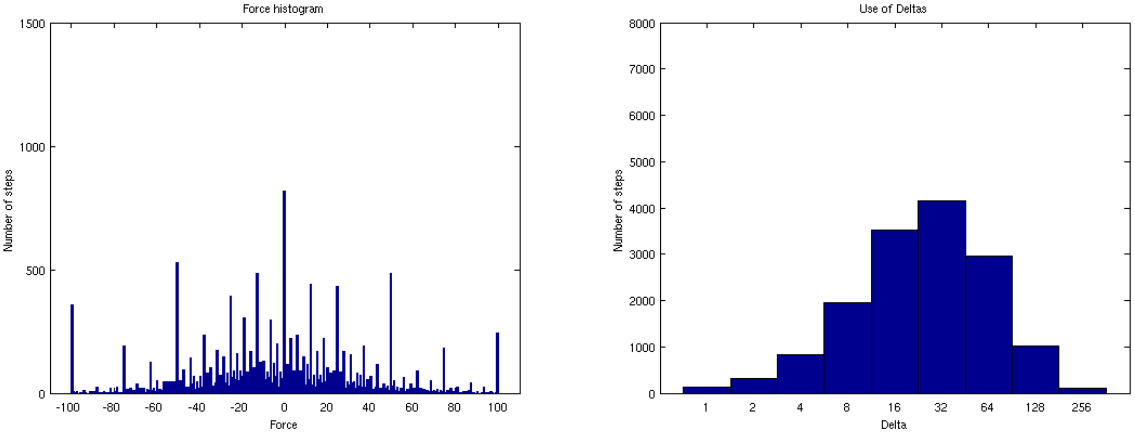

FIGURE 5.22: Inverted pendulum (LSPI)(100N) noise[-100, 100]:
Force histogram and use of Deltas. Mean force magnitude: 31.0365

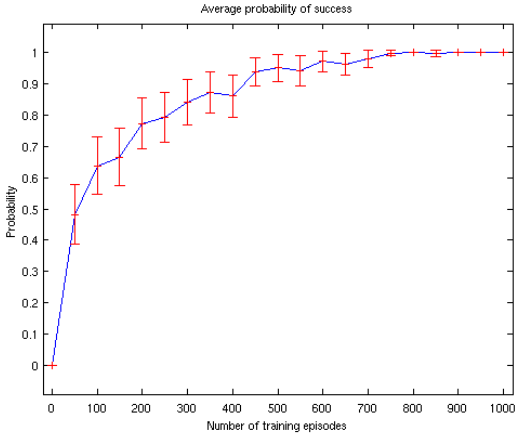

FIGURE 5.23: Inverted pendulum (LSPI): Average probability of success.(50N)

5.1.5 Statistical data

A number of systematic experiments were conducted to assess the effectiveness of the new learning scheme. Training samples were collected in advance from "random episodes", that is, starting in a randomly perturbed state very close to the equilibrium state $(0, 0, 0)$ and following a policy that selected actions uniformly at random. The average length of such episodes was about 36 steps for the 100 Newtons case and 42 steps for the 50 Newtons case, thus each one contributed about 36 or 42 samples to the set respectively. The same sample set was used throughout all iterations of each run of LSPI.

Figures 5.23 through 5.26 show the performance of the control policies learned by LSPI as a function of the number of training episodes. For each size of training episodes, the learned policy was evaluated 100 times to estimate accurately the average number of balancing steps (we do not show confidence intervals for this estimation). This experiment was repeated 100 times for the entire horizontal axis to obtain average results and the 95% confidence intervals over different sample sets. Each episode was allowed to run for a maximum of 15000 steps corresponding to 5 minutes of continuous balancing in real-time. A run that balanced for this period of time was considered to be successful.

We can see that for 800 training episodes, the success rate for both the 50 Newton and the 100 Newton controllers approaches 100%. This corresponds to approximately 33600 samples, or 11.2 minutes of real time for the 50 Newton controller and 28800 samples, or 9.6 minutes of real time for the 100 Newton controller. Although these results alone are not significantly better than the discrete case for 50 Newtons, we have to take into account the quality of solutions offered.

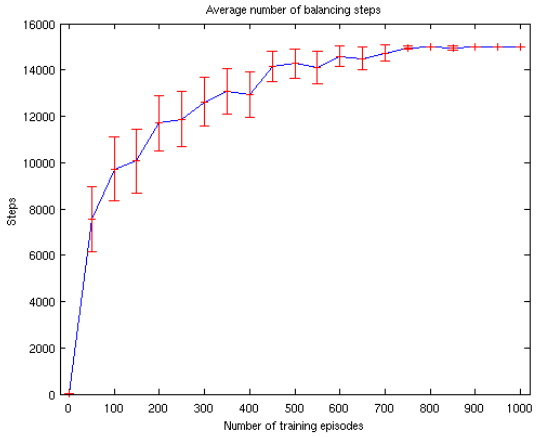

FIGURE 5.24: Inverted pendulum (LSPI): Average balancing steps.(50N)

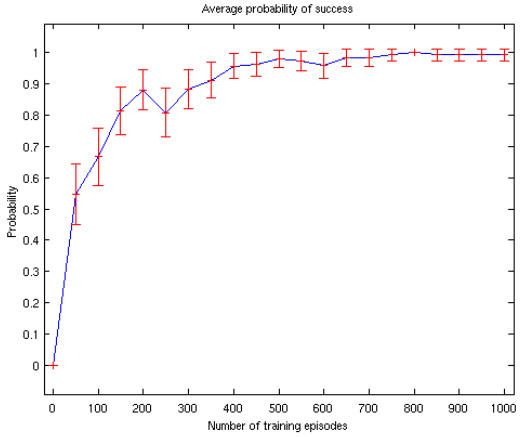

FIGURE 5.25: Inverted pendulum (LSPI): Average probability of success.(100N)

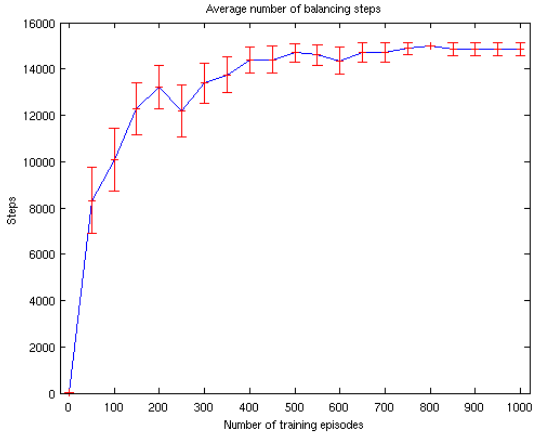

FIGURE 5.26: Inverted pendulum (LSPI): Average balancing steps.(100N)

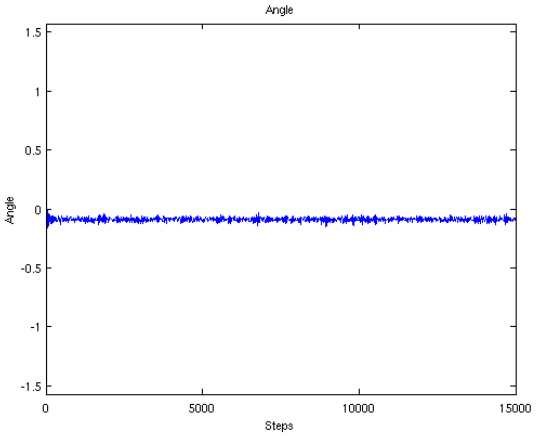

FIGURE 5.27: Inverted pendulum (Q-learning/ER): Angle over time.(100N)

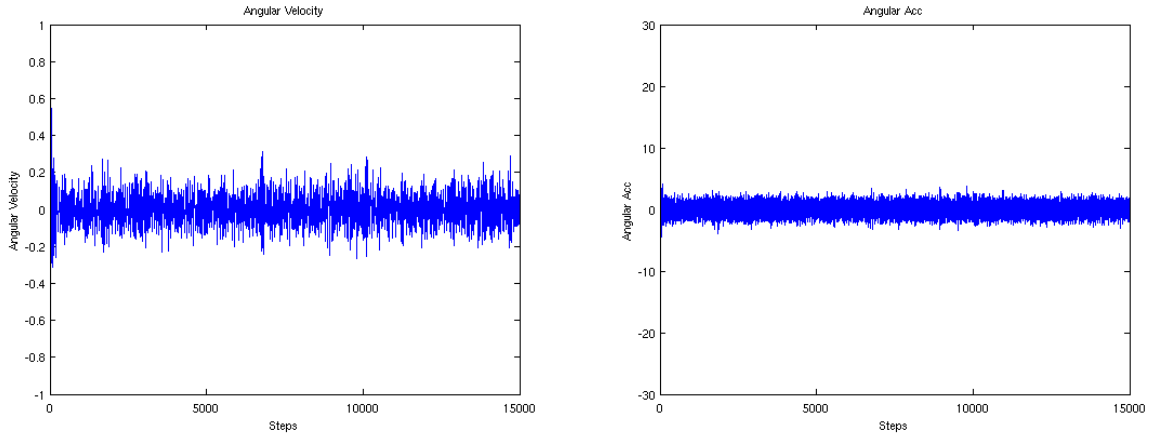

FIGURE 5.28: Inverted pendulum (Q-learning/ER)(100N): Angular velocity and Acceleration over time.

5.1.6 Q-learning/ER

The experiments above were repeated, using Q-learning with the experience replay technique for improved learning and our continuous-action selection algorithm for the $[-100\ Newtons,\ 100\ Newtons]$ range.

We can see from figures 5.27 through 5.31 that the results for single runs are comparable to the LSPI case, although the percentage of successfull policies is significantly less. One more thing to note with Q-learning is that very frequently it produced policies that balanced the pendulum at an angle slightly different than 0. Both LSPI and fitted Q iteration did not suffer from this behavior.

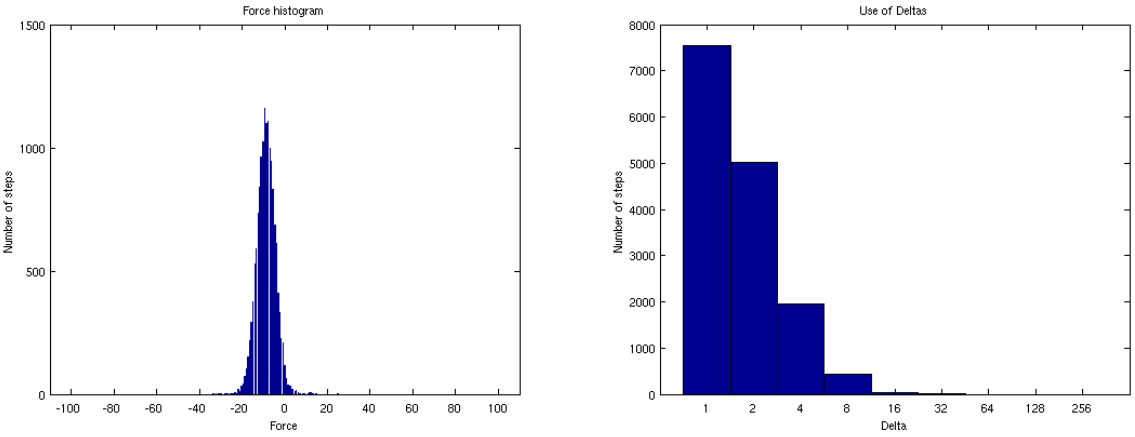

FIGURE 5.29: Inverted pendulum (Q-learning/ER)(100N):
Force histogram and use of Deltas. Mean force magnitude: 3.2352

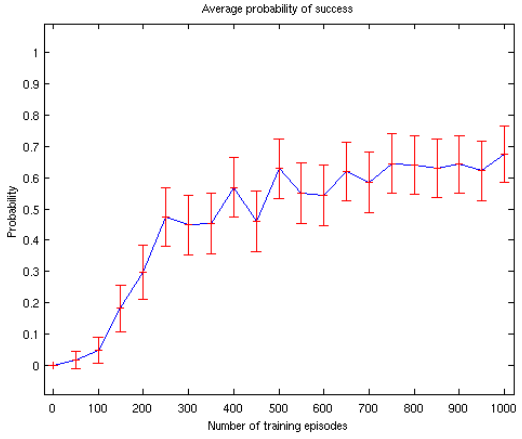

FIGURE 5.30: Inverted pendulum (Q-learning/ER)(100N): Average probability of success.

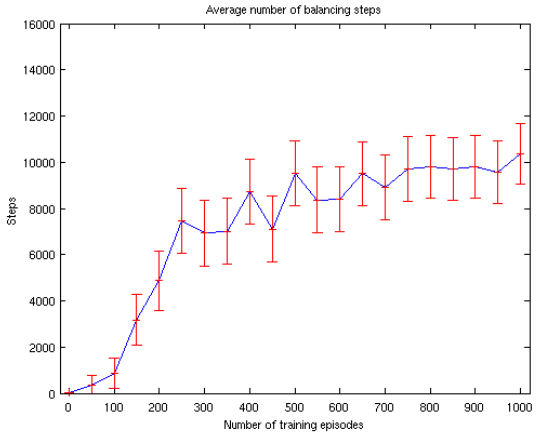

FIGURE 5.31: Inverted pendulum (Q-learning/ER)(100N): Average balancing steps.

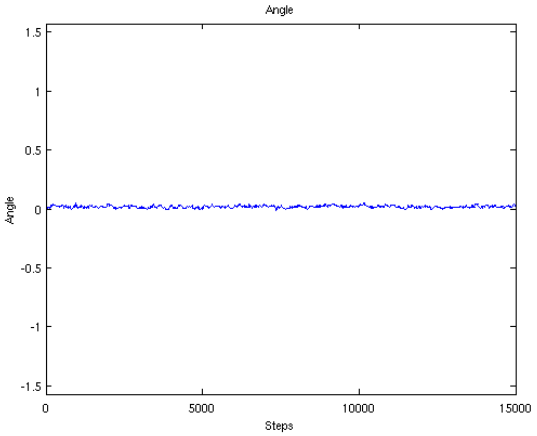

FIGURE 5.32: Inverted pendulum (Fitted Q iteration): Angle over time.(50N)

5.1.7 Fitted Q iteration

The experiments above were repeated, using Fitted Q iteration as the action selection algorithm. The same set of basis functions where used as the approximation architecture which was fitted to training set, using least squares regression.

A rather unexpected behavior was observed during testing. Although the algorithm did not appear to diverge with increased number of iterations, the policies did not necessarily improve. In fact although policies produced at the twentieth step where very successfull, for the same set of samples, if the algorithm was allowed to run up to one hundred iterations, almost none of the policies where still able to balance the pendulum. Since the twentieth iteration seemed to produce consistently good results, training for all experiments was cut off at that point.

Figures 5.32 through 5.34 and 5.35 through 5.37 show the individual runs results for the $[-50\ Newtons,\ 50\ Newtons]$ and $[-100\ Newtons,\ 100\ Newtons]$ ranges respectively. As we can see they are comparable to LSPI and Q-learning, although they exhibit worse mean force magnitude.

Figures 5.38 through 5.41 show the success rate and average balancing steps of the resulting policies. The results are very good approaching 100% success rate for 350 training episodes for the 50 Newton controller and 150 training episodes for the 100 Newton controller. This is far better than what the other two algorithms achieved, although as mentioned before the solutions where a bit lacking, as far as mean force magnitude is concerned.

A neural based implementation was also tested with mixed results. It was very successfull at balancing the pendulum with very few samples, but its performance at the applied

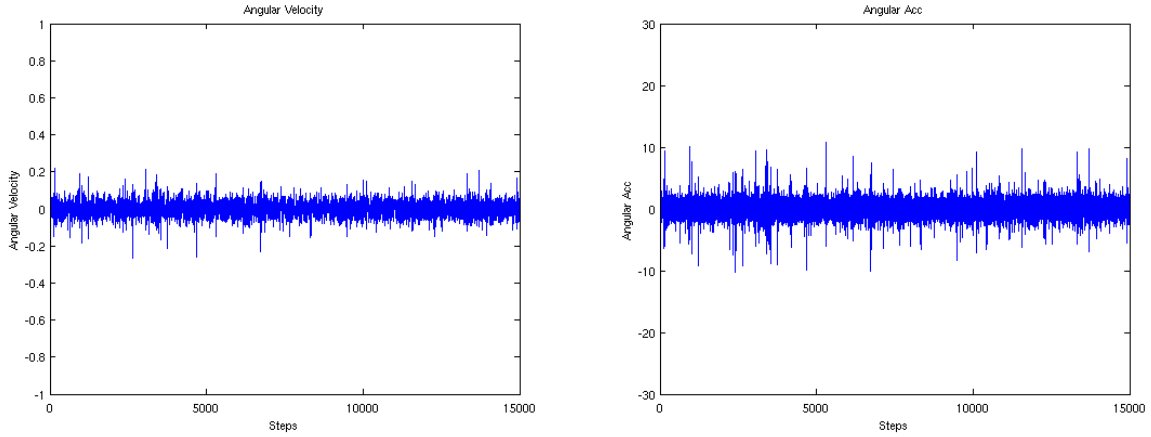

FIGURE 5.33: Inverted pendulum (Fitted Q iteration)(50N): Angular velocity and Acceleration over time.

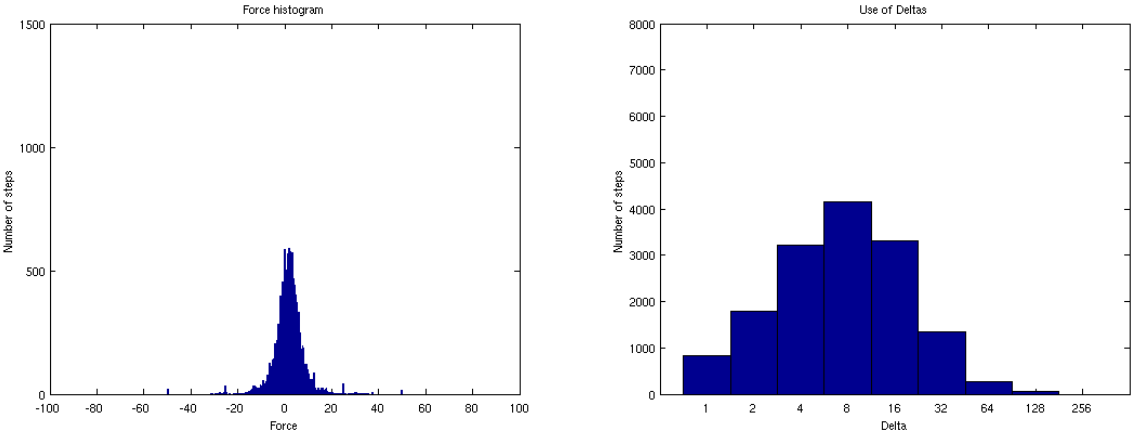

FIGURE 5.34: Inverted pendulum (Fitted Q iteration)(50N): Force histogram and use of Deltas. Mean force magnitude: 4.4526

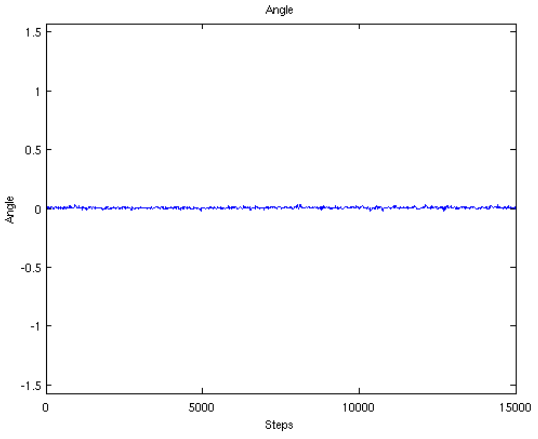

FIGURE 5.35: Inverted pendulum (Fitted Q iteration): Angle over time.(100N)

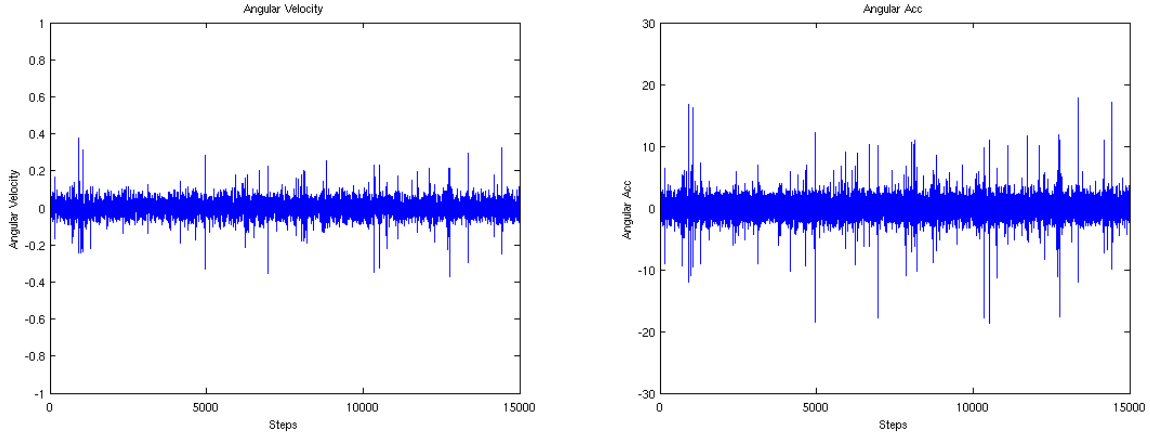

FIGURE 5.36: Inverted pendulum (Fitted Q iteration)(100N): Angular velocity and Acceleration over time.

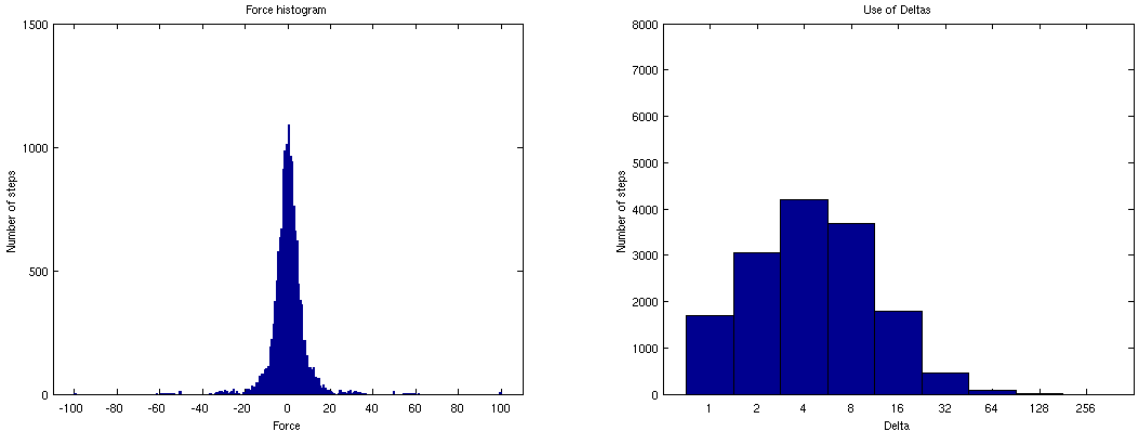

FIGURE 5.37: Inverted pendulum (Fitted Q iteration)(100N): Force histogram and use of Deltas. Mean force magnitude: 4.8124

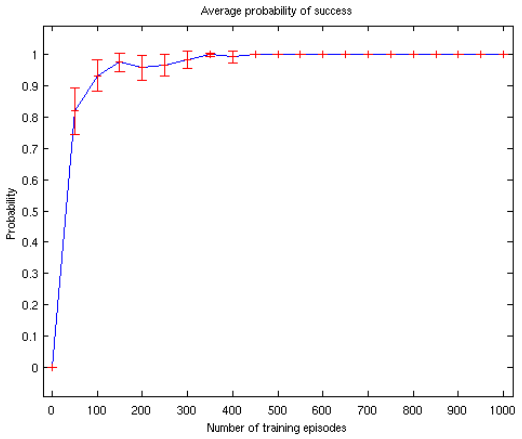

FIGURE 5.38: Inverted pendulum (Fitted Q iteration)(50N): Average probability of success.

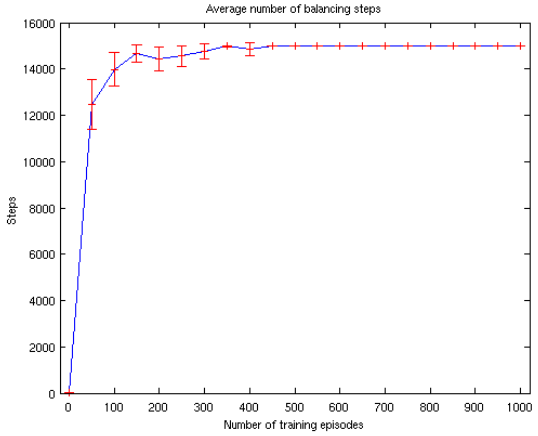

FIGURE 5.39: Inverted pendulum (Fitted Q iteration)(50N): Average balancing steps.

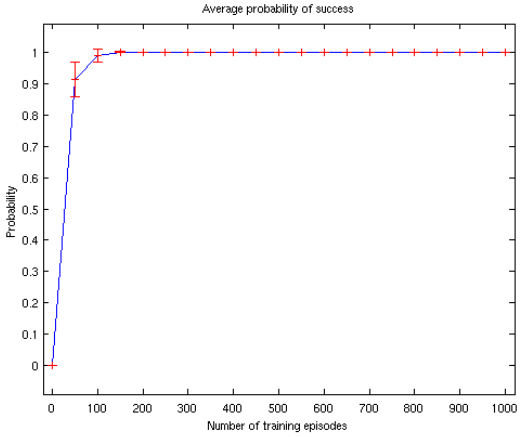

FIGURE 5.40: Inverted pendulum (Fitted Q iteration)(100N): Average probability of success.

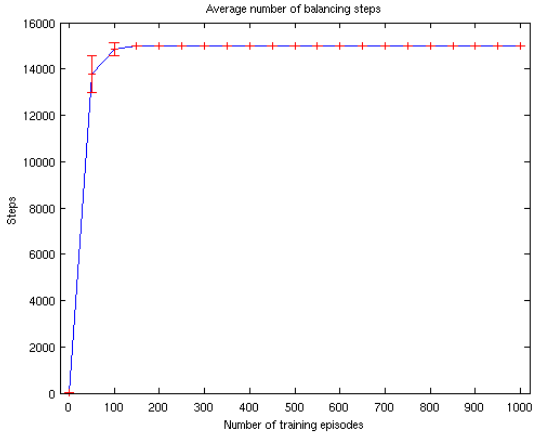

FIGURE 5.41: Inverted pendulum (Fitted Q iteration)(100N): Average balancing steps.

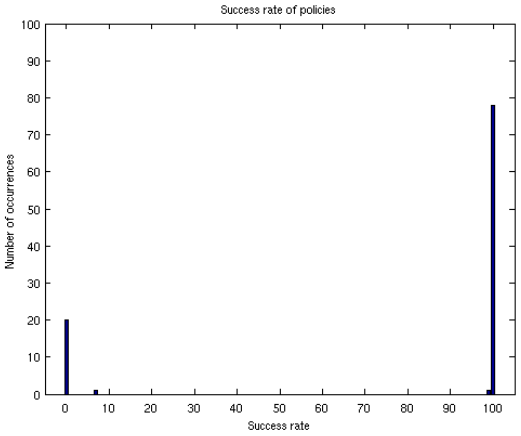

FIGURE 5.42: Inverted pendulum (LSPI)(50N): Policy success rate for 250 training episodes.

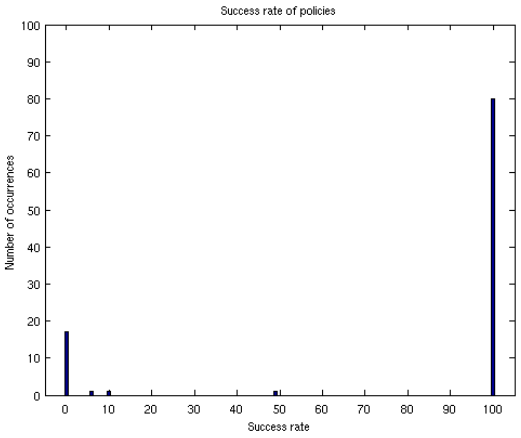

FIGURE 5.43: Inverted pendulum (LSPI)(100N): Policy success rate for 250 training episodes.

mean force magnitude was not very competent. Increasing the number of samples did not improve the situation, while in some cases it resulted in even choppier control.

5.1.8 Characterization of resulting policies

While conducting the above experiments we noticed a particularly interesting property of the resulting policies. In the majority of cases, the resulting controller could either succeed in balancing the pendulum all the time, or none of the time. Policies that managed to balance the pendulum occasionally, but failed the rest of the time where very rare, even for Q-learning. This is a very desireable property to have; it means that with little testing, one can have quite a good idea about the quality of a particular policy. Figures 5.42 through 5.44 show the policy success rate for 250 training episodes from the previous experiments.

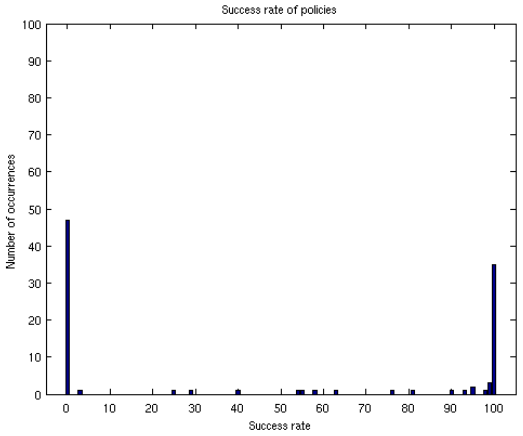

FIGURE 5.44: Inverted pendulum (Q-learning/ER)(100N): Policy success rate for 250 training episodes.

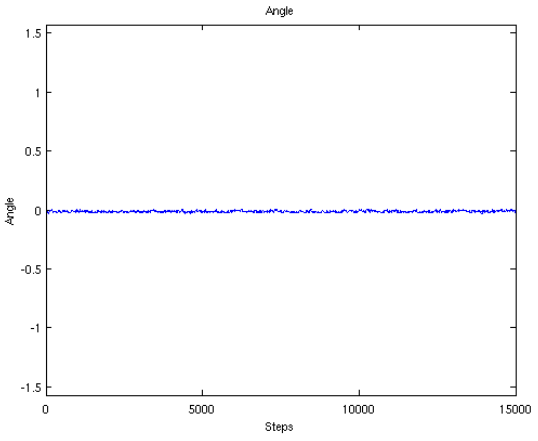

FIGURE 5.45: Inverted pendulum (LSPI 3-action): Angle over time.(50N)

5.1.9 Comparison with the discrete three action controller

Figures 5.45 through 5.47 and 5.48 through 5.50 show the performance of the original three action controllers, for the left force LF (-50 Newtons), right force RF ($+50$ Newtons), or no force NF (0 Newtons) and left force LF (-100 Newtons), right force RF ($+100$ Newtons), or no force NF (0 Newtons) domains respectively.

Comparing these controllers with the ones presented on section 5.1.2, we can conclude that the continuous controllers have a significant advantage over their discrete counterparts; especially so for the 100 newton case. One important thing to note when interpreting these results is that the performance of the discrete controllers is purely theoretical. It assumes that the system is able to go from +100 newtons to 0 newtons, or even worse from +100 newtons to -100 newtons instantaneously. Even though the simulation model does not prevent this, it would probably not be a realistic assumption

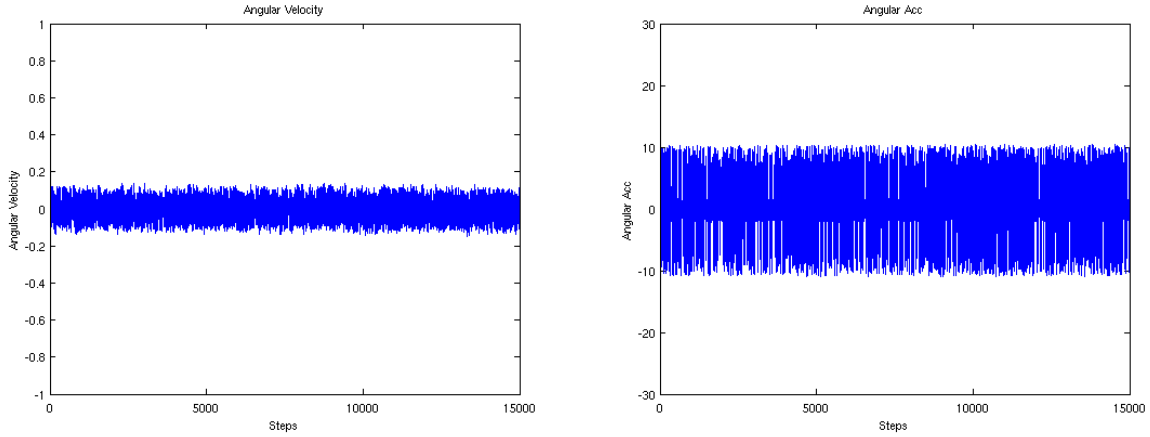

FIGURE 5.46: Inverted pendulum (LSPI 3-action)(50N): Angular velocity and Acceleration over time.

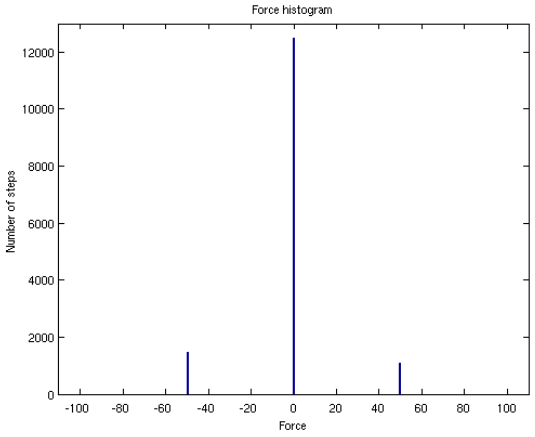

FIGURE 5.47: Inverted pendulum (LSPI 3-action)(50N): Force histogram. Mean force magnitude: 8.44

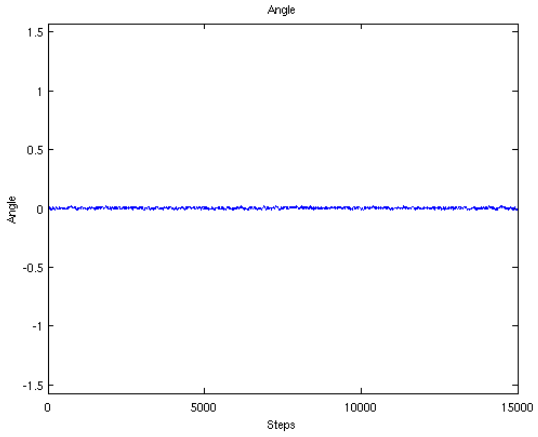

FIGURE 5.48: Inverted pendulum (LSPI 3-action): Angle over time.(100N)

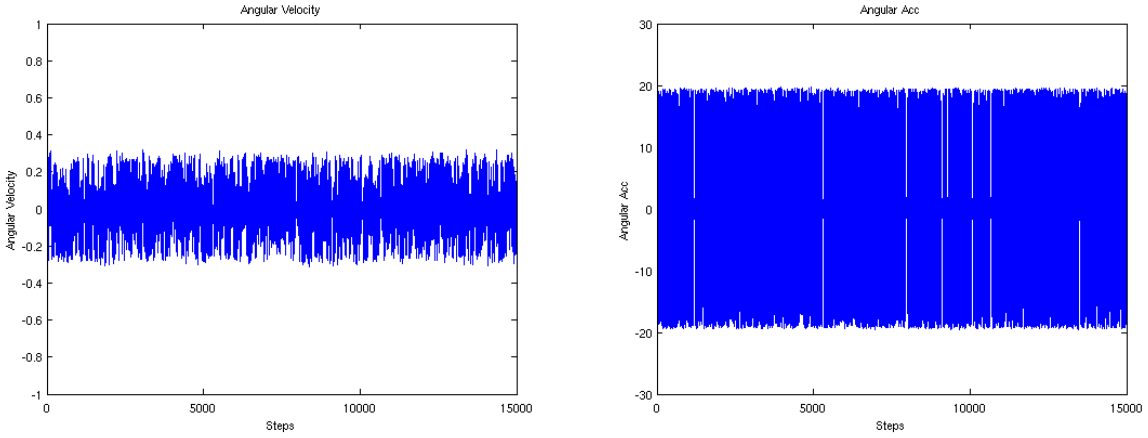

FIGURE 5.49: Inverted pendulum (LSPI 3-action)(100N): Angular velocity and Acceleration over time.

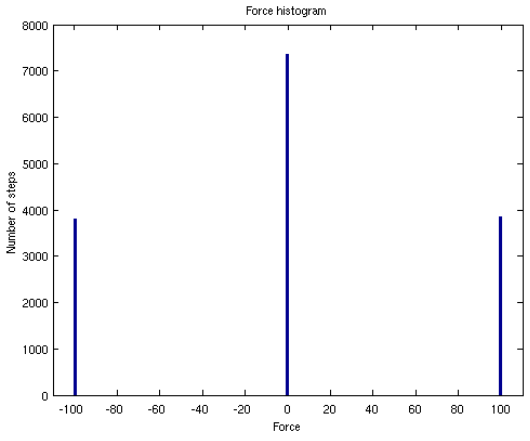

FIGURE 5.50: Inverted pendulum (LSPI 3-action)(100N): Force histogram. Mean force magnitude: 51.0133

for a real implementation running a control loop at 50 Hz. Even if the actuators were strong enough to achieve this, safeguards would have to be put in place to limit the rate at which the output can change. This is to prevent mechanical and/or electrical stresses from damaging the system. For the continuous controllers this behavior results naturally and can be enforced by limiting the Δ_{max} parameter of the algorithm. For our experiments Δ_{max} was the same as CA_{res} which resulted in at least two steps for the entire range. (if the controller is at one end of the spectrum with $\Delta = \Delta_{max}$ it can only go as far as the middle with a single step; the decision will change sign resulting in $\Delta = \Delta_{max}/2$)

5.2 Bicycle Balancing and Riding

The goal in the bicycle balancing and riding problem [18], is to learn to balance and ride a bicycle to a target position located 1 km away from the starting location. Initially, the bicycle's orientation is at an angle of 90^0 to the goal. The state description is a six-dimensional real-valued vector (θ, $\dot{\theta}$, ω, $\dot{\omega}$, $\ddot{\omega}$, ψ), where θ is the angle of the handlebar, ω is the vertical angle of the bicycle, and ψ is the angle of the bicycle to the goal. The actions are the torque τ applied to the handlebar (-2 to $+2$) and the displacement of the rider υ (-0.02 to $+0.02$). The state vector was augmented to include the current displacement of the rider and the current torque applied to the handlebar producing (θ, $\dot{\theta}$, ω, $\dot{\omega}$, $\ddot{\omega}$, ψ, τ, υ). The controller makes a decision to increase or decrease each variable at a time, giving a total of 4 combined actions ({-1,-1}, {-1,1}, {1,-1}, {1,1}). The noise in the system is a uniformly distributed term in [-0.02, +0.02] added to the displacement component of the action. The dynamics of the bicycle are based on the model of Randløv and Alstrøm (1998) and the time step of the simulation is set to 0.01 seconds. The state-action value function Q(s, a) for a fixed action a, is approximated by a linear combination of 29 basis functions $(1, \omega, \dot{\omega}, \omega^2, \dot{\omega}^2, \omega\dot{\omega},\ \upsilon, \upsilon\omega, \upsilon\dot{\omega}, \upsilon\omega\dot{\omega},$ $\theta, \dot{\theta}, \theta^2, \dot{\theta}^2, \theta\dot{\theta},\ \tau, \tau\theta, \tau\dot{\theta}, \tau\theta\dot{\theta},\ \omega\theta, \omega\theta^2, \omega^2\theta, \tau\upsilon,\ \psi, \psi^2, \psi\theta, \bar{\psi}, \bar{\psi}^2, \bar{\psi}\theta)^T$, where $\bar{\psi} = \pi - \psi$ for $\psi > 0$ and $\bar{\psi} = -\pi - \psi$ for $\psi < 0$. Note that the state variable $\ddot{\omega}$ is completely ignored. This block of basis functions is repeated for each of the 4 actions, giving a total of 116 basis functions (and parameters).

The following shaping reward was used:

$$-\frac{240}{\pi} \times \omega + \frac{(lastdistance - distance)}{(v \times dt)} \times 1.2$$

The first component $-\frac{240}{\pi} \times \omega$, penalizes large vertical angles for the bicycle, while the second component $\frac{(lastdistance - distance)}{(v \times dt)} \times 1.2$, rewards progress toward the goal and penalizes movement away from it. (*distance* and *lastdistance* is the current and previous distance of the bicycle to the goal respectively, v is the velocity of the bicycle and dt is the duration of the time step). Note that the algorithm sees the combination of these two rewards, as a single combined reward and not two different terms.

Each episode during sample collection was allowed to run for a maximum of 15 steps. If episodes of sample collection with a random policy, were allowed to run until the bicycle crashed, the distribution of samples would have been heavily biased, towards parts of the state space where the bicycle has entered a course toward crashing and was unrecoverable.

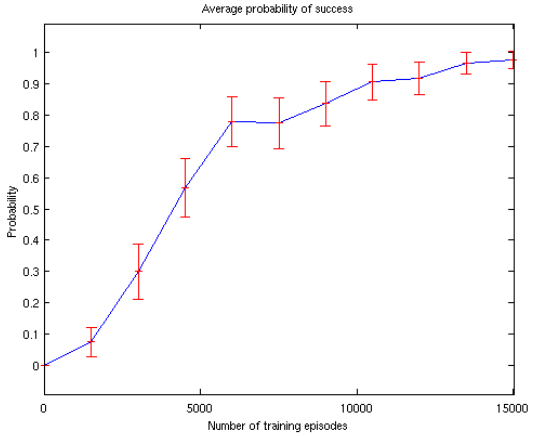

FIGURE 5.51: Bicycle (LSPI): Average probability of success.

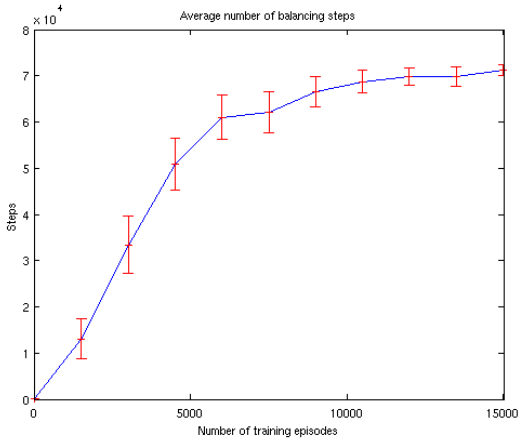

FIGURE 5.52: Bicycle (LSPI): Average number of balancing steps.

5.2.1 LSPI

Figures 5.51 through 5.56 show the performance of the learned controllers as a function of the number of training episodes. For each size of training episodes, the learned policy was evaluated 100 times in order to estimate accurately the success probability and the average number of balancing steps. This experiment was repeated 100 times for the entire horizontal axis, to obtain average results over different sample sets and the 95% confidence intervals. Each episode was allowed to run for a maximum of 72000 steps.

We can see from the results that LSPI needs quite a few training episodes in order to consistently provide good policies. This is a result not only of the complexity of the problem, but also of the very high noise term (has the same magnitude as the control action for the displacement component). Nevertheless it is obvious that as the number of training episodes increases, the resulting policies become consistently more successfull. For 15000 training episodes (225000 samples) the expected probability of

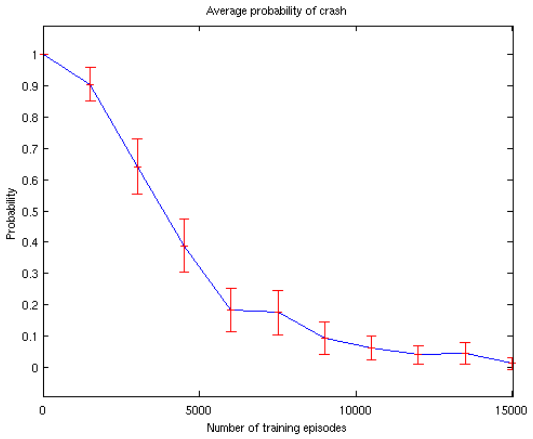

FIGURE 5.53: Bicycle (LSPI): Average probability of crash.

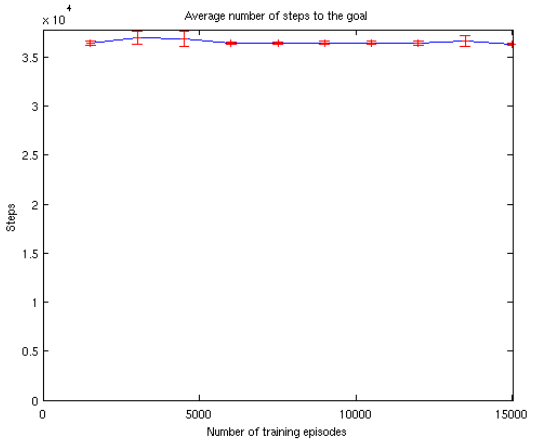

FIGURE 5.54: Bicycle (LSPI): Average number of steps to the goal.

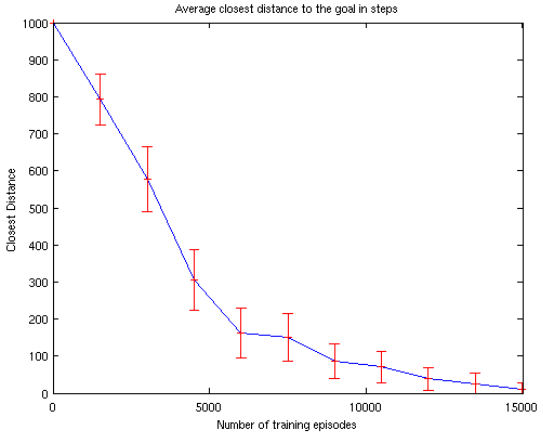

FIGURE 5.55: Bicycle (LSPI): Average closest distance to the goal in steps.

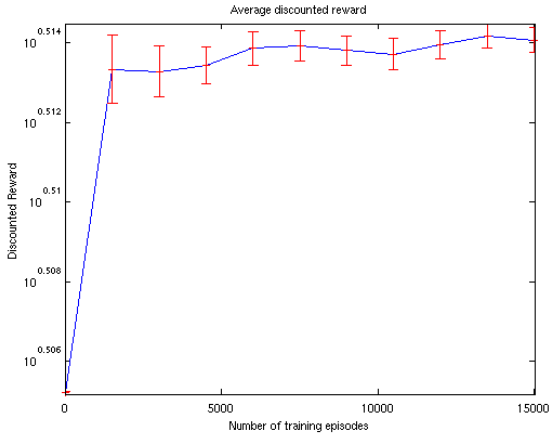

FIGURE 5.56: Bicycle (LSPI): Average discounted reward.

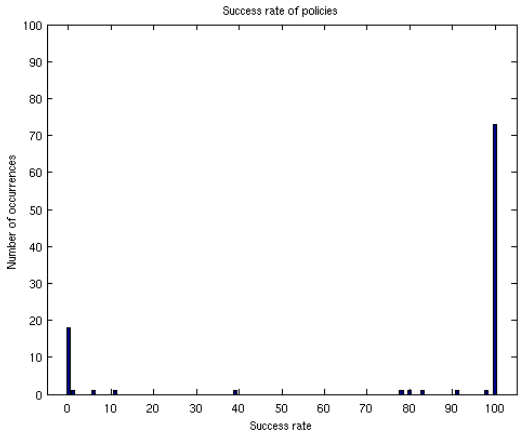

FIGURE 5.57: Bicycle (LSPI): Policy success rate for 6000 training episodes.

success is 97.56%, the expected probability of crashing is 1.38% and the expected number of balancing steps is 71231.

It is interesting to note, that the property we noticed before at the inverted pendulum domain, holds for the bicycle domain too. The majority of policies either succeed all the time or none of the time. Figure 5.57 shows the success rate for policies at 6000 training episodes.

5.2.2 Fitted Q iteration

The experiments above were repeated, using Fitted Q iteration as the action selection algorithm. The same set of basis functions where used as the approximation architecture which was fitted to training set, using least squares regression.

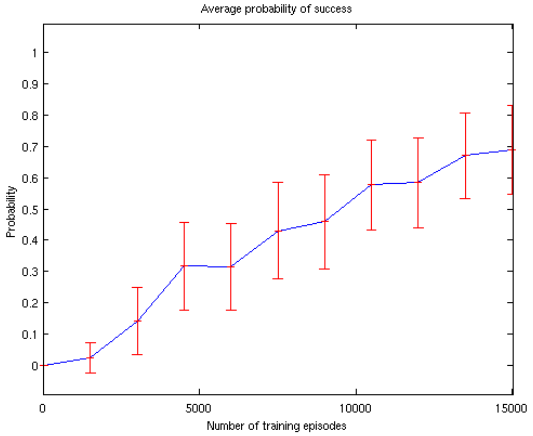

FIGURE 5.58: Bicycle (Fitted Q iteration): Average probability of success.

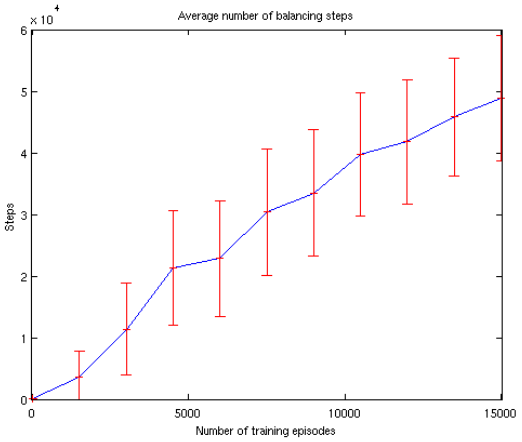

FIGURE 5.59: Bicycle (Fitted Q iteration): Average number of balancing steps.

The algorithm was very sensitive on the selection of the number of iterations N, much more than it was for the Inverted Pendulum domain. A number of experiments showed that $N = 50$ was a relatively good choice, although as figures 5.58 through 5.63 show its performance was not satisfactory. It is possible that there exists a N that would produce much better results.

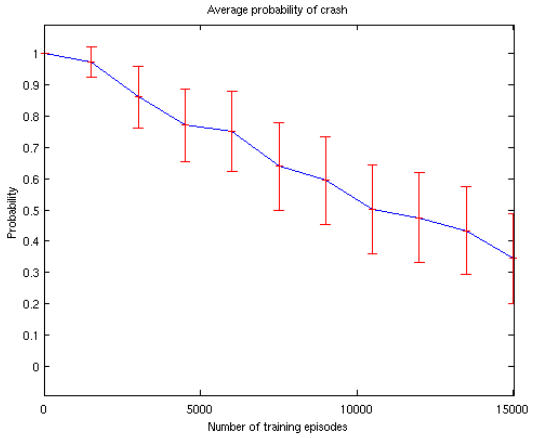

FIGURE 5.60: Bicycle (Fitted Q iteration): Average probability of crash.

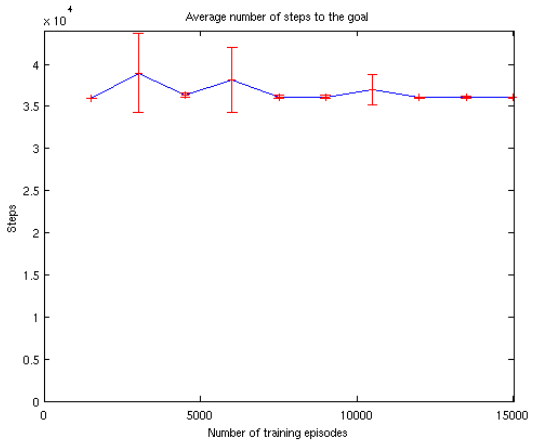

FIGURE 5.61: Bicycle (Fitted Q iteration): Average number of steps to the goal.

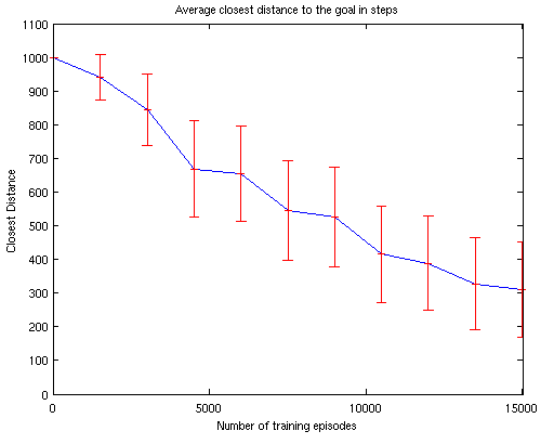

FIGURE 5.62: Bicycle (Fitted Q iteration): Average closest distance to the goal in steps.

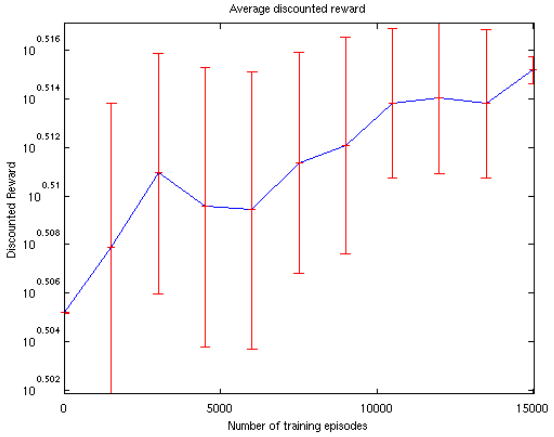

FIGURE 5.63: Bicycle (Fitted Q iteration): Average discounted reward.

Chapter 6

Discussion and Conclusion

6.1 Strengths and weaknesses

The method presented in this thesis, has a number of advantages that makes it a good candidate for domains where continuous actions are required:

- One of the greatest advantages of the proposed scheme, is its simplicity. It is very easy to implement, both conceptually and in terms of the amount of code required.
- Requires little, or no tuning. As soon as the desired resolution is decided, (that can be part of the problem statement) the only parameter left is K, which can be set to 2 with very competent results for most cases.
- Requires only 2 actions from the Reinforcement Learning algorithm of choice. This allows for very fast and efficient implementations of the policy function.
- Easily achieves resolutions impossible to reach with discrete actions and better than what is practically possible with popular approximation techniques.
- Can be used in conjunction with any Reinforcement Learning algorithm with discrete actions and is not tied to any particular implementation. Discrete implementations can be adapted to the continuous framework with little effort.
- Can be used in an online, offline, on-policy or off-policy setting.
- Has very low computational requirements. As was shown, the entire algorithm requires only a small number of additions, subtractions, shift operations and conditional statements. No divisions or multiplications are necessary and very little memory is required. This makes it perfect for embedded platforms with limited resources.

- Has better scalability properties than most (if not all) existing implementations, when the number of controlled variables increases.
- The "minimum required effort" behavior exhibited by the algorithm makes it an obvious choice for applications where it is important not to waste resources, such as mobile robotics.
- Results in smooth control, while being able to take fast action when the situation requires it.

Of course everything comes at a cost. There are three drawbacks with this approach:

1. As was mentioned before, this technique depends on temporal locality of actions. In fast changing domains we may have to decrease the time step (increase the rate at which our controller makes decisions). Obviously in domains that exhibit no temporal locality, this method is not suitable.
2. The resulting controller is not able to choose any possible action instantaneously. This may seem as a serious drawback, but for most cases, for a small enough time step, it doesn't hurt performance. A very important factor, that computer simulations usually overlook, is that on an actual system, changing the output too fast would cause severe mechanical and/or electrical stresses. In fact even on models that allow such abrupt changes, most implementations have safeguards to limit the rate at which the output can change.
3. The state space of the problem that we are trying to solve is now more complex. It depends on the original state variables, plus A, Δ and e_{n-1}. As we have seen adding A to the state vector is enough for most problems. Although it definitely puts a strain to our algorithm of choice for solving the problem, its impact is far less than that of increasing the number of possible decisions beyond a certain point.

We believe that the benefits far outweight the costs and the proposed approach is a viable choice, for Learning Continuous-Action Control Policies.

6.2 Future Work

In all applications presented in this thesis, the action space is divided into equally sized intervals. On a number of applications there may be areas of the state space that require finner resolution, while others are less important. Therefore, a skewing function could

be used, to distort the action space, to create a more appropriate fit for the available resolution.

There may be domains where adding more actions would be more appropriate than decreasing the time step. In such cases one could have four or more actions, where two will increase/decrease the continuous variable by a factor Δ, two by a factor $n \times \Delta$, and so on.

Sometimes dealing with a POMDP may be a serious drawback. In such cases, non adaptive schemes, where Δ is a constant and the state does not depend on e_{n-1}, can easily be implemented. Depending on the resolution and the reaction time desired, any number of actions for increasing/decreasing the continuous action by a constant amount, could be used.

The Δ update equation $\Delta_n \leftarrow \Delta_{n-1} K^{e_n \times e_{n-1}}$ is not the only valid rule for adaptive schemes. Other schemes, possibly with different growth and decay rates, offering better stability or other desirable properties, are not hard to develop.

A large amount of research, has been devoted on fields outside Reinforcement Learning to find ways to represent analog (continuous) signals with digital (discrete) means. Many of these results could potentially be adopted, providing insight from mature areas, to developing ones.

6.3 Conclusion

Inspired by methods used in telecommunications systems, this thesis has introduced a methodology for Learning Continuous-Action Control Policies in domains exhibiting temporal locality.

The computational efficiency of the algorithm has been demonstrated, along with its main strengths and drawbacks. Scalability, along with practical considerations have been addressed and potential sources of confussion have been clarified.

The proposed approach has been tested successfully, at the inverted pendulum and bicycle domains, using LSPI, Q-learning with experience replay and Fitted Q iterations. The experiments have demonstrated that the success of the controller depends on the quality of the learning algorithm used.

Appendix A

Explanation of Diagrams

A.1 Angle, Velocity and Acceleration Curves

Angle, Velocity and Acceleration curves are plotted against the number of steps. Each step is 0.02 seconds long, so they could be interpreted as the progress of each variable over time for the duration of a single experiment. The more these variables are concentrated around zero, the better since it means smoother control.

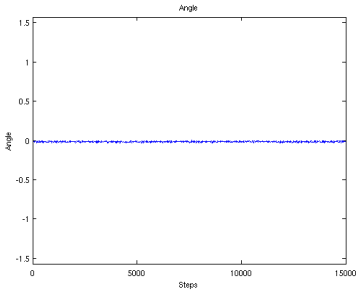

FIGURE A.1: Inverted pendulum (LSPI) s = $(\theta, \dot{\theta}, F)$: Angle over time.(100N)

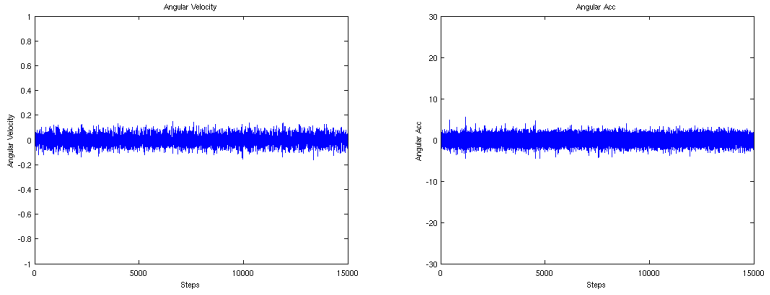

FIGURE A.2: Inverted pendulum (LSPI)(100N) s = $(\theta, \dot{\theta}, F)$: Angular velocity and Acceleration over time.

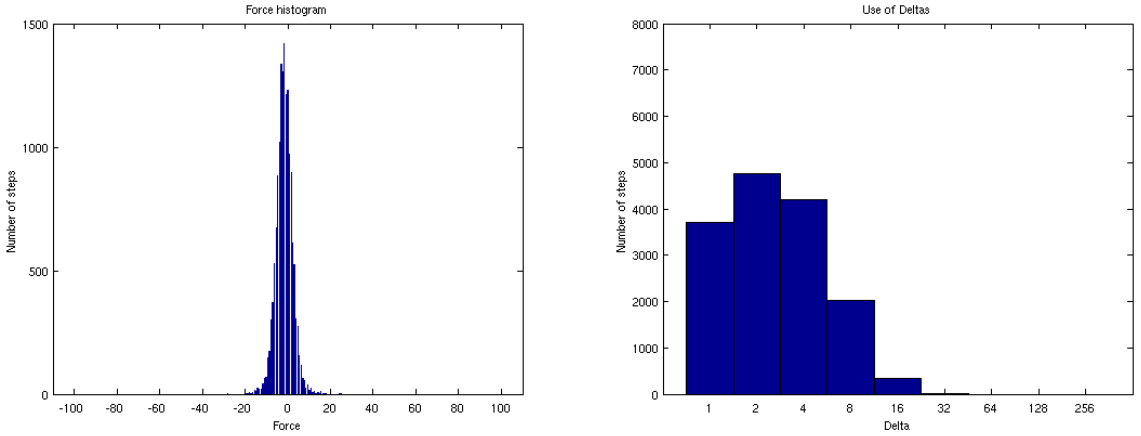

FIGURE A.3: Inverted pendulum (LSPI)(100N) s = $(\theta, \dot{\theta}, F)$:
Force histogram and use of Deltas. Mean force magnitude: 3.2352

A.2 Force and Delta histograms

Figure A.3 (left) is a histogram of the applied force, for a single experiment. The horizontal axis denotes the possible force values, while the vertical axis is the number of steps each force value was used in the duration of the experiment. Use of low force magnitudes is usually better, since it translates to less power consumption and less mechanical/electrical stresses.

Figure A.3 (right) is a histogram of Δ values used, again for a single experiment. The horizontal axis denotes the possible Δ values, while the vertical axis is the number of steps each delta was used in the duration of the experiment. Use of small Δ values is usually better, since it translates to finer control and efficient use of the available resolution.

Bibliography

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

[2] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts, 1998.

[3] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[4] Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, United Kingdom, 1989.

[5] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res.*, 6:503–556, 2005. ISSN 1533-7928.

[6] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research (JMLR)*, 4:1107–1149, 2003.

[7] John G.Proakis and Masoud Salehi. *Communication Systems Engineering*. Prentice Hall, 2001.

[8] Chris Gaskett, David Wettergreen, and Er Zelinsky. Q-learning in continuous state and action spaces. In *Proc. of the 12th Australian Joint Conference on Artificial Intelligence*, pages 417–428. Springer-Verlag, 1999.

[9] H. m. Gross, V. Stephan, and M. Krabbes. A neural field approach to topological reinforcement learning in continuous action spaces. In *Proc. of WCCI-IJCNN'98, Anchorage*, pages 1992–1997. IEEE Press, 1998.

[10] Juan Carlos Santamaría, Richard S. Sutton, and Ashwin Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6:163–218, 1998.

[11] Claude Touzet. Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 22:251–281, 1997.

[12] Thomas Strösslin and Wulfram Gerstner. W.: Reinforcement learning in continuous state and action space. In *Artificial Neural Networks - ICANN*, 2003.

[13] Kenji Doya. Temporal difference learning in continuous time and space. In *Advances in Neural Information Processing Systems 8*, pages 1073–1079. MIT Press, 1996.

[14] Augustine O. Esogbue and Warren E. Hearnes. A learning algorithm for the control of continuous action set-point regulator systems. *Journal of Computational Analysis and Applications*, Volume 1, Number 2 / April, 1999:121–234, 1999.

[15] Brian Sallans and Geoffrey E. Hinton. Reinforcement learning with factored states and actions. *J. Mach. Learn. Res.*, 5:1063–1088, 2004. ISSN 1533-7928.

[16] A. Lazaric, M. Restelli, and A. Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In *Advances in Neural Information Processing Systems 20*, pages 833–840, Cambridge, MA, 2008. MIT Press.

[17] Hua O. Wang, Kazuo Tanaka, and Michael F. Griffin. An approach to fuzzy control of nonlinear systems: Stability and design issues. *IEEE Transactions on Fuzzy Systems*, 4(1):14–23, 1996.

[18] Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of The Fifteenth International Conference on Machine Learning*, pages 463–471, Madison, Wisconsin, 1998.