Reinforcement learning and optimal adaptive control: An overview and implementation examples

Said G. Khan a,*, Guido Herrmann b, Frank L. Lewis c, Tony Pipe a, Chris Melhuish d

a Bristol Robotics Laboratory, University of the West of England, Bristol, UK
d Automation & Robotics Research Institute, University of Texas at Arlington, United States
b Bristol Robotics Laboratory and Department of Mechanical Engineering, University of Bristol, Bristol, UK
c Bristol Robotics Laboratory, University of the West of England, Bristol, UK

A R T I C L E   I N F O

Article info
Received 21 June 2011
Accepted 8 January 2012

Abstract

This paper provides an overview of the reinforcement learning and optimal adaptive control literature and its application to robotics. Reinforcement learning is bridging the gap between traditional optimal control, adaptive control and bio-inspired learning techniques borrowed from animals. This work is highlighting some of the key techniques presented by well-known researchers from the combined areas of reinforcement learning and optimal control theory. At the end, an example of an implementation of a novel model-free Q-learning based discrete optimal adaptive controller for a humanoid robot arm is presented. The controller uses a novel adaptive dynamic programming (ADP) reinforcement learning (RL) approach to develop an optimal policy on-line. The RL joint space tracking controller was implemented for two links (shoulder flexion and elbow flexion joints) of the arm of the humanoid Bristol-Elumotion-Robotic-Torso II (BERT II) torso. The constrained case (joint limits) of the RL scheme was tested for a single link (elbow flexion) of the BERT II arm by modifying the cost function to deal with the extra nonlinearity due to the joint constraints.

1. Introduction

Reinforcement learning controllers are bio-inspired and are based on the idea of learning from experience coupled with the principle of reward and punishment for survival and growth, borrowed from living things (human and animal) (Lewis & Vrabie, 2009). The agent (controller) is rewarded (positive reinforcement) or punished (negative reinforcement) for an action (evaluated by a reward function and cost function). This is a heuristic process where an agent tries to maximise its future rewards; in a control engineering context, the maximisation of reward is equivalent to the minimisation of a control cost. In this way, the agent tries to develop an optimal policy. Heuristic methods usually derived in the computer science domain have now found theoretical validation by control scientists, allowing for (model-free) learning, providing (model-free) solutions to the optimal control problems via ADP.

Reinforcement learning research is stretched over four decades. The origin of reinforcement learning (RL) is well rooted in computer science, though similar methods such as adaptive dynamic programming (ADP) and neurodynamic programming (NDP) were developed in parallel by researchers such as Werbos, Bertsekas and many others from the field of optimal control. In the late eighties and nineties these different schools seemed to be coming together, e.g., Sutton, Barto, and Williams (1992), Barto, Sutton, and Werbos (1989), Werbos (1990), Sutton and Barto (1998). Different methods of reinforcement learning share the same goal of learning optimality over time. According to Williams (2009), modern reinforcement learning is a blend of temporal difference methods from artificial intelligence, optimal control and learning theories from animal studies. Recent work of Werbos (2009, 2008, 2007, 2004) is pushing further the boundaries and taking the ideas of RL and ADP to ‘understand and replicate’ the functionality of the brain.

Reinforcement learning covers a large number of aspects and this paper does not present a detailed literature survey on reinforcement learning; it only highlights some of the key research in the field in particular relation to control. However, there is no doubt that there is significant scope of RL and its applications in other fields such as computer science, artificial intelligence and many other fields. Although, some of the main RL methods rooted in computer science and artificial intelligence are listed, expanding the discussion is beyond the scope of this paper. A few of the literature surveys are listed here, which will give more information for the interested reader. For instance, a detailed literature survey of reinforcement learning research has been authored by Kael, Little, and Moore (1997). Their work is mainly focussed on the RL schemes in computer science. Moreover, Sutton (1999) and also...
Thrun et al. (2000) presented overviews of RL. At a later point, Bertsekas (2005a), Bertsekas (2005b) provided a detailed literature survey of dynamic programming and explained various RL methods and techniques in great detail. In further work of Bertsekas (2006), neuro-dynamic programming (NDP), another term used for reinforcement learning/ADP was discussed (see also book by Bertsekas & Tsitsiklis (1996)). One of the important aspects of NDP/ADP is the application of neural networks (NN) to the dynamic programming (DP) problem, for approximation of the value function. Kappen (2008) has discussed the use of stochastic techniques and path-integral methods for reinforcement learning in detail. A short literature survey on ADP can be found in the work by Lin, Hui, Hua-yong, and Lin-cheng (2009). A recent book by Busoniu, Babuska, Schutter, and Ernst (2010a) provides a good background study as well as implementation examples for reinforcement learning and dynamic programming schemes based on function approximation techniques. An excellent survey on multi-agent RL is presented by Busoniu, Babuska, and De Schutter (2008), while it is here not the target to discuss multi-agent RL schemes (see for instance the work by Busoniu, Babuska, & De Schutter (2006), Shoham, Powers, & Grenager (2003)). The focus in this paper is on RL schemes for a single agent interacting with its environment in the optimal control and robotics context.

Recently, reinforcement learning approaches have been properly mathematically formalised in a control related context, e.g. Al-Tamimi, Lewis, and Abu-Khalaf (2007, 2008), Vrabie and Lewis (2009), Vrabie, Pastravanu, Abu-Khalaf, and Lewis (2009), Stigu and Lewis (2010). The challenge is to convert these ideas into practically feasible approaches. The aim of this work is to give an overview of the available RL techniques and highlight those RL techniques which are directed towards optimal adaptive control, especially in robotics. This is complemented by a first practical implementation example for the novel theoretically validated methods by Al-Tamimi et al. (2007). In general, the use of reinforcement learning techniques in optimal control has widened the horizon of the field and has overcome some of the limitations, e.g. the need for a full dynamic model in most of the traditional optimal control methods. At the same time, it brings optimality into the field of adaptive control.

Thus, the paper is structured as follows. Principles of reinforcement learning and the classification of reinforcement learning are discussed in Section 2. Temporal difference methods, the basis of adaptive optimal control approaches, are discussed in Section 3, followed by a detailed overview of adaptive optimal control approaches in Section 4. Applications of RL in control are surveyed in Section 5, while a specific example of an RL scheme implementation (both in simulation and a real robot experiment) for a humanoid robot arm, based on the work of Lewis’s group (Al-Tamimi et al., 2007; Stigu & Lewis, 2010) will be discussed in Sections 6 and 7. The paper is concluded in Section 8.

2. Principles of reinforcement learning and classification

In this Section, we may revisit some of the key elements and introduce the different categories of reinforcement learning. Reinforcement learning is learning from experience. An agent interacts with its environment through an action and this action is followed by a reward (positive RL signal) or punishment (negative RL signal) as mentioned earlier (see Fig. 1); in control engineering terms, reward implies a control cost decrease, while punishment causes a control cost increase. Hence, the RL algorithm learns an optimal behaviour over time which could ensure the RL agent’s survival and growth. This, in many circumstances, is a common behaviour in animals. The RL agent interaction is characterised by the state signal, action signal and the reward signal:

- The state signal describes the state of the environment.
- Through the action signal, the RL agent influences its environment.
- The reward signal gives feedback of the positive or negative outcome of the action taken (performance of the agent) (see Busoniu et al., 2010a).

The main elements of reinforcement learning schemes are the policy, the reward function, the value function and the model of the environment (Sutton & Barto, 1998).

- The policy is defined as the behaviour of an agent at a particular time. The policy is a very important part of RL schemes and it may be represented by a lookup table or a function. The policy may be stochastic or deterministic. In some RL schemes, a complex search process is employed in computing the policy (often, the policy is computed by minimisation of the value function, e.g. the example of Section 6). The RL process will lead to an optimal policy (behaviour) over time (Sutton & Barto, 1998).
- The reward function is defined on the basis of the goal of the reinforcement learning problem and it rewards the agent for each action taken. The agent always attempts to maximise its reward (or minimise its control cost) and hence, produces an optimal policy over time. A discount factor can be employed to set the preference for immediate reward or more future oriented reward.
- The value function is the prediction of the future reward and the basis on which the agent anticipates to take an action which will generate higher rewards. Normally two types of value functions are used, i.e. the state value function, represented generally by V(s) and the action value function, represented by Q(s,a), where s represents the state and a represents action. According to Sutton and Barto (1998), when the model of the environment is known, a state value function is used, such as in the case of Dynamic Programming methods. However, if the model of the environment is not known then an action value function is preferred, e.g. in the Monte Carlo based methods and the temporal difference methods.
- As explained in Sutton and Barto (1998), a reward function evaluates the immediate reward while the value function is the prediction of the total future reward, hence, decisions on taking actions are made on the basis of the state or action value function.

Sutton and Barto (1998) have listed three main methods to solve reinforcement learning problems: dynamic programming (DP), Monte Carlo (MC) Methods and temporal difference (TD) methods. According to Sutton and Barto (1998), DP was, even in 1998, mathematically well developed, however, it needs a full model of the environment. On the other hand, MC methods are completely model free, however, they are difficult to implement online as these methods are not incremental and work in episodes, i.e. the value function is evaluated at the end of each episode rather

- The value function is the prediction of the future reward and the basis on which the agent anticipates to take an action which will generate higher rewards. Normally two types of value functions are used, i.e. the state value function, represented generally by $V(s)$ and the action value function, represented by $Q(s,a)$, where $s$ represents the state and $a$ represents action. According to Sutton and Barto (1998), when the model of the environment is known, a state value function is used, such as in the case of Dynamic Programming methods. However, if the model of the environment is not known then an action value function is preferred, e.g. in the Monte Carlo based methods and the temporal difference methods.

As explained in Sutton and Barto (1998), a reward function evaluates the immediate reward while the value function is the prediction of the total future reward, hence, decisions on taking actions are made on the basis of the state or action value function.

Sutton and Barto (1998) have listed three main methods to solve reinforcement learning problems: dynamic programming (DP), Monte Carlo (MC) Methods and temporal difference (TD) methods. According to Sutton and Barto (1998), DP was, even in 1998, mathematically well developed, however, it needs a full model of the environment. On the other hand, MC methods are completely model free, however, they are difficult to implement online as these methods are not incremental and work in episodes, i.e. the value function is evaluated at the end of each episode rather
than at each time step. The TD methods are the combination of both DP and MC methods, hence, these are model-free unlike DP and can be easily implemented online with step by step computation, in contrast to MC methods (Sutton & Barto, 1998). One of the well known approaches, allowing for such a combination of DP and MC methods is the TD(λ) method. This is, in a sense, an averaging technique of the one-step TD method and the infinite-step MC method (Sutton & Barto, 1998).

Busoniu et al. (2010a) classify RL algorithms into the following three categories based on how the optimal policy is obtained:

- **Value iteration (VI) algorithms**: VI algorithms find the optimal value function. The optimal policy is obtained from the optimal value function. A VI algorithm can be model based or model free.
- **Policy iteration (PI) algorithms**: PI algorithms obtain value functions by evaluating policies and employ these value functions for achieving new improved policies. Like VI algorithms, PI algorithms can be model based or model free. In PI algorithms, generally a stabilizing initial policy is needed and should produce a finite cost.
- **Policy search (PS) algorithms**: PS algorithms directly search for an optimal policy via optimization techniques. PS algorithms can be gradient based or gradient free.

The next section will now consider temporal difference methods, especially, those approaches which are directly linked with the RL based adaptive optimal control approaches.

### 3. Temporal difference methods

In temporal difference methods, the value function is updated at each time step, in contrast to MC methods, where one has to wait till the end of the episode. A good foundational research treatise for temporal difference methods was provided by Sutton in his work (Sutton, 1984). TD methods (in particular Q-learning and actor-critic structures) probably are amongst the most popular of all RL schemes currently. Q-learning originated in the work of Watkins and Dayan (1992), Watkins (1989). The book on reinforcement learning by Sutton and Barto (1998) classified the temporal difference methods as:

- **Q-learning** (Watkins, 1989; Watkins & Dayan, 1992), is similar to ADHDP of Werbos (1992a) as noted by Al-Tamimi et al. (2007). Q-learning is an off-policy method, which means that the optimal action value function Q(s, a) is estimated, independently of the current policy (exploration).
- **SARSA (State-Action-Reward-State-Action)**: It is similar to Q-learning, however, it is an on-policy TD method (Sutton & Barto, 1998), hence, the action value function, Q(s, a), is estimated for the current policy and the state-action pair (it takes exploration into account unlike Q-learning).
- **Actor Critic (AC) Methods** (Sutton & Barto, 1998) use TD and the policy is independent of the value function, Fig. 2. It is an on-policy TD method. It uses state values to estimate the action value function.

An optimal Q-learning step (in a control engineering sense) can be expressed as (Bradtke, 1993; Sutton & Barto, 1998):

\[
Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t)(1 - \alpha) + \alpha \left[ r_{t+1} + \gamma \min_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) \right]
\]

where 0 < |\alpha| < 1 is the learning rate and 0 < |\gamma| ≤ 1 is the discount factor. The value of the learning rate is used to decide how much previous learning is retained. The value of the discount factor dictates the preference of immediate reward \( r_{t+1} \) (smaller value of \( \gamma \)) or future reward (larger value of \( \gamma \)). The minimal (nonnegative) future cost value is represented by \( \min_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) \).

The critic evaluates the performance of the action selected on the environment, i.e. the actor is the controller carrying out control actions. The action taken by the actor (controller) is evaluated by the environment, i.e. the actor is the controller carrying out control actions. The action taken by the actor (controller) is evaluated by the critic (reward function and cost function) using a temporal difference approach.

An actor-critic scheme can be seen in Fig. 2. The value function update step can be written as (Sutton & Barto, 1998):

\[
V_{t+1}(s_t) = V_t(s_t) + \alpha \epsilon_{TD}
\]

where, \( \epsilon_{TD} \) is the TD error (see Fig. 2) given as:

\[
\epsilon_{TD} = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)
\]

The critic evaluates the performance of the action selected on the basis of the TD error.

The further development of TD methods, to efficiently solve RL problems, is still an active field of research. For instance, from the machine learning community, Geist et al. (2010a) (see also Geist,
most effective method so far to produce reinforcement learning. Adaptive actor-critic structures are the most popular and probably optimal adaptive scheme in the presence of input constraints. Optimal control work is enshrined in ADP using adaptive actor-critic structures of Werbos (1992a) and suggested four main categories of ADP:

1. Heuristic Dynamic Programming (HDP): In this method, the value function is estimated by the critic directly.
2. Dual Heuristic Dynamic Programming (DHP): In DHP, derivatives of the value function with respect to states are estimated.
3. Action Dependent Heuristic Dynamic Programming (ADHPD): This is a slightly modified form of HDP where the actor network is directly connected to the critic network.
4. Action Dependent Dual Heuristic Dynamic Programming (ADDP-HP): This is the modified DHP where the actor network is directly connected to the critic network.

These adaptive critic design (ACD) methods come under the category of RL actor-critic TD methods. However, ACDs are strongly rooted in the RL-based optimal control community.

Prokhorov et al. (1997) have provided a detailed discussion of the adaptive critic structures of Werbos (1992a) and suggested two new methods, Global Dual Heuristic Programming (GDHP) and Action Dependent Global Dual Heuristic Programming (ADGDHP). The work by Girgin and Preux (2008) discusses developments in the natural actor critic structures. Bertsekas (2007) provides an in-depth discussion of the ADP schemes. Balakrishnan, Ding, and Lewis (2008) have provided a good discussion on the stability issues for feedback controllers based on NN and the ADP technique (adaptive critics). They have discussed model-based ADP schemes as well as model-free ADP approaches.
Al-Tamimi et al. (2007) have developed an ADP based model-free Q-learning scheme for a discrete linear system and tested the approach in simulation on a linearised F-16 aircraft model. In another instance, Al-Tamimi et al. (2008) have provided a convergence proof of the HDP scheme, for solving the Hamilton–Jacobi–Bellman (HJB) equation (this is also known as Bellman’s optimality equation) for a discrete-time nonlinear system. They have used two NNs (which is the case in most adaptive critic/actor-critic structures), one for the critic and one for the actor, approximating the policy. In this scheme, an HDP algorithm is using value iteration to solve an optimal control problem for a discrete-time nonlinear system.

Vrabie et al. (2009) have proposed a scheme based on an adaptive critic structure and policy iteration, for adaptive optimal control of a continuous-time linear system. Their scheme solves the algebraic Riccati equation online without any knowledge of the internal dynamics. The scheme is a novel optimal controller for online solution of a linear quadratic regulator (LQR) problem, in contrast to the conventional LQR scheme, which solves the algebraic Riccati equation off-line, but needs full information of the dynamics of the state-feedback system. They have provided a rigorous stability proof for the suggested scheme and have shown effectiveness of the scheme by testing it in simulation, for an optimal-load-frequency controller for a power system.

For a nonlinear continuous-time system, Vrabie and Lewis (2009) have proposed a similar adaptive optimal control scheme utilising an adaptive-critic design. The scheme is using a policy iteration algorithm. The proposed scheme uses NNs to approximately solve the HJB equation for a continuous-time nonlinear system without knowing the internal dynamics of the system. A convergence proof is provided for this adaptive optimal control scheme with an initial stabilising control policy.

Recently, Bertsekas (2010a) has presented a review of policy iteration with cost function approximation methods used in ADP. Bertsekas (2010b) has investigated reasons which could adversely affect practical implementations of the approximate policy iteration methods based on finite state stochastic dynamic programming. The main focus of the work by Bertsekas (2010b) is to address the policy oscillation problem (oscillation between poor policies) in the policy iteration method (see also Bertsekas, 2009). A detailed discussion of the TD methods used in ADP can be found in Bertsekas (2011).

5. Reinforcement learning control applications in robotics

The field of reinforcement learning originally emerged as a new area in computer science. However, it is now used in a number of other areas, for example, retail inventory management (e.g. Van Roy, Bertsekas, Lee, & Tsitsiklis (1997)) intelligent databases (e.g. Rudowsky, Kulyba, Kunin, Parsons, & Raphan (2006)), electrical power systems control (e.g. Ernst, Glavic, Geurts, & Wehenkel (2005b)), flight control studies (e.g. Kampen, Chu, & Mulder (2006)) dynamic power management (e.g., Liu, Tan, & Qiu (2010)), computer aided design, UAVs (e.g. Stingu & Lewis (2010)), robotics and optimal adaptive control (e.g. Khan (2011a, 2011b)).

Here, we focus on RL as an approach to achieve optimal control in robotic systems. One of the disadvantages of traditional optimal control is that it needs full knowledge of the system dynamics. In addition to this, the design is usually carried out off-line so it cannot deal with the changing dynamics of a system during operation, e.g., service robots, which have to perform different tasks in an unstructured and dynamic environment. On the other hand, adaptive control is well known for online system identification and control. However, adaptive control is not always optimal and may not always be suitable for applications such as humanoid robots/service robots where optimality is highly desirable. Furthermore, robots to be used in a human environment, should have the capability to learn over time and produce an optimal solution in a bio-mechanical and robotics sense, while dealing with changing dynamics. Optimality, in the robotics sense, could be the minimum use of energy or minimum force exerted on the environment during physical interaction. Aspects of safety can also be incorporated into the cost function, e.g., joint or actuator limits.

In the last decade or so, the application of reinforcement learning in robotics has increased steadily. Schaal (1997) has used RL for robot learning in an inverted pole-balancing problem (learning by demonstration). Peters et al. (2003) have classified reinforcement learning techniques in humanoid robots as greedy methods, vanilla policy gradient methods, and natural gradient methods. They strongly support the natural-actor-critic structure, using a natural gradient method, to be used for the control of humanoid robots, because they converge very quickly and are more suitable for high-dimensional systems such as humanoid robots. They have presented various approaches to design of RL based control for humanoid robots. Recently, Bhatnagar, Sutton, Ghavamzadeh, and Lee (2009) have presented an extension of this work. Theodorou, Peters, and Schaal (2007), used reinforcement learning for optimal control of arm movements. Peters and Schaal (2008b) have presented Natural Actor Critic (NAC) applications to robotics. The NAC uses a natural gradient method for estimation. Buchli, Theodorou, Stulp, and Schaal (2010) proposes reinforcement learning based on the policy improvement with a path integral approach for variable impedance control schemes. The effectiveness of the proposed scheme was demonstrated in simulation only. Greater detail of actor-critic based reinforcement learning in robotics can be found in other work of this group (Atkeson & Schaal, 1997; Hoffmann, Theodorou, & Schaal, 2008; Peters & Schaal, 2008a). More recently, Theodorou, Buchli, and Schaal (2010), have tested reinforcement learning based on policy improvement with path integral (Kappen, 2005) and tested this approach on a robot dog.

Some other examples of RL applications in robotics are as follows. Digne & Jun (1996) has presented a nested Q-learning method for a robot interacting with its environment. Kuan et al. (1998) have proposed a reinforcement learning mechanism in combination with a robust sliding mode impedance controller for compliance tasks and tested this approach in simulation. A reinforcement learning mechanism is used in their work to deal with the variation in the different compliance tasks. Bucak and Zohdy (2001), Bucak and Zohdy (1999) have presented a reinforcement learning control scheme for one and two link robots. Gaskett (2002) has explored Q-learning for robot control. Smart and Kaelbling (2002) have used RL for a mobile robot navigation task.

Izawa, Kondo, and Ito (2002) have employed an RL actor-critic structure for optimal control of a musculoskeletal type robot arm with two joints and six muscles. They have used the proposed scheme for an optimal reaching task. Althoefer, Krellberg, Husemier, and Seneviratne (2001) have applied RL to a Fuzzy rule-based system for a robot manipulator, for reaching motion and obstacle avoidance. Shah and Gopal (2009) have presented reinforcement learning control for robot manipulators in uncertain environments. Kim, Park, Park, and Kang (2010), Kim, Kang, Park, and Kang (2008) have used the reinforcement learning approach to find suitable compliance for different situations through interaction with the environment. The effectiveness of the RL based impedance learning scheme by Kim et al. has been shown in simulation. Riedmiller, Gabel, Hafner, and Lange (2005) have employed batch reinforcement learning for robot soccer. They use multilayered perceptrons to approximate the value function, employing a batch mode framework.
The most recent work by Adam, Busoniu, and Babuska (2011) presents an experimental implementation of experience replay Q-learning and experience replay SARSA methods for robot goal keeper and inverted pendulum examples. In this type of RL scheme, the data gathered in the online learning process is kept and is fed to the RL scheme repeatedly (Adam et al., 2011). The results produced are indeed promising, though the way it is implemented might not be suitable for all practical systems as the exploration phase implies highly erratic, almost unstable behaviour, which may damage a more fragile plant system.

It should be noted that some of the RL schemes discussed above are theoretically well developed and convergence proofs have been provided. However, significant work on RL is still to be done, real-time implementations of most of these schemes still pose a great challenge. Moreover, there is a greater need to introduce proper benchmark problems (Hafner & Riedmiller, 2011), against which newly developed or improved existing RL methods could be tested. In the next section, experimental implementation examples are presented to show the effectiveness and suitability of a recently developed RL scheme by Al-Tamimi et al. (2007) and Stingu and Lewis (2010).

6. A novel Q-learning scheme applied to a robot arm

In this section, a practical implementation of a novel Q-learning based adaptive optimal controller of a humanoid BERT II arm (BERT II robot is shown in Fig. 3) is presented based on the work of Al-Tamimi et al. (2007) and Stingu and Lewis (2010), (see also, our recent work Khan, 2011a; Khan et al., 2011b). Similar techniques such as LSP (Busoniu et al., 2010a, 2010b, 2010c) and the batch mode neural fitted Q-iteration (NFQI) (Hafner & Riedmiller, 2007, 2011) have been developed in the machine learning community. To prove practically the effectiveness of the theoretical ideas of Al-Tamimi et al. (2008), examples of unconstrained and constrained cases are included here. These will provide experimental evidence that the suggested schemes by the Lewis’s group at the Automation & Robotics Research Institute (ARRI), Texas University at Arlington, are indeed approaching the optimal control solution.

One of the main motivations of using this RL based optimal adaptive control scheme is that for its implementation, no prior information about the parameters of the robot is necessary. Only the states and control signal measurements have been used. The work by Stingu and Lewis (2010) and Al-Tamimi et al. (2007) is used in this example for implementation on a humanoid robot arm, while it has been studied so far in extensive relevant simulations only. A detailed explanation of such techniques can be found in Lewis and Vrabie (2009), Abu-Khalaf and Lewis (2005) and Abu-Khalaf, Huang, and Lewis (2006). Other useful references are Vrabie et al. (2009) and Vrabie and Lewis (2009).

Stingu and Lewis (2010) (see also Lewis & Vrabie, 2009) have simulated the scheme using a quadrotor unmanned aerial vehicle (UAV). They used a radial basis function neural network (NN) based on the work of Sarangapani (2006) to model the Q-function and control policy. In our approach, we are using higher order polynomials.

The controller is designed for a discrete-time system with non-linear right hand side. The cost used is in the form of an infinite quality-function sum with nonnegative values, which are usually quadratic; they can also be higher order to express, for instance, constraints. The cost function is modelled by NNs. An estimate of the cost function is obtained in an iterative process, stability has been proven by Al-Tamimi et al. (2008) for these schemes.

6.1. Q-learning algorithm

In this section, the Q-learning algorithm is described for a tracking problem based on the work by Al-Tamimi et al. (2007), Stingu and Lewis (2010). Considering the following discrete-time system:

\[ x_{k+1} = f(x_k) + g(x_k)u_k, y_k = x_k \]  \hspace{1cm} (1)

where, \( x_k \in \mathbb{R}^n, y_k \in \mathbb{R}^m \) and \( u_k \in \mathbb{R}^m \) is the control input. We consider the infinite horizon value function:

\[ V^*(x_k, d_k) = \min_{u_k} \sum_{i=k}^{\infty} \gamma^i r(x_i, u_i, d_i) \]  \hspace{1cm} (2)

where \( 0 < \gamma < 1 \) is a discount factor, defined earlier. Where \( r(x_i, u_i, d_i) = r(x_i, d_i) + u_i^TRu_i \) and \( R \) is a positive definite matrix. The vector \( d_i \) can be interpreted as a demand so that \( r(x_i, d_i) \geq 0 \) can represent a cost for tracking, rather than for a simple regulation problem.

The goal is to find an optimal control policy, \( u^*_k \), considering feedback stabilising policies in the same way as in (Al-Tamimi et al., 2007). Using the principle of adaptive dynamic programming (ADP) and optimal control, the cost, given by (2), can be written (Al-Tamimi et al., 2007):

\[ V^*(x_k, d_k) = \min_{u_k} \left( r(x_k, u_k, d_k) + \gamma V(x_{k+1}, d_{k+1}) \right) \]

\[ = \min_{u_k} \left( r(x_k, u_k, d_k) + \gamma V^*(x_{k+1}, d_{k+1}) \right) \]  \hspace{1cm} (3)

As in the work by Stingu and Lewis (2010), the concept of the Q-function applied to an optimal control problem can be expressed as an optimal value function in the form (the Q-function associated with the control policy \( h \)):

\[ Q_h(x_k, u_k, d_k) = r(x_k, u_k, d_k) + \gamma V_h(x_{k+1}, d_{k+1}) \]  \hspace{1cm} (4)

The policy \( u_k = h(x_k, d_k) \) will have to be stabilising and should produce a finite cost \( V_h(x_k, u_k, d_k) \) (this is a practical requirement). The
weights of the NN that calculates the control policy are initialised with stabilising initial gains.

The optimal cost function can be rewritten as:
\[
Q'(x_k, u_k, d_k) = r(x_k, u_k, d_k) + \Gamma V'(x_{k-1}, d_{k-1})
\]
Bellman's optimality equation can be written in terms of \( Q' \):
\[
V'(x_k, d_k) = \min_u Q'(x_k, u_k, d_k)
\]
The optimal control policy is:
\[
h'(x_k, d_k) = \text{argmin}_u Q'(x_k, u_k, d_k)
\]
Hence, the control inputs can be calculated by solving
\[
\frac{\partial Q'}{\partial u}(x_k, u_k, d_k) = 0
\]
for \( u_k \), assuming \( Q' \) is sufficiently smooth (differentiable).

6.2. Algorithm

The cost of the control problem is modelled via a neural network \( \phi(\cdot) \) with weights \( w_i \). We have the following parameterisation:
\[
\hat{Q}(x_k, u_k, d_k, w_l) = w_l^T \phi(z_l(x_k, u_k, d_k))
\]
The function \( z_l(x_k, u_k, d_k) \) is used to simplify the definition of the NN nodes \( \phi(\cdot) \). An explanation will be given later. \( \hat{Q}(z_l, w_{l-1}) \) has to fit \( \delta(\cdot) \):
\[
\delta(z_l(z_k(x_k, u_k, d_k)), w_l) = r(x_k, d_k) + \hat{u}_k(x_k)^T R \hat{u}_k(x_k) + \hat{Q}(z_{k+1}, \hat{u}_k(x_{k+1}), d_{k+1})
\]
in a least squares sense to find \( w_{l-1} \).

Hence, the vector \( w_l \) of the NN weights, is calculated by error minimisation between the target value function (8) and (9) in a least-squares sense as given [Al-Tamimi et al., 2007]:
\[
w_{l-1} = \text{argmin}_{w_{l-1}} \left\{ \int |w_l^T \phi(z_l) - d(\phi(z_k), w_l)|^2 dx_k \right\}
\]
Solving the least squares problem, we get (see Al-Tamimi et al., 2007):
\[
w_{l,-1} = \int_0^1 \phi(z_l) \phi(z_l)^T \int_0^1 \phi(z_l) d(\phi(z_k), w_l) dx_k
\]
To improve the robustness of the algorithm, an update law
\[
w_{l\text{-app},l} = \gamma w_{l-1} + (1 - \gamma) w_{l\text{-app}}
\]
is used, where \( \gamma, 0 < \gamma < 1 \), is a forgetting factor. Hence, \( w_{l\text{-app}} \) is practically used for the implementation of the control policy.

6.3. Simplification for practicality

We assume that the neural network model allows for a quadratic cost model in the control signal
\[
\phi(z_l(x_k, u_k, d_k)) = \begin{bmatrix}
\phi_1(z_l(x_k, u_k, d_k)) \\
\phi_2(z_l(x_k, u_k, d_k))u_k \\
\hat{u}_{l1} \\
\hat{u}_{l2} \\
\vdots \\
\hat{u}_{lm}
\end{bmatrix}
\]
thus
\[
Q_l(x_k, u_k, d_k) = w_{l1}^T \phi_1(z_l(x_k, u_k, d_k)) + w_{l2}^T \phi_2(z_l(x_k, u_k, d_k))u_k + \hat{w}_{l1}^T \hat{u}_{l1} + \hat{w}_{l2}^T \hat{u}_{l2} + \cdots + \hat{w}_{lm}^T \hat{u}_{lm}
\]
where \( w_{l1} = [w_{l11}, w_{l12}, \ldots, w_{l1m}]^T \), \( w_{l2} = [w_{l21}, w_{l22}, \ldots, w_{l2m}]^T \), and \( u_k = [u_{l1}, u_{l2}, \ldots, u_{lm}]^T \).

6.4. Implementation

The learning period will create data for \( z_k(x_k, u_k, d_k) \) and \( \delta(\phi(z_k), w_k) \). Hence, at the end of each learning stage, Eq. (11) can be solved. However, this is easier if a standard recursive least squares (RLS) approach is used, where the inverse \( \left( \int \phi(z_k(x_k, u_k, d_k)) \phi(z_k(x_k, u_k, d_k))^T \right)^{-1} \) is recursively obtained for each data sample. Hence, the RLS algorithm is:
\[
W_k = d(\phi(z_k(x_k, u_k, d_k)), w_l) \]
\[
\hat{W}_k = w_{l\text{-app},l} \phi_k
\]
\[
P_k = P_{k-1} - \frac{P_{k-1} \phi_k \phi_k^T P_{k-1}}{1 + \phi_k^T P_{k-1} \phi_k}
\]
\[
K_k = P_k \phi_k
\]
\[
w_{l(k)} = w_{l(k-1)} + K_k(W_k - \hat{W}_k)
\]
This RLS algorithm is run during a single learning cycle to achieve (11) for large enough \( k \), i.e. sufficiently long learning time. A large initial value for the inverse correlation matrix, \( P \), should be selected so that the RLS algorithm converges quickly. It should be noted that the inverse correlation matrix, \( P \), is initialised at the start of each learning cycle.

In summary, the algorithm is implemented as shown in Fig. 5. Initially, stabilising controller gains are selected for the NN. Some discussion about this initial stabilising controller will be provided later on, in this paper. White noise is added to both the control inputs to meet the condition of persistency of excitation condition in the scheme for calculating \( Q \). The NN weights are calculated using the recursive least squares algorithm. However, new NN-gains are applied at the end of a learning cycle. The control input is calculated by minimising the Q-function at each time step i.e. \( \hat{\omega} = 0 \).

As mentioned previously, an update law (12) is invoked once at the end of a learning cycle, using a forgetting factor, \( \gamma \), to make the algorithm robust; then, the new weights are kept and the control policy (14) for the upcoming cycle is updated.

The process described above continues until the NN weights converge. It should be noted that in Fig. 5, the finish point represents the end of the learning process. At the end of the learning process, i.e. at the finishing point in the flow chart in Fig. 5, weights of the NN are kept and the controller continues operating with constant NN weights.
7. Practical test

In this section, simulation and experimental implementation of the RL algorithm described above is presented. Initially, implementation of an unconstrained RL controller for a two link robot arm is discussed, simulation and experimental results are presented (see Section 7.1). The constrained case of the RL control scheme, for one link of the BERT II arm is presented in Section 7.2.

7.1. RL controller 2-DOF problem without constraints

The algorithm described in the previous section for a tracking problem, has been experimentally tested on a two link arm, i.e. using shoulder flexion and elbow flexion of the BERT II arm (Khan, Herrmann, Pipe, Melhuish, & Spiers, 2010b; Khan, Herrmann, Pipe, & Melhuish, 2010a; Khan, Lenz, Herrmann, Pipe, & Melhuish, 2011c) (The BERT II arm has 7 DOF) (see Fig. 7). In our case, we are not using any information from the dynamic model of the robot, in contrast to Stingu and Lewis (2010), hence, our cost function reduces to:

\[ r(x_k, e_k, u_k) = e_k^T Q e_k + (u_k + 1/C_0 u_k)^T R (u_k) \]

where, \( e_i = [e_1, e_2, e_3, e_4]^T \) is the tracking error for the two links of the BERT II arm. The scalars \( e_1, e_2, e_3, e_4 \) are the shoulder flexion joint position error, elbow flexion joint position error, shoulder flexion joint velocity error and elbow flexion joint velocity error respectively. The matrices \( Q \in \mathbb{R}^{4 \times 4}, R \in \mathbb{R}^{2 \times 2} \) and \( S \in \mathbb{R}^{2 \times 2} \), are positive definite and diagonal. For the two link case, \( u_k \in \mathbb{R}^{2 \times 1} \). The control input is \( u_k = h(x_k, e_k) \), solving (14).

We assume the general structure of the robot dynamics for \( n \) DOF is given by:

\[ M(q) \dot{q} + V(q, \dot{q}) + G(q) = \tau \]  

(15)

where \( M \in \mathbb{R}^{n \times n} \) is the inertia matrix, a function of the \( n \) joint angles \( q \). \( V \in \mathbb{R}^{n \times 1} \) is the coriolis/centripetal vector, which also represents viscous and nonlinear damping. \( G \in \mathbb{R}^{n \times 1} \) is the gravity vector. The vector \( \tau \in \mathbb{R}^{n \times 1} \) is the input torque. Hence, we can write the robot Eq. (15) in state-space form:

\[ \begin{bmatrix} \dot{q} \\ q \end{bmatrix} = \begin{bmatrix} 0 \\ -M^{-1}(V + G) \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix} \tau \]  

(16)

or

\[ \dot{x} = A(x) + B(x)u = f(x, u) \]  

(17)

Fig. 4. NN architecture used for estimating Q-function, where \( m = 78, n = 4 \) and \( p = 4 \), for the two link robot experiment.

Fig. 5. Flow chart of the scheme.

Fig. 6. Reinforcement learning actor-critic scheme for BERT II arm (Stingu & Lewis, 2010).
Fig. 7. BRL BERT II arm.

Fig. 8. Simulation of a regulation problem for the two-link case.
where,
\[
A(x) = \begin{bmatrix} \dot{q} \\ -M^{-1}(V + G) \end{bmatrix}
\]
and
\[
B(x) = \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix}
\]
provided that \(M^{-1}\) is invertible (Lewis, Dawson, & Abdallah, 2003).

Using a fast sample-and-hold process, the robot arm can be modelled as a discrete time system (1). Hence, for the two link case: \(x\) is a 4 \(\times\) 1 state vector where \(x_1 = q_1\), is the shoulder position angle, \(x_2 = q_2\), is the elbow position angle, \(x_3 = \dot{q}_1\), is the shoulder angular velocity and \(x_4 = \dot{q}_2\), is the elbow angular velocity. The control input \(u = \tau\) has the size of 2 \(\times\) 1 for a two link robot. It should be noted that for the implementation of this RL controller, no information of the dynamic model, i.e. the inertia matrix \(M(q)\), or \(V(q, \dot{q})\) and \(G(q)\), is necessary.

7.1.1. Choice of NN-structure

The choice of structure for the NN is very important as this enables the RL-scheme to properly model system dynamics and approximate the cost function. It should be noted that we use only one NN (for critic only) in this scheme in contrast to the work by Stingu and Lewis (2010) and to the other usual adaptive critic/actor-critic schemes, not only because of computational limitations but also to keep the scheme as simple as possible.

The \(Q\)-function is approximated by the NN structure as shown in Fig. 4 i.e.
\[
\hat{Q} = w_t^T \phi(z_t(x_t, e_t, u_t))
\]
where \(w_t\) consists of the NN weights to be learnt over time. The vector \(\phi(z_t(x_t, e_t, u_t))\) resulting from the Kronecker product of \(z_t(x_t, e_t, u_t)\):
\[
z_t(x_t, e_t, u_t) = [u_{s1}, u_{s2}, e_1, \ldots, e_4, \dot{e}_1, \ldots, \dot{e}_4, \ddot{q}_1, \ddot{q}_2]^T
\]
for the two links of the BERT II robot arm, where \(e_t = [e_1, e_2, e_3, e_4]^T\) (size 4 \(\times\) 1), is the tracking error and the input vector is \(u = \tau_{s, e}\). Hence, the NN structure contains some quadratic elements and also has higher order terms of the states and the tracking errors. This enables the NN to learn the nonlinearity of the robot arm. This also eliminates the need to have a large number of RBF neurons to learn the nonlinearity in the robotic system. We use 78 neurons in our experiments, which gives us satisfactory results.

7.1.2. Controller initialization

The policy iteration based RL schemes requires an initial stabilising control policy (Al-Tamimi et al., 2007, 2008). For practical implementation, the initial controller gain values are very important: They need to keep the controller stable during the first learning period so that a new control policy can be calculated (Stingu & Lewis, 2010). Very high initial controller gains for \(h_0(x_0, e_0)\) will defeat the learning process (Stingu & Lewis, 2010). In our case, the NN gains were initialised in such a way to produce a stable initial proportional-derivative (PD) controller. In case of a model free approach, PD makes an easy choice, though some trial and error may be needed to adjust the PD gains.

7.1.3. Learning period

The learning period is the time duration after which the control policy is updated. Al-Tamimi et al. (2007) have suggested the
number of points collected should be at least more than $\ell \times (\ell + 1)/2$, where $\ell$ is the sum of the states and the control inputs. However, in a complex nonlinear system, more points would be required to solve the least square problem in Section 6.2. As described earlier, we use the RLS algorithm solving for the NN weights at each time step. However, the NN weights are updated only after each learning cycle, see Fig. 5. A shorter learning cycle is better so that it can converge to an optimal policy quickly. However, if the learning period is very small, then the dynamics of the robot arm will not be learnt properly and this may lead to instability. In the simulation results, we found that a 1.8 s length for learning is suitable in our case, while in the real-time implementation, a learning period of more than 12 s was found appropriate. Note that this roughly coincides with the control bandwidth in simulation and experiment, i.e. tracking demands of 0.5Hz in simulation and about 0.2Hz in practice are considered (see Fig. 6).

7.1.4. Results

The RL control scheme (Section 7) has been simulated using the MATLAB/Simulink SimMechanics toolbox and experimentally implemented for the two links of the BERT II humanoid robot arm. Simulation of the regulation case for the two links of the BERT II arm is depicted in Figs. 8 and 9. The RL tracking controller’s simulation results for two links the BERT II arm are shown in Fig. 10. A multi-step demand input for the shoulder flexion joint and a sine wave demand input for the elbow flexion joint have been used.

Fig. 10. Shoulder and elbow position tracking in simulation.
Results produced with the unconstrained RL controller for the two links of the real BERT II arm, for a multi-step demand input for both shoulder and elbow flexion joints are shown in Figs. 11 and 12. It is obvious from the Figs. 11 and 12, that the tracking performance improves over time. In another experiment, real-time results, for sine wave demand inputs are shown in Figs. 13 and 14. The NN weights, $w_i$, for this experiment are shown in Fig. 15; they converge after a few policy iterations. It can be seen in Fig. 15 that the weights are changing after 12 second (one learning cycle).

7.2. Reinforcement learning based controller with constraints

The beauty of the RL control scheme is the ease with which any constraints such as joint limits (Abu-Khalaf & Lewis, 2005; Abu-Khalaf et al., 2006) can be incorporated into the cost function. The cost function becomes highly nonlinear, in contrast to the usual quadratic cost. However, more neurons will be required than in the unconstrained case, to learn this higher order nonlinearity. To implement the joint limit, $C(q)$, a tangent based function is
introduced into the cost function and to the neural network. The function \( C(q) \) is defined as:

\[
C(q) = \begin{cases} 
\tan^2 \left( \frac{q}{q_L} \right), & \text{if } ||q|| < q_L \times \lambda \\
\tan^2 \left( \lambda \times \frac{q}{q_L} \right), & \text{if } ||q|| \geq q_L \times \lambda 
\end{cases}
\]  

where \( q \) is the joint position in radians, \( q_L \) is the joint limit and \( \lambda \), \( 0 < \lambda < 1 \), is a positive constant very close to 1. The logic of using this function is to increase the cost of the control if the joint is going towards its limits. Hence, the controller tries to stop the joint going towards the limits. The new cost function is:

\[
\tan^2 \left( \frac{q}{q_L} \right), \quad \text{if } ||q|| < q_L \times \lambda \\
\tan^2 \left( \lambda \times \frac{q}{q_L} \right), \quad \text{if } ||q|| \geq q_L \times \lambda
\]

\[(21)\]
The vector $x_k = [x_1, x_2]^T$ is the state vector i.e. $x_1 = q$ is the joint position in radians, $x_2 = \dot{q}$, velocity of the joint. $e_k = [e_1, e_2]^T$ is the error vector; it consists of $e_1$, position error and $e_2$, velocity error.

The rest of the operation of the RL constrained control scheme is similar to the unconstrained case described in Section 7.1. However, the controller is now capable of dealing with constraints in the form of joint limits.

7.2.2. Controller initialization

The scheme is similar to the unconstrained case, i.e. it is also a policy iteration based scheme. Hence, an initial stabilising policy is needed to start with. In the NN-structure, the weights are initialised in such a way as to produce a PD controller so that the controller can track the demand trajectory in the first learning cycle and to provide enough data to calculate the first RL based control policy.

7.2.3. Learning period

As discussed before, the learning period is very important in these schemes and it should be of suitable length to be able to capture the dynamics of the robot arm and learn the constraints on the system states in this case as well. For the one link constraints simulation, a learning period of 1.8 s is used in this work, while for experimentation, a learning period of 16 s is employed.

7.2.4. Results

The 1DOF RL with constraint (Section 7.2) has been tested for the elbow flexion joint of the BERT II arm. The elbow flexion joint is used for the one link constrained RL controller. The BERT II arm is shown in Fig. 7.

For the constrained case (joint limits), the RL controller results produced with one link (shoulder flexion joint) of the BERT II arm in simulation are shown in Figs. 16–18. Fig. 16 shows that at time, $t = 8$ s, the demand position is at 15 degrees. However, the shoulder flexion joint does not cross the pre-defined joint limit of 15 degrees. The significant increase in the control cost can be observed in Fig. 17 at time $t = 8$ s.

The constrained RL controller has been experimentally tested on the elbow flexion joint of the real BERT II arm, see Figs. 19–21. It should be noted that the zero position of the elbow flexion joint of the BERT II arm is at a bent position as shown in Fig. 7. The joint limits have been fixed at ±20 degrees. It is shown in Fig. 19, that at time $t = 52$ s, the controller is able to keep that limit of −20 degrees without significantly degrading the performance. Hence, the link is kept away from the vertical downward
position. It is evident in Fig. 20 that there is a large increase in the control cost (just after 52 s), to keep the elbow flexion joint within the ±20 degree limits.

7.3. Discussion

The use of a polynomial based neural network for estimating the quality function, enables us to learn the robot dynamics with a small number of neurons. We have shown through experimental results that the tracking performance improves over time for the two-link unconstrained case. The constrained case of the RL controller has been tested on one link of the BERT II arm. The simplicity and ease of introducing constraints into the scheme, together with the supporting simulation and experimental results, are very encouraging and elaborate the effectiveness of this scheme. The implementation of this RL control scheme is simple. However, the use of neural networks makes it computationally demanding as compared to other control schemes which do not employ NN.

Fig. 19. Elbow position for real robot experiment, the elbow joint keeps above the joint limit of –20 degrees (see Fig. 7).

Fig. 20. \( \dot{q} \) vs. cost function real robot experiment (joint limit).

Fig. 21. The NN-weights in the real robot experiment (joint limit).
Note that the lower arm is more than 2 kg while the upper arm is more than 4 kg. This also introduces a high inertia for the links of the BERT II arm. Achieving a practical response time of less than 2 s is significant with this high inertia, considering that the controller is learnt during online operation.

8. Conclusion

In this paper, we presented an overview of reinforcement learning and optimal adaptive control. ADP techniques, such as adaptive critic or actor-critic methods, are the key to achieve optimal adaptive control online. Reinforcement learning techniques are set to change the face of modern optimal control significantly. Although a lot of theoretical work has already been done, there is still a need for a better solution to the practical implementation problem. The last decade has seen many examples of RL based optimal adaptive control schemes, particularly, the popularity of these novel techniques is immense among the robotics community. This paper provides a brief overview of the RL based optimal adaptive control schemes and shares an optimism about this bio-inspired optimal adaptive control field which has a great potential to change the field of control and robotics in the near future. A recently developed, novel Q-learning scheme implementations for a humanoid robot arm (for unconstrained and constrained cases) has been presented to emphasise the practical usefulness of these techniques. Although the field is futuristic, it is opening new horizons in control and robotics, however, practical implementations of such schemes are a challenge. One of the main hurdles is the so-called ‘curse of dimensionality’ which makes things more complicated when applied to higher dimensional problems, and also entails a much greater requirement for computational power as compared to traditional schemes. The introduction of schemes which can keep dimensionality low but retain convergence and algorithm robustness is one of the great challenges in reinforcement learning based control. We have shown through experimental examples that with a slightly modified and subsequently simplified approach towards NN estimation of the cost function, satisfactory results could be produced in real-time.

Acknowledgements

The research leading to these results has received funding from the European Community’s Information and Communication Technologies Seventh Framework Programme [FP7/2007-2013] under Grant Agreement No. [215805], the CHRIIS project, from the NSF (Grant ECCS-0801330) and the ARO (Grant W911NF-05-1-0314).

References

Said Ghani Khan is a postdoctoral RA at the department of Mechanical Engineering, University of Bristol. He is a member of the Nonlinear Robotics Control Group (NRCG) at the Bristol Robotics Laboratory. He has completed his PhD at Bristol Robotics Laboratory, University of the West of England and University of Bristol in 2011 and M.Sc. Robotic with Distinction from the University of Plymouth, UK in 2006. He received his B.Sc. Mechanical Engineering, First Division with Honours, from NWFP University of Engineering Technology (UET) Peshawar, Pakistan in 2003. He was a part-time lecturer in University of the West of England, Bristol, from September 2008 to October, 2011, and a Research Associate at the GIK Institute of Engineering Sciences and Technology, Pakistan, from March 2007 to July 2008.

Guido Herrmann received the German degree “Diplom-Ingenieur der Elektrotechnik” (with highest honours) from the Technische Universität zu Berlin, Germany, and the Ph.D. degree from the University of Leicester, UK, in 1997 and 2001, respectively. From 2001 to 2003, he was a Senior Research Fellow at the Data Storage Institute in Singapore. From 2003 until 2005, he was a Research Associate, Fellow, and Lecturer at the University of Leicester. He joined the University of Bristol, Bristol, UK, as a Lecturer in March 2007, and has been, since August 2009, a Senior Lecturer. His research considers the development and application of novel, robust and nonlinear control systems. He is a Senior Member of the IEEE and a Technical Editor of the IEEE/ASME Transactions on Mechatronics and an Associate Editor of the International Journal on Social Robotics. He is leading the Nonlinear Robotics Control Group (NRCG) at the Bristol Robotics Laboratory.

F.L. Lewis. IEEE Fellow, IFAC Fellow, Fellow Inst. Measurement & Control, PE Texas, UK Chartered Engineer. Distinguished Scholar Professor and Moncrief-O’Donnell Chair at University of Texas at Arlington. Works in feedback control and intelligent systems. Author of 6 US patents, books, and several journal papers. Awards include Fulbright Research Award, NSF Research Initiation Grant, ASEE Terman Award, and International Neural Network Society Gabor Award. Selected as Engineer of the Year by Ft. Worth IEEE Section.

Professor Tony Pipe is Deputy Director of the Bristol Robotics Laboratory, which is jointly funded by both University of Bristol and the University of the West of England (UWE), Bristol. He is also Professor of Robotics and Autonomous Systems at UWE, where he has been carrying out research and teaching activities for 23 years. He has 17 years experience of carrying out research in biologically-inspired robotics, machine learning and adaptive behaviour, applied to intelligent and distributed control, and self-learning VLSI electronic hardware for safety-critical applications; adaptive neural control of complex nonlinear systems, e.g., robot manipulators, hydraulic/pneumatic actuators; rehabilitation robotics (robots that can aid or support human recovery). He has co-authored over 150 internationally reviewed book chapters, journal and conference papers on these subjects.

Professor Chris Melhuish is Director of the Bristol Robotics Laboratory, a partnership between the University of Bristol and the University of the West of England. The Bristol lab is the largest multi-disciplinary facility of its kind in the UK. He is a Fellow of the British Computer Society, a Chartered Engineer and Fellow of the British Computer Society and Institute of Engineering and Technology. His research interests include: robot-human interaction, haptic tele-presence, neuro-inspired systems, energy autonomy in robots and collective mobile robotics. In addition to his chair at the University of the West of England in Intelligent Autonomous Systems he also holds the chair in Robotics and Autonomous Systems at the University of Bristol.