Guaranteed globally optimal continuous reinforcement learning

Hildo Bijl

***Abstract*—Self-learning and adaptable autopilots have the potential to prevent aircraft crashes. However, before they will be applied, there must be guarantees that the resulting controllers satisfy certain performance requirements, and these guarantees have – at least for continuous reinforcement learning (RL) controllers – not been provided. In fact, guaranteeing convergence of continuous reinforcement learning (RL) algorithms has long been an open problem. It has been accomplished for a few special (often linear) cases. Also convergence proofs to locally optimal policies have been established. But attempts to design an algorithm with proven convergence to the globally optimal policy for a general RL problem have been met with little success.**

This article examines the issues behind guaranteeing convergence of an RL algorithm to the optimal policy. It then continues by presenting Interval Q-learning: a novel continuous RL algorithm with guaranteed convergence to the optimal policy for deterministic model-free RL problems with continuous value functions. Next to a convergence proof, also bounds on the speed at which this algorithm converges are given.

This algorithm is then applied to a practical application. This experiment first of all shows that, for RL problems with a large number of state/action parameters, large amounts of runtime and memory space are required. However, the experiment also shows that the algorithm indeed works as the theory predicts, thus confirming that convergence to the optimal policy is guaranteed. Finally, a look is given to how the algorithm can be used to improve aircraft safety.

***Index Terms*—Reinforcement Learning, Interval Analysis, Self-learning controllers, Proven convergence, Interval Q-learning**

I. Introduction

A. Motivation

IN AVIATION it regularly occurs that an airplane encounters some unexpected phenomenon. For example, an engine may fall off as occurred during the Bijlmer crash (El Al Flight 1862) in October 1992, see [1], the aircraft might fly too slowly as was the case for the Buffalo crash (Colgan Air Flight 3407) in February 2009, see [2], or something as simple as a pitot tube icing may occur, as happened with Air France Flight 447, see [3]. If something like this happens, the autopilot realizes it has not been designed for the current situation and gives the majority of the control of the aircraft back to the pilots. However, often the pilots have an insufficient situational awareness, and without the aid of the autopilot this regularly results in a crash.

It is likely that an autopilot which can adapt to unexpected events would have been able to prevent at least some of these crashes. However, when using such an adaptive autopilot, it is important to be sure that the autopilot will not make things worse. Providing such performance guarantees has been the subject of a lot of research, but this research has so far been met with little success.

This is especially so in the field of the (self-learning) Reinforcement Learning (RL) controllers, as explained in [4] or [5]. In RL applications, an agent interacts with its environment. At every time iteration, the agent has a state and chooses an action. The environment subsequently puts the agent in a new state and gives it a reward. The agent then adjusts its behavior (its policy), with as ultimate goal to maximize the (weighted) sum of all future rewards. It does this by keeping track of a value function which estimates the expected sum of all future rewards.

When the states and actions are discrete – that is, there is a finite number of possible states and actions – then a variety of RL algorithms can be applied. In model-based RL, in which a model of the environment is known, value iteration and policy iteration are popular methods. In model-free RL, in which no model of the environment is available, commonly used methods include Q-learning and SARSA. As long as all possible state-action combinations are continually visited, all these methods can be proven to converge to the optimal policy. (See, for example, [4].)

However, for airplanes the states and actions are often continuous. Expanding the above mentioned algorithms to the continuous realm is not straightforward. Some kind of function approximation needs to be implemented. Very different attempts at doing so have been made in [6]–[11]. Sometimes these attempts have resulted in controllers with satisfactory performance. However, in none of these cases has convergence to the optimal policy been proven.

B. Issues in reinforcement learning

The issues in continuous RL have been investigated in [12]–[14]. The main problem is that in continuous RL there are infinitely many possible different states. To properly apply any kind of RL algorithm, the value function thus needs to be approximated by some kind of function approximator. Furthermore, when updating the value function for a certain state s, also the values of nearby states s' need to be updated. This is the cause of various problems.

First of all, *divergence* of the value function may occur. When the value of nearby states s' is not properly updated, then the error in the value function may increase over time. Secondly, in some learning algorithms, *no convergence* may occur. In this case, the value function continues to vary over

Ir. H.J. Bijl is a PhD student at the Delft University of Technology, Delft, The Netherlands (e-mail: h.j.bijl@tudelft.nl).

Manuscript received January 15, 2013.

time. Finally, *wrong convergence* may take place: while the value function has converged, the resulting policy is not optimal. This can have several causes. The function approximator of the value function may have gotten stuck in a local optimum instead of a global optimum. Or alternatively, it may have converged to the global optimum, but through its design and constraints the function approximator was simply incapable of sufficiently approximating the value function.

Literature has also shown many examples of continuous RL controllers with convergence proofs. In [15] a continuous RL controller has been applied to linear systems. For this specific case, convergence has been proven. [16] describes an algorithm whose parameters converge to a region, but not necessarily to a fixed point. (And in fact, examples have been shown in which the controller parameters do not converge, but continue to vary within this region.) The articles [17]–[20] focus on linear function approximators. For these function approximators, several convergence proofs are derived. However, linear function approximators are evidently not capable of accurately approximating any function. There are therefore no guarantees that an accurate approximation of any value function is found, nor can it be certain that the optimal policy is derived.

The only article which the author managed to find which had a convergence proof of a continuous RL algorithm with nonlinear function approximators was [21]. This article has shown for the first time that a version of policy iteration with arbitrary differentiable function approximation is convergent to a locally optimal policy. Though this is a powerful result, there are two strong downsides. First of all, the algorithm is not practically applicable. This is because the algorithm consists of two loops and the inner loop (the policy evaluation) requires an infinite amount of iterations before the outer loop (the policy improvement) can even do a single iteration. Secondly, the function approximators (through their gradient descent methods) may have proven convergence to a local optimum, but they do not have proven convergence to a global optimum. Hence, it is still not certain whether the value function is approximated with sufficient accuracy to obtain the optimal policy.

Concluding, so far no continuous RL algorithm has been devised with proven convergence to the globally optimal policy. This article will explain why this has not been achieved so far. It will continue by introducing the Interval Q-learning algorithm – a novel continuous RL algorithm with guaranteed convergence to the globally optimal policy. It then presents a practically feasible adaptation of this algorithm.

C. Article overview

This article is set up as follows. Section II discusses why previous attempts in reinforcement learning to prove convergence to the (globally) optimal policy have met with little success, and what would be required to prove such convergence. Section III introduces the theoretical framework of the Interval Q-learning algorithm and proves that it converges to the optimal policy. It then also expands the algorithm into a practically feasible version. In Section IV, this algorithm is then applied to various practical problems, giving an impression of its performance. Section V discusses how Interval Q-learning can be used to improve aircraft safety. Finally, section VI closes off with conclusions and a discussion of the algorithm.

II. Requirements on a continuous RL algorithm

Consider any continuous RL problem. To solve it, an accurate prediction of the value function for every state s needs to be obtained. The problem here is that there is an uncountable amount of different states and actions. In the case that no information whatsoever can be derived on an unvisited state s or an untried action a, it is not possible for any algorithm to have guaranteed convergence to the optimal value function.

To solve this, it is common to use a function approximator which aims to approximate the optimal value function. Whenever a state s is adjusted, this approximation is updated not only for the state s itself, but also for nearby states s'. Literature has shown that this often results in a satisfactory performance. But the main question here is: can convergence to the optimal value function actually be assured?

As was mentioned in the introduction, many RL algorithms from literature do not have guaranteed convergence. Given the data from this paragraph, the reason now seems obvious. If it is not possible (upon visiting state s and trying action a) to derive data for nearby states s' and/or nearby actions a', then convergence to the (globally) optimal value function cannot be guaranteed whatsoever. The flip side of this argument is that, should convergence to the optimal value function be guaranteed, it needs to be possible in some way to derive data about an unvisited state s'.

The way in which that is done in this article is through the *RL value slope assumption*. This assumption states that 'All derivatives (slopes) of the optimal value function Q^* have known bounds.' That is, for every state parameter s_i and action parameter a_i, the absolute values of the derivatives $\partial Q^*/\partial s_i$ and $\partial Q^*/\partial a_j$ have known bounds b_{s_i} and b_{a_j}, respectively. (Note that, if a value function V is used that only depends on the state, then of course only derivatives with respect to the state are used. However, this article focuses on an adaptation of the Q-learning algorithm and hence uses Q for the value function.) It is through this assumption that guaranteed convergence to the optimal value function can be obtained.

How exactly the above bounds b_{s_i} and b_{a_j} are established is a very problem-specific question. It is therefore not further discussed here. Instead, it is simply assumed that such bounds are known. However, as will be mentioned in section VI, the Interval Q-learning algorithm will be able to detect when the assumed bounds are incorrect.

III. The Interval Q-learning algorithm

A. The discrete Interval Q-learning algorithm

In this section a new algorithm called the Interval Q-learning algorithm is presented. It is a combination of the Q-learning

algorithm, as presented in [4], with Interval Analysis (IA), described in [22].

The main idea is that for every state s and action a, the optimal value Q^* is not approximated by a number Q anymore. Instead, it is enclosed by an interval Q. This interval is initialized and updated such that the actual optimal value Q^* always falls within this interval. The lower bound of this interval Q is denoted by $\underline{Q}$ while the upper bound is written as $\overline{Q}$.

To see how this algorithm works, first the discrete case is examined. Suppose that, for every state s and action a, bounds $\underline{Q}(s,a)$ and $\overline{Q}(s,a)$ are known. Now the RL agent is put into a certain state s_k from which it chooses an action a_k. The environment subsequently gives it a reward r_{k+1} and puts it in the state s_{k+1}. The bounds on the value function are then updated through

$$\underline{Q}(s_k,a_k) \leftarrow \underline{Q}(s_k,a_k) + \alpha\left(r_{k+1} + \gamma \max_a \underline{Q}(s_{k+1},a) - \underline{Q}(s_k,a_k)\right), \quad (1)$$

$$\overline{Q}(s_k,a_k) \leftarrow \overline{Q}(s_k,a_k) + \alpha\left(r_{k+1} + \gamma \max_a \overline{Q}(s_{k+1},a) - \overline{Q}(s_k,a_k)\right). \quad (2)$$

Here, γ is the *RL discount rate*, defined in the RL problem, and $\alpha \in (0,1]$ is the *learning rate*. In Interval Q-learning, as applied in this article, α is set to 1.

If the interval $Q(s,a)$ contains the optimal value $Q^*(s,a)$ for all (s,a) prior to the update, then the above update laws guarantee that the same will hold after the update. Furthermore, if the width of an interval is defined as $w(Q) = \overline{Q} - \underline{Q}$, then it follows that

$$w(Q(s_k,a_k)) \leq \gamma \max_a w(Q(s_{k+1},a)). \quad (3)$$

Hence, if s_k and a_k are chosen such that $Q(s_k,a_k)$ is maximized, then the width of the interval during an update is guaranteed to decrease by at least a factor γ.

This strategy of selecting the state and the action with maximum interval width (i.e. maximum uncertainty) is called the *Max-width exploration strategy*. It ensures that the widths of all intervals converge to zero and hence that the value function bounds converge to the optimal value function. Also, through relation (3), bounds can be derived on how fast this algorithm converges. Because of its usefulness, it is assumed in the remainder of this article that the Max-width exploration strategy is applied to select trial states and actions.

B. The continuous Interval Q-learning algorithm

The discrete Interval Q-learning can be extended to the continuous domain by using the RL value slope assumption. Again suppose that an agent in state s_k chooses an action a_k. For brevity of notation, merge the vectors s and a into one vector x. Through relations (1) and (2) the interval $Q(s_k,a_k)$ (or $Q(x_k)$) can be updated. At the same time, thanks to the RL value slope assumption, intervals for nearby states and action x' can be updated through

$$\underline{Q}(x') = \underline{Q}(x_k) - \sum_{i=1}^{D} b_i \left|x'_i - (x_k)_i\right|, \quad (4)$$

$$\overline{Q}(x') = \overline{Q}(x_k) + \sum_{i=1}^{D} b_i \left|x'_i - (x_k)_i\right|, \quad (5)$$

where D is the dimension of x. That is, it is the number of state and action parameters together. D is also known as the *dimension* of the problem. (Also note that the bound vector b is the concatenation of the bound vectors b_s and b_a.)

Applying relation (4) results in adding (for one-dimensional problems) a triangle or (for multi-dimensional problems) a pyramid to the landscape of $\underline{Q}$. Similarly, relation (5) adds an inverted pyramid to $\overline{Q}$. This idea can be seen in Figure 1. The pyramids which are added with every update are called *update pyramids*.

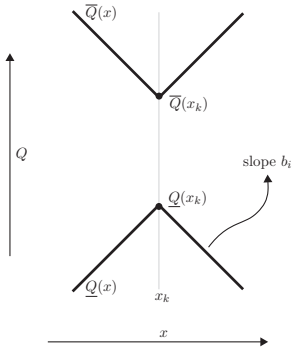

Fig. 1. Visualization of a single one-dimensional update pyramid resulting from an experiment at state/action combination x_k. Update pyramids always have the slope b_i. Multiple updates will result in a 'landscape' of such pyramids. Multi-dimensional problems of course also have multi-dimensional pyramids.

C. Guaranteed convergence of the algorithm

To prove convergence of the continuous Interval Q-learning algorithm, the *volume* between the lower and the upper bound should be considered. This volume can be found through

$$V = \int_X w(Q(x))\,dx, \quad (6)$$

where the integration is performed over all possible values of x. (Note that, though the states and actions are continuous, it is assumed that every parameter x_i takes its values from a finite range R_i.)

Related to the volume is the *average width* of the Q-function

$$w_{av} = \frac{V}{A} = \frac{\int_X w(Q(x))\,dx}{\prod_{i=1}^{D} w(R_i)}. \quad (7)$$

Here, A is the (D-dimensional) area spanned by all ranges R_i.

Now consider an update. At every update, an additional update pyramid causes the volume (and equivalently the average width) to decrease. Through applying relation (3), this

reduction in average width can be bounded. In fact, it always holds that

$$\Delta w_{av} \leq -\frac{2}{D+1}\left(\frac{1-\gamma}{2}\right)^{D+1}\frac{w_{max}^{D+1}}{\prod_{i=1}^{D} b_i w(R_i)}. \quad (8)$$

where w_{max} is the maximum width of the Q-function. (A derivation of this bound can be found in [23].)

Through the above bound, also a bound can be given on how fast w_{av} decreases over time. First define the *reference width* as

$$w_r = \sqrt[D]{\frac{D+1}{2}\left(\frac{2}{1-\gamma}\right)^{D+1}\prod_{i=1}^{D} b_i w(R_i)}. \quad (9)$$

This reference width is a problem-specific constant. The average width after k iterations is now bounded by

$$(w_{av})_k \leq \frac{2w_r}{\sqrt[D]{Dk + \left(\frac{2w_r}{w_0}\right)^D}}, \quad (10)$$

where w_0 is the initial (average) width of the value function bounds. (Again, for a proof, see [23].) As $k \to \infty$, this relation proves not only that w_{av} goes to zero, but it also gives a bound on how fast it does that. This bound proves to be very useful when applying the algorithm to practical applications.

D. Making the algorithm practically feasible

Currently, the Interval Q-learning algorithm suffers from a significant problem. At every iteration, when using relations (1) and (2), the lower and the upper bounds of the interval need to be maximized. How this is done depends on the data structure used to store these bounds.

One option would be to store all update pyramids that have been generated in the past. If n denotes the number of iterations, then maximizing the lower bound can be done in order $\mathcal{O}(n)$ time, which is usually practically feasible. Maximizing the upper bound is a far more difficult problem though, and is done in $\mathcal{O}(n^D)$ time. This run-time order means that the algorithm is certainly not practically applicable whenever $D > 2$. (Possibly a more efficient algorithm for maximizing the upper bound can be set up, but it is not expected that this will solve the problem.)

Hence a different data structure needs to be used. The approach that was chosen in this article leads to the *varying-block Interval Q-learning algorithm*. The idea is to split up the state/action space into blocks. For every parameter x_i, the range R_i is split up into n_b sub-intervals. (Often $n_b = 3$ is used.) This means that the entire state/action space is split up into n_b^D blocks. Every block then keeps track of linear bounds of the optimal value function.

Now consider the process of updating these value function bounds. During every iteration, an update pyramid is generated, according to relations (4) and (5). The bounds of every block are then narrowed as much as possible towards this update pyramid. (Narrowing here means adjusting the linear bounds such that the volume between the bounds decreases as much as possible.) The exact process of narrowing for a one-dimensional problem is shown in Figure 2.

Having a fixed number of blocks will not allow the algorithm to converge to the optimal value function though. In fact, after some time there will be an update which does not allow any of the blocks to narrow their bounds. In this case, the block in which this update occurred needs to be split up into another n_b^D sub-blocks. In this way, a tree structure of blocks is created.

It must be noted that, in this new varying-block algorithm, update pyramids aren't always fully used. (This can also be seen in Figure 2.) This means that the bound (10) doesn't necessarily hold anymore. However, through the block splitting strategy, it can be shown that always at least a fixed part of the update pyramid is used. Hence, if the right side of relation (8) is multiplied by an appropriate constant, then it still holds. Subsequently, convergence to the optimal value function is still guaranteed for the varying-block algorithm. In fact, when there is a sufficient amount of sub-blocks, then the bounds can approximate any function that satisfies the RL value slope assumption.

Next to this, the resulting tree structure of blocks also provides various other benefits. If the data structure is set up well, the algorithm is able to maximize the upper and lower bounds in a very efficient way. This maximization algorithm runs in the worst case in order $\mathcal{O}(n)$ time, but in practice it often runs with a run-time of order $\mathcal{O}(\log(n))$.

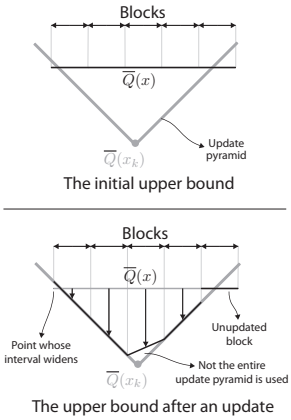

Fig. 2. Visualization of how to narrow blocks towards an update pyramid. This is always done so as to maximize the reduction of the volume between the bounds. This may sometimes have interesting side-effects.

IV. Experiment results

The varying-block Interval Q-learning algorithm has been implemented and run on the problem of controlling a simple cart. The goal of the controller is to keep the cart position z near zero. The reward function is therefore $r_k = -|z_k|$. The input is the force F applied to the cart. The equation of motion is

$$\frac{d^2 z}{dt^2} = F. \quad (11)$$

This problem has two state parameters (z and dz/dt) and one input (action) parameter F. It is thus a three-dimensional problem.

The varying-block continuous Interval Q-learning algorithm has been run for 10^4 iterations. On a normal home-computer,

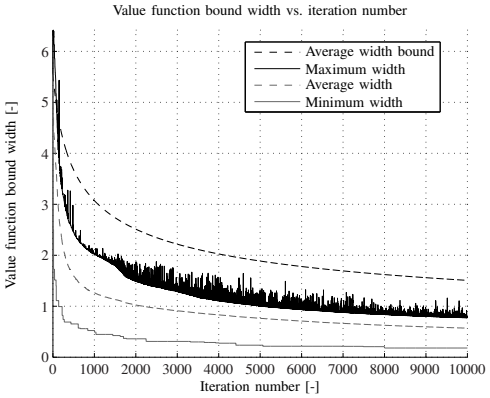

Fig. 3. Widths of the value function bounds during training of the simple cart problem. The bounded average width is calculated through relation (10).

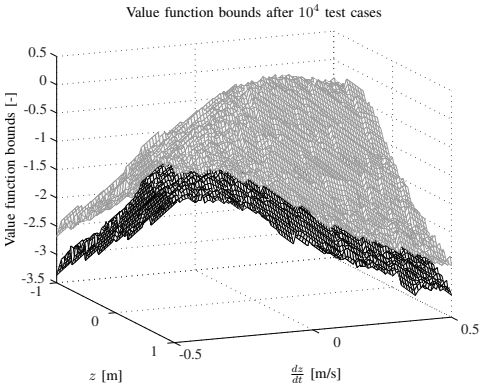

Fig. 4. Plot of the value function bounds for the simple cart problem after 10^4 iterations. Q is plotted versus z and dz/dt with $F = 0$. The lower surface indicates $\underline{Q}$ whereas the upper surface indicates $\overline{Q}$.

this took several minutes. (The Java programming language has been used.) Throughout these iterations, the average width of the value function bounds has decreased. This can be seen in Figure 3. This figure shows the way in which the average width decreased. It also shows that the average width was indeed well below its bound.

The value function bounds resulting from all 10^4 iterations are shown in Figure 4. Though there was still some distance between the lower and the upper bound, it is quite easy to visualize what the actual optimal value function would have looked like.

Since a decent value function had been established, it can be expected that the resulting controller also had a reasonable performance. And this indeed turned out to be the case. From every initial state, the RL controller was able to bring the cart near the zero position and keep it close to it. The controller wasn't able to bring the cart exactly on the zero position, but further training would enable it do so as well.

Concluding, this experiment supports the theory and shows that the algorithm indeed works.

V. Application to aircraft

The question remains how the algorithm can be applied to improve aircraft safety. It must be noted that this algorithm cannot be applied to immediately derive a controller for any unexpected event. The algorithm would be too slow for that. And in addition, the algorithm also requires the ability to experiment with all possible actions. In case of an emergency, experimenting with all possible actions is simply not possible.

Instead, the strength of the Interval Q-learning algorithm lies in its ability to derive an optimal policy by itself. Currently, designing an autopilot includes the manual tuning of various parameters. It requires time and effort from a human control engineer. With the Interval Q-learning algorithm, that will no longer be necessary. And this is the fact that enables the construction of an adaptable controller.

The first step in the design of an adaptable controller is to define dozens (if not more) of possible failure scenarios. For each of these failure scenarios, an appropriate aircraft model is made. Given the fact that defining a failure scenario is often a matter of adjusting aircraft parameters, this shouldn't be an unmanageable amount of work. The second step is then to let the Interval Q-learning algorithm come up with a suitable controller for each of these failure scenarios. This will take lots of computations, but through parallel processing this can be done within a few months – a lot less time than the average development time of an aircraft. All controllers are subsequently stored in a big onboard database.

The actual operation of the autopilot is now displayed in figure 5. During a flight, the autopilot continuously identifies the aircraft model. (Several model identification schemes for this are already available in the literature.) At every point in time, the autopilot then compares this aircraft model with all the models it has stored in its database and chooses the one that's closest. Subsequently, it uses the autopilot corresponding to that model. For most situations, this will mean that the nominal controller will be used. But when some kind of failure occurs, the autopilot will automatically switch to the corresponding controller, enabling the pilots to continue to optimally control the aircraft.

An actual implementation of this control scheme has not been made. Its effectiveness has therefore not been proven. Testing the set-up of figure 5, including its effectiveness against failure scenarios that have not been appropriately

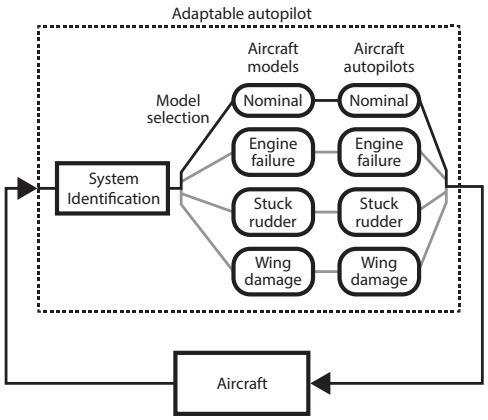

Fig. 5. Overview of the autopilot during its operation. The autopilot continuously identifies the aircraft model and compares it with models from its database. It selects the closest model and applies the autopilot created for that model.

modelled, is hence left as a suggestion for future research.

VI. Conclusions and discussion

This article has presented a novel continuous reinforcement learning algorithm with guaranteed convergence to the optimal policy for deterministic model-free RL problems with continuous value functions. Next to this, bounds are also available on how quickly the algorithm converges. These bounds have been established by applying the RL value slope assumption. Application of the algorithm to actual experiments has shown that these bounds hold and that the algorithm works as predicted by the theory.

Now one may ask: what would happen if the RL value slope assumption does not hold? That is, if for some state/action combination x, the bound $\partial Q^*/\partial x_i \leq b_i$ is violated? Or what if the initial interval Q does not contain the optimal value Q^* for some point x? In this case, it can be shown that, at some point during the algorithm execution, the upper bound $\overline{Q}$ will drop below the lower bound $\underline{Q}$. It's easy to implement a check for this in the algorithm. Hence, the algorithm can detect whether the assumptions hold, and if they do not, a warning is given. This warning adds to the integrity of the algorithm.

Though the algorithm as presented in this article is working well, there are still various ways in which it can be improved. First of all, the bounds b_i can be looked at more closely. It follows from relation (10) that a big value of b_i slows down the algorithm. The algorithm can be expanded such that b_i is allowed to vary per region in the state/action space. This would speed up algorithm convergence. Secondly, different data structures to keep track of the value function bounds Q can be examined. It is expected that there may be data structures which make more efficient use of both run-time and memory space. Especially for problems with a high dimension D, this will be beneficial. Thirdly, the strategy of when and how to split up blocks can be improved. By doing this in a smart and efficient way, again the necessary run-time and memory can be reduced. And fourthly, different exploration strategies can be investigated. Though the max-width exploration strategy is very effective, it does involve maximizing $w(Q)$ which is computationally intensive. Perhaps an additional improvement can be made here as well.

It must also be noted that this algorithm only works for deterministic systems. Whether it can be expanded for stochastic systems as well would be a very interesting yet also very challenging subject, which is left as a suggestion for a follow-up project. And finally, another suggestion for a follow-up project is to apply the Interval Q-learning algorithm to actual airplane models and their potential failure scenarios (according to figure 5) to see how well Interval Q-learning is suitable to increase aircraft safety.

Acknowledgment

The author would like to thank dr.ir. E. van Kampen, dr. Q.P. Chu and prof.dr.ir. J.A. Mulder for support and supervision during the project that led to this article.

References

[1] Raad voor de Luchtvaart (Netherlands Aviation Safety Board), "Aircraft accident report 92-11 - el al flight 1862," Ministerie van Verkeer en Waterstaat, Tech. Rep., 1994.

[2] Aviation Safety Network. (2009) ASN Aircraft accident de Havilland Canada DHC-8-402 Q400 N200WQ Buffalo Niagara International Airport, NY (BUF). [Online]. Available: http://aviation-safety.net/database/record.php?id=20090212-0

[3] BEA (Bureau d'Enquêtes et d'Analyses pour la sécuritéde l'aviation civile). (2011, July) Third interim report on flight af 447. BEA. [Online]. Available: http://www.bea.aero/docspa/2009/f-cp090601e3.en/pdf/f-cp090601e3.en.pdf

[4] R. S. Sutton and A. H. Barto, *Reinforcement Learning*. MIT Press, 1998.

[5] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[6] E. van Kampen, Q. P. Chu, and J. A. Mulder, "Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, Colorado*, no. AIAA-2006-6429. AIAA, Aug. 2006.

[7] J. Beitelspacher, J. Fager, G. Henriques, and A. McGovern, "Policy gradient vs. value function approximation: A reinforcement learning shootout," School of Computer Science, University of Oklahoma, Tech. Rep., 2006.

[8] M. Irodova and R. H. Sloan, "Reinforcement learning and function approximation," *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2005 - Recent Advances in Artifical Intelligence*, pp. 455–460, 2005.

[9] M. G. Lagoudakis, R. Parr, and L. Bartlett, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[10] D. Prokhorov and D. Wunsch, "Adaptive critic designs," *IEEE Transactions on Neural Networks*, vol. 8, pp. 997–1007, 1997.

[11] Y. Zheng, S. Luo, and Z. Lv, "Control double inverted pendulum by reinforcement learning with double cmac network," in *Proceedings of the 18th International Conference on Pattern Recognition - Volume 04*, ser. ICPR '06, 2006, pp. 639–642.

[12] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," in *Advances in Neural Information Processing Systems 7*, 1995, pp. 369–376.

[13] A. da Motta Salles Barreto and C. W. Anderson, "Restricted gradient-descent algorithm for value-function approximation in reinforcement learning," *Artificial Intelligence*, vol. 172, no. 4-5, pp. 454–482, 2008.

[14] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proceedings of the Fourth Connectionist Models Summer School*, 1993.

[15] S. J. Bradtke, "Reinforcement learning applied to linear quadratic regulation," in *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, 1993, pp. 295–302.

[16] G. J. Gordon, "Reinforcement learning with function approximation converges to a region," in *Advances in Neural Information Processing Systems*, 2001, pp. 1040–1046.

[17] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor critic algorithms," *Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

[18] F. S. Melo, S. P. Meyn, and M. I. Ribeiro, "An analysis of reinforcement learning with function approximation," in *Proceedings of the 25th international conference on Machine learning*, ser. ICML '08, 2008, pp. 664–671.

[19] A. Merke and R. Schoknecht, "Convergence of synchronous reinforcement learning with linear function approximation," in *Proceedings of the twenty-first international conference on Machine learning*, ser. ICML '04, 2004, pp. 75–80.

[20] J. N. Tsitsiklis and B. van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[21] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," AT&T Labs, Tech. Rep., 1999.

[22] R. E. Moore, R. Baker Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, 3rd ed. SIAM - Society for Industrial and Applied Mathematics, Philadelphia, 2009.

[23] H. J. Bijl, "Guaranteed globally optimal continuous reinforcement learning," Master's thesis, Delft University of Technology, 2012. [Online]. Available: http://repository.tudelft.nl/