

On-Line Model-Based Continuous State Reinforcement Learning Using Background Knowledge

Bernhard Hengst

University of New South Wales
Computer Science and Engineering
bernhardh@cse.unsw.edu.au
<http://www.cse.unsw.edu.au>

Abstract. Without a model the application of reinforcement learning to control a dynamic system can be hampered by several shortcomings. The number of trials needed to learn a good policy can be costly and time consuming for robotic applications where data is gathered in real-time. In this paper we describe a variable resolution model-based reinforcement learning approach that distributes sample points in the state-space in proportion to the effect of actions. In this way the base learner economises on storage to approximate an effective model. Our approach is conducive to including background knowledge to speed up learning. We show how different types of background knowledge can be used to speed up learning in this setting. In particular, we show good performance for a weak type of background knowledge by initially overgeneralising local experience.

Keywords: Artificial Intelligence, Machine Learning, Reinforcement Learning, Robotics, Background Knowledge, Function Approximation, Continuous State, Model-Based.

1 Introduction

In robotics it is often the case that we wish to control a real, possibly non-linear, continuous dynamic system when the model is unknown. In principle reinforcement learning (RL) holds out the promise to solve this control problem by simply letting the learner interact with the system. By specifying a reward signal to guide the learner to the control goal the theory suggests that a policy can be learned over time given enough training experience. Unfortunately in practice the sheer number of trials required to learn a good policy becomes prohibitive. It may take a long time to learn anything useful, the real system may break down before learning is complete, or it may be costly to run a system sub-optimally for the duration of the training phase. It is clearly desirable to have methods that can speed up learning by finding a good policy with as few trials as possible. Sometimes we have an approximate model of the system and would like to use this as a starting point. However, many reinforcement learning

methods do not provide representations that are conducive to including background knowledge to help the learner. In this paper we describe a RL approach that:

- automatically changes the resolution of the representation based on the effect of actions. It stands to reason that the representation of the dynamics can be coarser where actions move the system over a greater distance. A sparser representation requires less trials to learn.
- is able to include background knowledge by generating or seeding the state transition dynamics in particular parts of the state space. We show how background knowledge can be introduced in a natural way to significantly speed learning.
- over-generalises local dynamics to larger parts of the surrounding state space to quickly find a working policy and then progressively refining the representation to improve performance over time.

RL has been studied and applied for well over a decade and we assume the reader has a basic understanding of its methods. Introductory material can be found in several AI texts, for example[1,2], or in specifically dedicated texts [3,4,5]. In brief, reinforcement learning involves an agent learning to take appropriate actions to maximise future reward by interacting with the environment. The reinforcement learning problem is usually couched in the form of a Markov decision problem $\langle S, A, T, R \rangle$, where S is a set of states, A is a set of actions, $T : S \times A \times S \rightarrow [0, 1]$ is a stochastic state transition function and $R : S \times A \times \mathbb{R} \rightarrow [0, 1]$ is a stochastic reward function. The transition function and the reward function define the *model* of the environment. The agent interacts with the environment by iteratively observing state $s \in S$, taking an action $a \in A$ and at the next time-tick receiving reward $r \in \mathbb{R}$, and observing the next state $s' \in S$. The aim is to learn an action *policy*, $\pi : S \rightarrow A$, to maximise a measure of future reward. This is usually achieved by learning an optimum value function, e.g. $Q^* : S \times A \rightarrow \mathbb{R}$ using *policy iteration*, from which follows the optimum policy $\pi^*(s) = \arg \max_a Q^*(s, a)$. So called *model-free* methods exist that learn the value function directly from environmental interaction, but we take the view in this paper that explicitly learning the model has advantages.

We will briefly review related work modelling continuous dynamic systems and methods for speeding up learning. We describe our approach to variable resolution modelling and how to include background knowledge. The presentation uses a simple cart to illustrate the concepts. Results for a cart showing that learning is speeded up by a factor of 2 and even more when more explicit background knowledge is provided.

2 Related Work

Sutton and Barto [4] show how reinforcement learning can be speeded up with the Dyna architecture. Transition experiences of the form $(s_t, a_t, r_{t+1}, s'_{t+1})$ are

retained, thereby model-learning the dynamics of the system. The trick is to play transitions back as simulated experience interleaved with real-experience during Q-learning. The relatively slow interactions for a real robot may be interleaved with many simulated experiences that help converge the value function much faster. Related methods to help speed learning are the use of eligibility traces that store transition experience and back up recent experience along the trace, and prioritised sweeping that prioritises back-ups of the value function in the reverse direction to the directionality of state transitions. All these methods rely on storing the model to some degree. The approach in this paper is to incrementally store all novel transition experiences and run Q-Learning until convergence or until the next real experience becomes available.

As we are modelling continuous systems with an infinite number of states, and RL uses finite tables, we need a method for *function approximation* to generalise the value of learned points to nearby query points. Function approximation can use any supervised machine learning method. Here we use a non-uniform case-based lazy-learner approach, similar to methods by Santamaria, et al [6] in which neighbouring cases are chosen based on the Euclidian distance d_i between the i -th case point and query point. This linear function approximator is a nearest neighbour function that uses a Gaussian kernel, $K(d_i) = \exp(-d_i^2/\tau^2)$, to weigh each i -th neighbour's contribution to the query point Q value, where τ is a smoothing parameter that controls the blending of values of the nearest neighbours. Gabel and Riedmiller further investigate the use of Case-Based Reasoning function approximation methods in reinforcement learning [7]. Where our approach differs is that we store the local model in each case and vary the resolution of cases in each dimension of the state space based on the effect of actions. We store separate cases for each action.

More recent work by Jong and Stone [8] also stores the model as a set of instances and uses kernel-based averages to approximate the value of a query state. Kernel-based averages are known to converge. We use a similar approach to represent the model and to approximate the value function. Our concern in this paper is how we incorporate and employ background knowledge about the dynamics of the system to help speed learning.

A different but related research topic is qualitative modelling and simulation [9]. Qualitative modelling reasons about continuous aspects of systems in logical form (without numbers). For example, the statement “acceleration increases velocity” imparts information about a system's dynamics without being specific about the numerical value about of the acceleration or the velocity. The objective is to reduce search time by reasoning and constraining the learner subject to the qualitative descriptions. We show how qualitative knowledge may be used to guide a RL by seeding the learner with compliant transitions. This suggests a method whereby qualitative modelling can bridge quantitative implementations in specific instances.

3 Approach

We start by introducing an intuitive example sufficient to illustrate the continuous modelling of a dynamic system. We then describe our model-based approach to RL and show how background knowledge can be incorporated.

3.1 Motivating Example

Our physical system consists of a frictionless wheeled 104kg cart moving in one dimension on a horizontal flat 40 meter table as shown in Figure 1. At a frequency of 6 Hertz the controller is able to sense the position and velocity of the cart and apply a positive force of 1000 Newtons through the wheels, a negative force of the same magnitude, or to let the cart coast under its own momentum. The aim of the reinforcement learning controller is to learn to bring the cart to rest at the goal position. We model the cart system as an instance of the `PhysicsVehicle` class in the open source real-time physics simulator `jMonkeyEngine` [10].

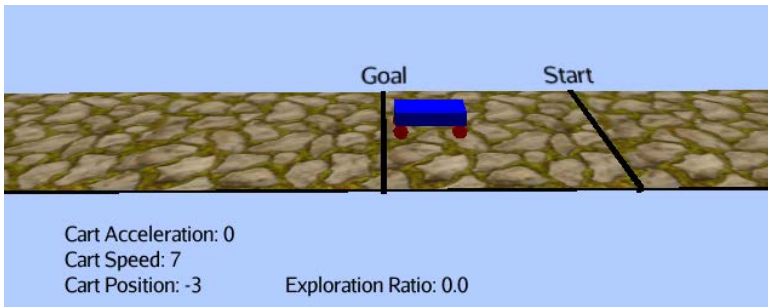


Fig. 1. The cart on a horizontal table as rendered by the real-time `jMonkeyEngine` 3D game and physics simulator. The positive direction is towards the left. The cart is started at random during learning and at -15 meters to periodically measure performance. The goal is to reach the origin in the middle of the table.

3.2 Method

As the reinforcement learner interacts with the environment it experiences the model of the system as successive tuples of the form (s, a, r, s') where taking action a in state s results in a reward r and a transition to the next state s' . The state s may be multi-dimensional, hence $s = (x_1, x_2, \dots, x_n)$ for an n -dimensional state vector. In the case of the cart the state has 2-dimensions: $s = (\text{position}, \text{velocity})$. We assume all dimensions are continuous.

Rather than throwing the experience away after updating the value function as one would in model-free RL, we store the model information as a *case*, subject to a resolution criterion. Experiencing similar transitions over and over again leads to diminishing information gain. We therefore only store new experiences when

the state s is greater than a threshold distance from previously stored cases with the same action for any one of the dimensions. The threshold itself is variable (subject to a minimum value) throughout the state space and depends on the effect of the action recorded for the nearest neighbouring state. If the following `isNeighbour` function returns true for any existing case the experience is not added as a new case.

Algorithm 1. Tests if existing case, `caseID`, covers the experience (s, a)

```
isNeighbour(s[], a, caseId, spread){
    if(cases[caseId].a!=a) return false;
    for (int d=0; d<dimS; d++) {
        v1 = s[d];
        v2 = cases[caseId].s[d];
        delta = cases[caseId].threshold[d];
        if(Math.abs(v1-v2)>spread*delta) return false;
    }
    return true;
}
```

The parameter *spread* in the above function regulates the resolution of the cases representing the system. It can be used to increase the resolution if required, for example to better represent stochasticity. We set *spread* = 0.9. Figure 2 shows the transition cases stored for the cart after plying the RL with random transitions for a long period.

It is reasonable to reduce the number of cases in parts of the state-space where the effect of actions leads to greater transition distances. For the cart the density of cases is reduced along the position dimension with an increase in the magnitude of the velocity, as the cart moves further at higher speed.

At each time-tick we find the optimum policy by repeatedly iterating through all the cases stored so far using a temporal difference Q-learning method in the spirit of the Dyna architecture, halting only after convergence or in an anytime fashion when the next experience arrives. As the next state s' for each case is unlikely to be equal to a state s from another case, we approximate the Q -values of s' from the neighbouring cases using a Gaussian kernel. Figure 3 illustrates the estimation of the Q -value of s' for action a' using Algorithm 2.

Without enough experience the controller is unlikely to find a solution. Nevertheless it solves the problem with whatever cases are available and attempts an optimal policy. Early in the learning, when there are no neighbours, we set the next Q values optimistically high to encourage exploration. During Q -learning we use a discount rate of 0.99 to ensure the value function remains bounded while discovering the model, and a learning rate of 1.0, making the tacit assumption that the system is deterministic.

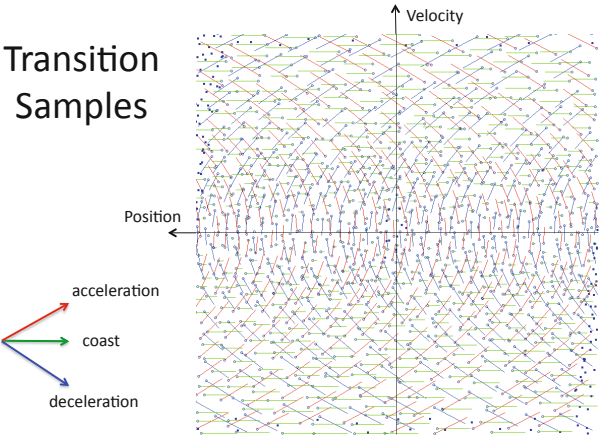


Fig. 2. Transition cases for the cart showing the effect of different actions in the position-velocity space. Note the reduction in resolution for position for higher velocity values. Solid dots indicate terminations cases - goal with reward 0, -100 otherwise.

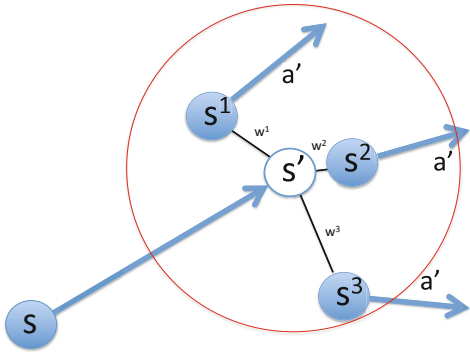


Fig. 3. Estimating successor Q -values from neighbouring cases. The red circle delimits the neighbourhood cases.

3.3 Inserting Background Knowledge in RL

Background knowledge includes all the assumptions, inductive bias (both representational and preference), parameter settings, and system dynamics provided to the learner beforehand. We focus here on background knowledge about the system dynamics. The language provided by the MDP model is used to describe this background knowledge, namely the state transition and reward function. With this representational approach it is possible to generate background knowledge as case samples and feed these to the learner beforehand, giving the learner a head-start. As an extreme example, if the dynamics of the system are known,

Algorithm 2. Estimate max Q-value of next state s' of case t from Neighbours

```

dimension estQneighbour[numA]; totWeight[numA];
ret = -100;
for(int n=0; n<numNeighbours(t); n++){
    tn = cases[neighbour(n)];
    dist = sharpness*distance(tn.s, s');
    weight = exp(-dist*dist);
    totWeight[tn.a] += weight;
    estQneighbour[tn.a] += weight*tn.Qvalue;
}
for(int nA=0;nA<numA;nA++) {
    if(totWeight[nA]<=0f)
        if(measuring) estQneighbour[nA] = -100; //act greedily
        else estQneighbour[nA] = 0f; //encourage exploration
    else estQneighbour[nA] /= totWeight[nA];
    if(estQneighbour[nA]>ret) ret = estQneighbour[nA];
}
return ret;

```

we could generate experience cases to cover the whole state-action space so that the learner would not require any training, effectively dynamic programming the optimal policy using the Q-Learner. In other situations, a qualitative description or an approximate model may be available that could be used to generate cases a priori to speed up learning.

It is also possible to generate additional cases from background knowledge during the learning process. For example when we know the system dynamics exhibits symmetry we could mirror experiences speeding up learning. In the following results we will show two other examples of the use of background knowledge to insert cases while learning. In one case we model common sense knowledge that physics is invariant to the inertial frame of reference, and in the other we over-generalise local experiences.

When we use background knowledge to insert approximate cases we label them as *provisional* and allow cases from real experience to progressively replace them. The objective is to quickly learn a satisficing [11] policy and improve it over time.

4 Results

In this section we show experiments with the cart that illustrate how background knowledge can be used to speed up learning.

We start the cart at a random position and velocity with the goal of coming to rest at the origin. We accept any position and velocity satisfying $|position| < 1$ meter and $|velocity| < 1$ meter/second as having reached the goal. For practical purposes we limit the position and velocity space to lie in a region ± 20 meters

and ± 20 meters/second and terminate a trial if the cart moves outside these limits, takes more than 50 time-steps to reach the goal, or reaches the goal. We administer a reward of -1 per time-step, implicitly setting the objective to reach the goal in minimum time. The minimum resolution for cases for both position and velocity is set at 0.9. The RL is initialised with optimistic Q values to encourage exploration. We periodically measure the performance of the controller by starting the cart at rest at position -15 meters as show in Figure 1. During this time we use the latest optimum greedy policy. Learning is turned off during performance measurement so that the results are not influenced by this observation process.

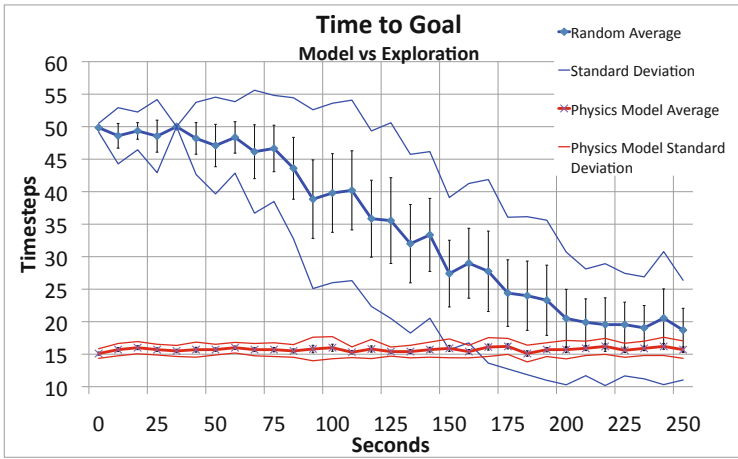


Fig. 4. The number of time-steps to reach the goal given the time in seconds of real-time training. The blue graph shows the results when learning without any extra background knowledge by taking random exploration actions. The red graph shows the results when the learner is given the physics model in the form of sample transitions that cover the state-action space and provides a benchmark of the best performance achievable with this controller i.e. 15.7 ± 1.1 . The experiments are each run between 10 and 20 times and the graphs show the standard deviation and the standard error.

The first two experiments were designed to set upper and lower bounds on the RL's performance. We plot the number of time-steps to reach the goal vs the number of seconds of real-time training. In the first experiment the learner was given no additional background knowledge and required to explore and learn an optimum policy just from interaction with the simulated real-time system. As shown in Figure 4 (blue plots) the RL took over 4 minutes on average to learn a good policy. At the other extreme, if the learner was supplied with the exact physical model of the cart by seeding the RL with sample transitions prior to learning similar to that show in Figure 2, the RL was able to learn and execute an optimal policy without any real-time training (Figure 4, red plots).

The standard deviation in performance is much greater when learning in real-time due to the hit and miss nature of the random exploration actions taken by the RL. The error bars indicate that after about 4 minutes our confidence even at 68% cannot separate the two experiments¹.

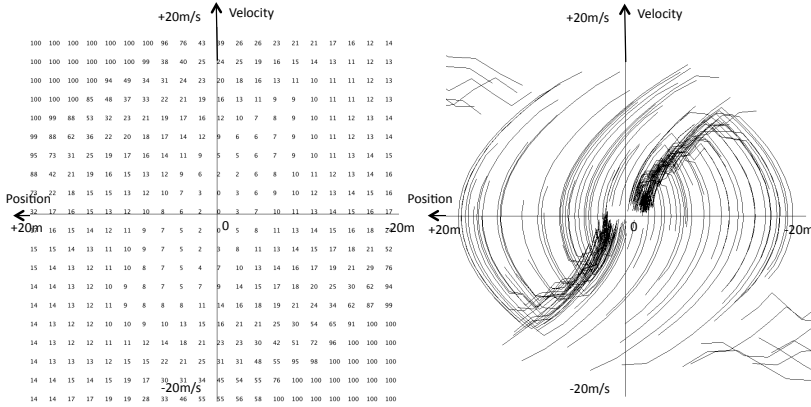


Fig. 5. RIGHT - $\max|Q|$ values sampled uniformly over the state-space, LEFT - typical trajectories from random starting positions

Figure 5 gives some further insight to the working of the RL. The left of the Figure shows the maximum $|Q|$ values sampled over the relevant state-space. As the goal becomes unreachable from certain parts of the state space values start to approach the undesirable termination value set to 100. The right shows the trajectories after learning when the cart is started at random locations. The trajectory to reach the goal area is not a simple straight line in the position-velocity space, but usually requires a carefully timed 2-phased strategy involving acceleration and deceleration.

In the next experiment we include the qualitative background knowledge that the physics of the cart is invariant to position. Since the table is uniformly level, the effect of an action on velocity is the same for any position, and the cart's change in position is also expected to be the same for any position.

To make use of this background knowledge, whenever we experience a transition at one position and velocity, we duplicate the experience and seed this transition for all positions for the same velocity. Figure 6, left shows the RL seeded with duplicated position invariant transition cases from just a few

¹ Both the standard deviation and the standard error were calculated assuming Gaussian distributed data. The termination of trials after 50 time-steps and the lower bound of the theoretical optimum policy means that the Gaussian assumption does not hold and the interpretation of the plot of these measures needs to take this into account.

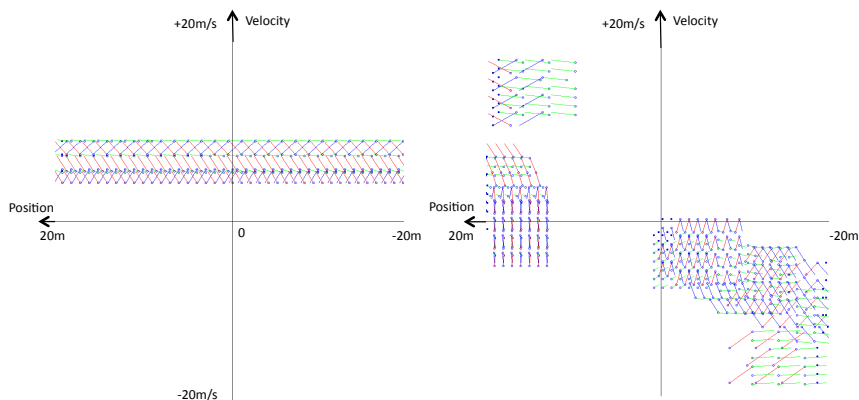


Fig. 6. RIGHT - The transition cases duplicated from background knowledge that physics is position invariant, and LEFT - Transitions seeded by overgeneralising from limited real experience

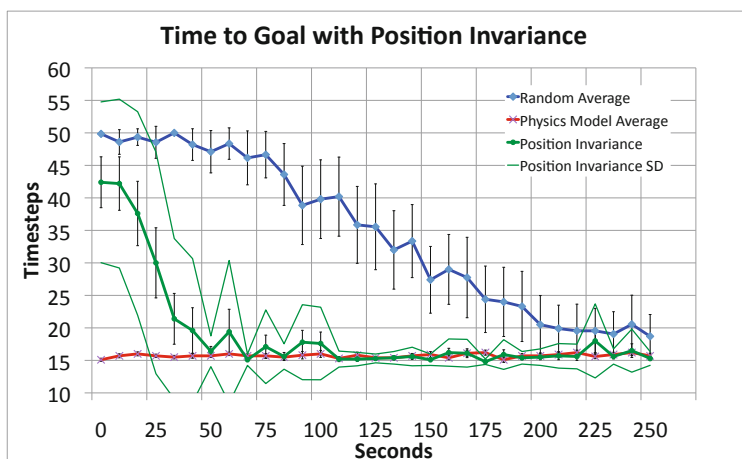


Fig. 7. The green graphs show the improvement in learning with the addition of the common-sense background knowledge, namely that the physics of the cart is the same at all positions on the table. Graphs from Figure 4 are retained for comparison.

real-time experiences. The improvement in learning is shown as the green plot in Figure 7. It is apparent that the inclusion of this background information has had a significant effect on learning time which is now reduced to less than a minute.

Position invariance may not always be appropriate background knowledge. It will not apply for example to the mountain car test-bed [12] where the surface is hilly. We tested a weaker form of background knowledge that would apply in many continuous dynamic systems. We assume that experienced transitions are

a good approximation in a larger neighbourhood and seed the RL accordingly. While this approximation may not produce an optimal solution, it does nevertheless find a solution quickly that can later be refined when more real experiences come to hand.

Figure 6 (right) shows this generalisation applied to several cases in a rectangular neighbourhood around a real-experience. In this way the state-action space can be quickly filled with approximate transitions to allow the learner to find a path to the goal. Figure 8 (black plots) charts the performance of this overgeneralisation approach. Other experiments are retained on the graph to show the relative performance. Overgeneralisation for the cart results in a two-fold speed-up, learning an optimal policy in about 2 minutes, and even after one minute of training, the RL has made good progress.

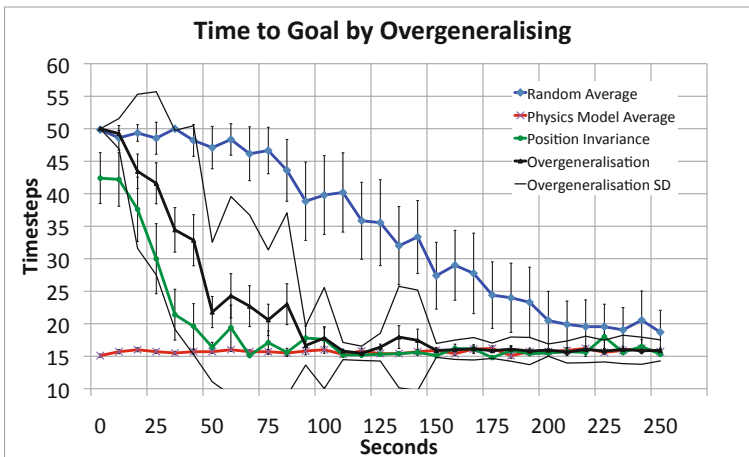


Fig. 8. The performance of the RL by over-generalising experience in a local neighbourhood. This weak type of background knowledge shows a significant speedup. The approximation is progressively improved with additional real experience over time.

5 Discussion

An advantage of learning the model of the system rather than just the value function is that the model can be reused to achieve alternate goals. For the cart, not only do we have a policy to bring the cart to rest from any position and velocity, but we can now set any $(position, velocity)$ as a goal and the RL can use the stored experience to learn a new policy with minimal on-line learning.

While the implementation has been developed for a general multi-dimensional continuous system we have only tested the RL algorithm on the cart problem. We would next wish to test it on standard problems such as the 2-dimensional mountain-car and the 4-dimensional pole and cart. The latter provides an opportunity to add background knowledge in the form of the symmetry of the

inverted pendulum. These problems are not expected to exhibit any surprises. The issue that will need attention is scaling the implementation to larger and higher dimensional spaces. For the current implementation neighbouring cases are precomputed and store as part of each case to make *Q*-Learning more efficient. For query points during execution we could use a *kd*-tree and for higher dimensions an approximate nearest neighbour algorithm. There are also several parameters that need setting, such as the minimum resolution and spread, that are problem dependent.

One application that we propose to tackle is bipedal walking that was previously reported using just a naive discretisation for function approximation and model-free learning [13]. In conclusion, we have outlined a variable-resolution model-based approach to tackle RL for continuous systems and have shown how to include various forms of background knowledge to help speed up learning.

References

1. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River (1995)
2. Mitchell, T.M.: Machine Learning. McGraw-Hill, Singapore (1997)
3. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. In: John Wiley & Sons, John Wiley & Sons, Inc., New York (1994)
4. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
5. Wiering, M., van Otterlo, M. (eds.): Reinforcement Learning: State of the Art. Adaptation, Learning, and Optimization, vol. 12. Springer (2012)
6. Santamaria, J.C., Sutton, R.S., Ram, A.: Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior* 6(2) (1998)
7. Gabel, T., Riedmiller, M.: CBR for State Value Function Approximation in Reinforcement Learning. In: Muñoz-Ávila, H., Ricci, F. (eds.) ICCBR 2005. LNCS (LNAI), vol. 3620, pp. 206–221. Springer, Heidelberg (2005)
8. Jong, N.K., Stone, P.: Compositional Models for Reinforcement Learning. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009, Part I. LNCS, vol. 5781, pp. 644–659. Springer, Heidelberg (2009)
9. Kuipers, B.: Qualitative simulation. *Artificial Intelligence* 29, 289–338 (1986)
10. jMonkeyEngine 3D Game Development SDK (2012), <http://jmonkeyengine.org/>
11. Simon, H.A.: Rational choice and the structure of the environment. *Psychological Review* 63(2), 129–138 (1956)
12. Moore, A.W.: Efficient memory-based learning for robot control. Technical Report UCAM-CL-TR-209, University of Cambridge, Computer Laboratory (November 1990)
13. Hengst, B., Lange, M., White, B.: Learning ankle-tilt and foot-placement control for flat-footed bipedal balancing and walking. In: 11th IEEE-RAS International Conference on Humanoid Robots (2011)