IET Cyber-Physical Systems: Theory & Applications

Research Article

Adaptive embedded control of cyber-physical systems using reinforcement learning

ISSN 2398-3396
Received on 18th March 2017
Accepted on 28th June 2017
doi: 10.1049/iet-cps.2017.0048
www.ietdl.org

Hamid Mirzaei Buini[1] ✉, *Steffen Peter*[1], *Tony Givargis*[1]

[1]*Center for Embedded and Cyber-Physical Systems, University of California, Irvine, USA*

✉ *E-mail: mirzaeib@uci.edu*

Abstract: Embedded control parameters of cyber-physical systems (CPS), such as sampling rate, are typically invariant and designed with a worst case scenario in mind. In an over-engineered system, control parameters are assigned values that satisfy system-wide performance requirements at the expense of excessive energy and resource overheads. Dynamic and adaptive control parameters can reduce the overhead but are complex and require in-depth knowledge of the CPS and its operating environment – which typically is unavailable during design time. The authors investigate the application of reinforcement learning (RL) to dynamically adapt high-level system parameters, at run time, as a function of the system state. RL is an alternative approach to the classical control theory for CPSs that can learn and adapt control properties without the need of an in-depth controller model. Specifically, we show that RL can modulate sampling times to save processing power without compromising control quality. We apply a novel statistical cloud-based evaluation framework to study the validity of our approach for the cart-pole balancing control problem as well as the well-known mountain car problem. The results show an improved real-world power efficiency of up to 20% compared with an optimal system with fixed controller settings.

Nomenclature

x	cart position
θ	pole angle
l	pole length
m_c	cart mass
m_p	pole mass
T	kinematic energy
U	potential energy
k_1	cart viscous friction coefficient
k_2	pole angular viscous friction coefficient
f	applied force to the cart
t_r	processing time
h_i	ith sampling time
p_r	processing power in run mode
p_s	processing power in idle mode
N	total number of sampling times (steps) before the battery energy ends
T	total time before the battery energy ends
E_{proc}	total processing power

1 Introduction

In a cyber-physical system (CPS), most generally, a physical system is controlled by an embedded control system (ECS). The ECS is the cyber part of the CPS. The ECS contains the control programme that periodically processes sensor inputs and generates actuator outputs to achieve the stability, quality and performance goals of the CPS. The performance of the ECS is determined not only by hardware decisions, such as the applied computation platform, but also by software-defined decisions such as sampling rate and resource allocation.

Most of today's embedded control design approaches assume the ECS parameters to be fixed quantities, which are set at design time. Using classical control theory, the system parameters are set to work in the most challenging (worst case) scenario, for which the designer validates the stability of the system. Such over-engineering results in resource usage inefficiency, for example when the sampling rate designed for temporary high-bandwidth disturbances or non-linear dynamics exceeds the required value for the current system state.

In this paper, we investigate the feasibility and the effect of online adaptation of ECS parameters to improve the resource utilisation and energy consumption of the ECS and the entire CPS. Adaptive parameters have already been applied in isolated cyber systems, for instance to dynamically tune the voltage and frequency of a system [1]. However, the approaches do not consider the effect of the changes on the physical part of the CPS. Different sampling rates and computation settings influence the stability and correctness of the CPS, as well as its overall power consumption, with non-trivial trade-offs [2].

Therefore one of the main challenges of adaptive ECSs (A-ECSs) is to model and understand the effects of parameter tuning of the ECS on the physical system dynamics and control performance. Existing approaches [3, 4] rely on classical control theory and require complex application-specific models. To reduce the modelling complexity, the work in our paper relies on reinforcement learning (RL). In RL methods, the prior knowledge about the system dynamics is not required because RL can learn the optimal control policies just by experiencing the environment and observing the reward signal. Therefore, RL is a promising candidate to control time-varying and non-linear systems with uncertainties in the model or system states. RL has already been successfully demonstrated to control real-world physical systems, such as autonomous transportation [5], smart grids [6] and robotics [7], however, without considering the effects of the ECS parameters.

In our work, we investigate if the benefits of RL are applicable not only to learn properties of the control part of the system but also to adapt attributes of the ECS at run time to improve usage of system resources. Specifically, we present the A-ECS framework that utilises RL to control the sampling time depending on the system state, i.e. at each sampling time the controller determines the next sampling time. Using A-ECS, we show that processing time and consumed energy can be reduced by 20% for the classical cart-pole example, compared to an optimal implementation with fixed controller settings. At the same time, A-ECS improves the control quality in the presence of model uncertainties, compared to fixed controllers and event-triggered controllers (ETC). Our results are obtained with a novel cloud-based co-simulation framework. Theoretical and experimental results for two benchmark applications indicate the practical suitability of RL to control online parameters of ECSs as part of CPSs.

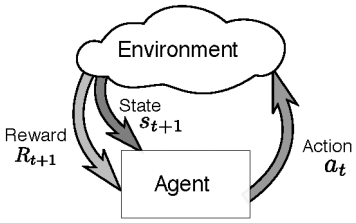

Fig. 1 *Agent–environment interaction model in RL*

This paper is structured as follows: We review related work in Section 2 and introduce RL in Section 3. In Section 4, we present our framework for the online adaptation of ECS properties. Sections 5 and 6 present experimental setups and our results, before we conclude the paper.

2 Related work

The impact of design decisions of the ECS to the overall CPS system performance has been discussed in a range of works [2, 4, 8]. For instance, the authors in [2, 8] applied holistic cyber-physical design-space exploration to show the existence of optimal design points regarding control quality and power consumption. Specifically, Buini *et al.* [2] showed that long sampling rates might increase the power consumption of the physical part of the system, while short sampling rates increase the average power consumption of the ECS. However, the works do not consider multi-mode system or time-variance yet.

Time-varying control has been discussed directly [3, 4] or in form of schedule planning [9, 10]. For instance, the optimal sampling time assignment in feedback controllers was studied in [4]. The result is that online sampling period assignment can deliver significantly better control performance than the state of the art, static period assignment. However, computing the optimal rate either requires complex online computations or large look-up tables which in turn reduces the power efficiency of the system. Sala [3] realised time-varying sampling periods and delayed actuation by time-varying observers and Kalman-filter-based state-feedback controllers. While the approach demonstrates the feasibility of variable sampling periods, the implementation requires complex application-specific control knowledge to be feasible.

The works in [9, 10] extended the idea to develop an optimal scheduling strategy for multiple control tasks on a shared computation platform that uses feedback from the physical system to optimise the control quality. While [9] still requires a complex application-specific online optimisation strategy, the work in [10] added a dedicated feedback scheduler to control the computation resources. On a higher system level, multi-rate control and time-varying sampling were investigated in [11, 12]. However, the aim of the work in [11] is not to improve the efficiency of the ECS, but more generally to cope with variable sampling times as they occur in distributed and wireless sensor systems. The motivation in these works is to cope with timing uncertainty of sampled values, while our work aims to add time variance in order to improve computation performance without degradation the control quality of the CPS.

Non-uniform sampling time in digital-only control is studied in [13] and the proposed method is applied for tracking control of a linear actuator. Khan [13] applies non-uniform sampling time for a lower average sampling rate to achieve a lower average processing time. The proposed method is based on an adaptive change of digital controller coefficients as the sampling time changes, while a number or simplifying assumptions have been made such as linearity and time-invariance of the physical system dynamics.

In [14], different non-uniform sampling schemes, such as variable sampling period, non-synchronous sampling and multi-rate sampling are discussed for heterogeneous sensor systems. The solution, however, relies on linear dynamics in the physical system, which is not applicable to many CPSs.

In [15], authors have shown that optimal adaptive control algorithms can be developed using RL. The authors also have used practical examples that RL-based controller can achieve desirable results in real time. While our work does not aim to optimise the control quality as shown in [15], we apply the RL technique to control the properties of the ECS.

Event-triggered control and its variants also can realise non-uniform sampling in control systems [16]. However, in event-triggered control, a number of restricting assumptions have to be made such as explicit modelling of the physical system and existence of the Lyapunov control function. We have used an event-triggered method as baseline solution in one of the case studies.

3 Reinforcement learning

In this section, we briefly review RL and introduce the notations used in the rest of paper. In Fig. 1, the agent–environment model of RL is shown. The 'agent' interacts with the 'environment' by applying 'actions' that influence the environment state at the future time steps and observes the state and 'reward' in the next time step resulting from the action taken. The 'return' is defined as sum of all the rewards from the next steps to the end of current 'episode':

$$G_t = \sum_{i=t+1}^{T} r_i \tag{1}$$

where G_t is the return at time t, r_i are future rewards and T is total number of steps in the episode. An 'episode' is defined as a sequence of agent–environment interactions. In the last step of an episode the control task is 'finished.' Episode termination is defined specifically for the control task of the application.

For example, in the cart-pole balancing task that we discuss in more detail in Section 5, the agent is the controller, the environment is the cart-pole physical system, the action is the force command applied on the cart, and the reward can be defined as $r = 1$ as long as the pole is nearly in upright position and a large negative number when the pols falls. The system states are cart position, cart speed, pole angle and pole angular speed. The agent task is to maximise the expected return G_t, which is equivalent to preventing pole from falling for the longest possible time duration.

In RL, a control policy is defined as a mapping of the system state to the actions:

$$a = \pi(s) \tag{2}$$

where a is the action, s is the state and π is the policy. An optimal policy is one that maximises the expected return for all the states, i.e.:

$$v_{\pi*}(s) >= v_{\pi}(s), \quad \text{forall } s, \pi \tag{3}$$

where v is the expected return (value) function. Equation (3) means that the expected return under optimal policy $\pi*$ is equal or greater than any other policy for all the system states.

Another important concept in RL is the action-value function, $Q_\pi(s, a)$ defined as the expected return (value) if action a is taken at state s under policy π. This function is related to the optimal value function introduced in (3) by the following equation:

$$v_{\pi*}(s) = \max_a Q_{\pi^*}(s, a) \tag{4}$$

To develop algorithms to find the optimal policy, $\pi*$, the environment dynamics need to be modelled. Contrary to most of the control design methods, many RL algorithms do not require the models to be known beforehand. The elimination of the need of modelling the system under control is a major strength of RL. The main assumption about the environment is that it has the *Markov property*. A system has the Markov property if at a certain time instant, t, the system history can be captured in a set of *state variables*. By the Markov property assumption, the RL problem can be expressed as *Markov decision process* (MDP). The MDP problem can be modelled with the following conditional property:

Data: $Q_0(s,a)$ ▷ Initial action value function
s_0 ▷ Initial State
Result: $Q^*(s,a)$ ▷ Optimal Action-Value function
1 $Q \leftarrow Q_0$;
2 **repeat**
3 $s \leftarrow s_0$;
4 **while** *s is non-terminal* **do**
5 $a \leftarrow \pi_{\epsilon,Q}(s)$;
6 Take action a observe new state s' and reward r;
7 $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \max_{a'} Q(s',a') - Q(s,a)\right)$;
8 $s \leftarrow s'$;
9 **until** *convergence*;

Fig. 2 *Algorithm 1: Q-learning control*

Data: $\boldsymbol{\theta_0}$ ▷ Initial parameter vector
s_0 ▷ Initial State
Result: $Q^*(s,a)$ ▷ Optimal Action-Value function
1 $\boldsymbol{\theta_a} \leftarrow \boldsymbol{\theta_{a0}} \quad \forall a \in \mathcal{A}$;
2 **repeat**
3 $s \leftarrow s_0$;
4 **while** *s is non-terminal* **do**
5 $a \leftarrow \pi_{\epsilon,Q}(s)$;
6 Take action a observe new state s' and reward r;
7 **if** *s' is terminal* **then**
8 $\theta_{ai} \leftarrow \theta_{ai} + \alpha(r - Q(s,a)) \quad \forall i \in \{i | f_i(s) = 1\}$;
9 **else**
10 $\theta_{ai} \leftarrow \theta_{ai} + \alpha(r + \max_{a'} Q(s',a') - Q(s,a)) \quad \forall i \in \{i | f_i(s) = 1\}$;
11 $s \leftarrow s'$;
12 **until** *convergence*;

Fig. 3 *Algorithm 2: Q-learning control for linear approximated value function and binary features*

$$p\{s(t+1), r(t+1) | \text{system history up to time } t\} = p\{s(t+1), r(t+1) | s(t), a(t)\}, \tag{5}$$

which means that the reward and the next state only depend on the current state and action. Markov property holds for many of CPS application domains and therefore MDP and RL can be applied as the control algorithm.

Q-learning [17] is an important example of RL solution methods and it is used in this paper as the optimal policy learning method. The Q-learning control algorithm is shown in Algorithm 1 (see. Fig. 2). In Q-learning, to find the optimal policy, we start from an arbitrary initial action-value function Q_0 and update it at each step of MDP by observing the reward gained by the taken action. Therefore, the optimal policy can be learned by the agent just by interacting with the environment. At each learning algorithm step, the greedy policy π^*_Q corresponding to learned action-value Q function can be defined as

$$\pi^*_Q(s) = \arg \max_a Q(s,a) \tag{6}$$

As the learning algorithm proceeds, the learned $Q(s, a)$ function converges to the optimal $Q*(s,a)$ and therefore the greedy policy converges to optimal policy $\pi*$ defined in (3). However, to explore the action-state space for the optimal solution, an exploratory policy should be used. One example of such policies is ϵ-greedy policy defined as

$$\pi_{\epsilon,Q}(s) = \begin{cases} fa_i, \quad i \in \{1, \ldots, N_a\} & \text{with probability } \epsilon/N_a \\ \pi^*_Q(s), & \text{with probability } 1-\epsilon \end{cases} \tag{7}$$

where N_a is the number of available actions. Equation (7) means that a random action is picked instead of the greedy action with small probability ϵ in ϵ-greedy policy. In CPS applications, we choose ϵ depending on the uncertainty level in the application domain. For example, in our case study, a very small value is chosen for ε since the system is deterministic.

In Q-learning (Algorithm 1, Fig. 2), the agent starts from the initial state, takes action using ϵ-policy and observes the reward. The incremental optimal policy learning is done in line 7. α is the step size parameter used to update current action-value function towards greedy target value at each iteration.

3.1 Linear approximation of continuous value functions

The Q-learning algorithm described in Algorithm 1 (see Fig. 2) is applicable to a discrete state space where the action-value function can be defined in a tabular representation. However, in most CPS applications, the state space is continuous and cannot be expressed by a finite number of states. To be able to use methods mentioned in previous subsection, one approach is to approximate the continuous space with a linear combination of feature functions of the state variable. Coefficients of the mentioned linear combination can be expressed as the parameter vector:

$$\boldsymbol{\theta} = (\theta_1 \quad \cdots \quad \theta_{N_f})^T. \tag{8}$$

In this case, the action-value function can be expressed as

$$Q(s,a) = \boldsymbol{\theta}_a^T \boldsymbol{f}(s) = \boldsymbol{\theta}_a^T \left(f_1(s) \quad \cdots \quad f_{N_f}(s)\right)^T \quad a \in \mathcal{A} \tag{9}$$

where $f_i(s)$ are the feature functions of the continuous state space, N_f is the number of functions and $\mathcal{A}$ is the finite action set. Here we assume that only the state variables are continuous and the action space is still discrete and finite.

The objective of the learning algorithm is to estimate the parameter vector $\boldsymbol{\theta}$. The gradient-descent method can be used for this purpose. Assuming that f_i are binary functions, the Q-learning for linear approximate value function can be described as shown in Algorithm 2 (see Fig. 3).

An example of binary features is tile coding [18] where each dimension of the continuous state space is divided into a number of disjoint intervals. For example, if the state space has d dimensions and each dimension is divided into k intervals, the whole space is divided into d^k hyper-cubes (tiles). Each tile i defines a feature as

$$f_i(s) = \begin{cases} 1 & s \text{ resides in tile } i \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Algorithm 2 (Fig. 3) with tile coding function approximation requires low computational overhead and can be readily implemented in an embedded controller because we update the parameters towards the greedy value only for the activated features (lines 8 and 10 of Algorithm 2, Fig. 3).

4 A-ECS based on RL

In this section, first, we explain our proposed A-ECS framework to extend the adaptive control concept to change embedded system parameters (e.g. sampling time or voltage) in real time based on RL methods. We apply the term *adaptive* not only for the controller but also in a broader sense, that is changing ECS system parameters, such as sampling time or memory allocation, based on the online system state. We also introduce the specific variable sampling time ECS (VS-ECS) as an example of A-ECSs where the controller sampling time is changed in real time to realise more efficient embedded control in CPS applications.

In the second part of this section, we discuss our cloud-based evaluation framework as an extension to facilitate simulation-based design and evaluation of A-ECS. Further, we explain the steps to apply our methods on a generic CPS application.

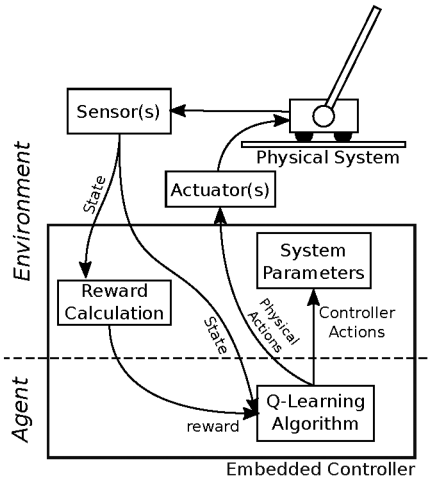

Fig. 4 *Proposed RL-based A-ECS*

1 $s \leftarrow s_0$;
2 **while** *true* **do**
3 $(\boldsymbol{a_p}^\intercal \quad h)^\intercal \leftarrow \pi^*_Q(s)$;
4 apply $\boldsymbol{a_p}$ to the physical system (actuator);
5 wait for h seconds ;
6 observe new state s' ;
7 evaluate the reward based on observed s';
8 Perform one step of Algorithm 2 to learn $\boldsymbol{\theta}^*$ (lines 7 to 10);
9 $s \leftarrow s'$;

Fig. 5 *Algorithm 3: variable sampling-time embedded control*

4.1 A-ECS RL environment and actions

The A-ECS can be realised by the following two extensions to the conventional RL-based control algorithm:

- Since RL does not require prior assumptions about the environment and the available action set, RL can be used on a broader definition of an environment that includes the physical system along with elements of the embedded controllers.
- We can extend the action set to include actions that change ECS parameters such as sampling time and memory allocation in real time.

Once the extended RL control problem is solved, the optimal policy will include the ECS parameter real-time adaptation and physical system control commands at the same time. This idea is outlined in Fig. 4 where the environment/agent boundary is crossing the ECS so that system parameters of ECS are included in the environment.

To realise the explained A-ECS, we augment the action vector by parameter change actions. Formally, we can represent RL action vector as

$$\boldsymbol{a} = (\boldsymbol{a}_\mathrm{p}^\intercal \quad \boldsymbol{a}_\mathrm{c}^\intercal)^\intercal \tag{11}$$

where $\boldsymbol{a}_\mathrm{p}$ is a vector containing the actions that influence the physical system such as force applied to cart in cart-pole balancing task and $\boldsymbol{a}_\mathrm{c}$ is the vector of actions that change the ECS parameters, such as sampling time. The remaining RL elements correspond to conventional RL approaches, as discussed in Section 3. Therefore, we can define a reward so that the agent optimise the objectives of CPS system using conventional RL algorithms. Hence, the reward calculation based on the system state and the online learning algorithm can be integrated in the ECS.

Now, we can describe VS-ECS as an example of A-ECS described above. In VS-ECS, the sampling time is changed in a fine grained manner, i.e. in each sample time the controller decides about the very next sampling time. The controller can choose from a limited number of available sampling times. Using a variable sampling time scheme, we expect to reduce processing time and system power by decreasing the sampling rate whenever fast sampling is not required to stabilise the system. There is a trade-off in selecting number of available sampling times. If this number increased we have more flexibility and possibly better performance. On the other hand, larger number of selections degrade the performance of the learning algorithm due to the optimisation problem that needs to be solved in each time step which scales exponentially with the number of possible actions.

For the VS-ECS, the only controller parameter is sampling time h. Therefore, the action vector defined in (11) can be rewritten as

$$\boldsymbol{a} = (\boldsymbol{a}_\mathrm{p}^\intercal \quad h)^\intercal \tag{12}$$

In Algorithm 3 (see Fig. 5), the RL algorithm for VS-ECS based on RL and Q-learning is described.

4.2 Cloud-based evaluation framework

While A-ECS can be used to develop algorithms to control the physical system and change ECS parameters online with no prior modelling of the system, model-based simulations help to learn preferable parameters and policies, and test identified settings before deployment. Furthermore, with efficient simulation models we can speed up the learning process.

To improve the performance of those simulations, we propose a parallel cloud-based evaluation process using a simulation model of the physical system, the ECS and the RL algorithm. The approach helps to find a superior policy by running multiple instances of the simulation and picking the learned parameters of the instance with maximum performance. In all discussed cases, the RL-based ECS can apply the learned parameters to change ECS parameters online.

Our simulation approach is shown in Fig. 6. While the Q-learning algorithm is an iterative and therefore a sequential algorithm inherently, we still can leverage recent cloud-based parallel platforms to run multiple instances of the simulation model for statistical evaluation of overall CPS performance. To realise this requirement, we require the physical model, ECS and Q-learning algorithm expressed as ordinary differential equations (ODEs). ODEs can be efficiently solved utilising C++ and the Boost odeint library [19] to create a native executable binary file for the simulation. We can launch multiple instances to run the simulation with different random seeds. Then we can run a 'reduce' script that aggregates simulation results of multiple instances, i.e. training curves (RL return versus training step number) and episode trajectories. The reduce script also generates evaluation statistical results. For example, we can pick the learned parameters of the instance that achieves the maximum performance. Fig. 6 shows the flow of the evaluation framework. In this figure, some examples are given for each block inside the parenthesis.

Although the described evaluation framework is not strictly 'model free', for many complex systems it is a considerably easier task to build simulation models instead of explicit models needed in conventional control design methods. Even if we can develop explicit or the analytical models of challenging systems (e.g. non-linear, hybrid, time-varying etc.), control algorithm design is not trivial using conventional methods.

4.3 A-ECS development workflow

To summarise this section, we provide a list of required steps to apply our framework for a CPS. The steps are:

1. Identify the RL elements in the CPS, i.e. environment and actions. Especially, it should be decided which parts of the ECS can be changed in real time and what are the actions that apply these changes.
2. Design a reward formulation based on the performance objectives. In contrast to conventional control theory, we can address actual design objectives directly, by rewarding the agent (controller) proportional to the most important

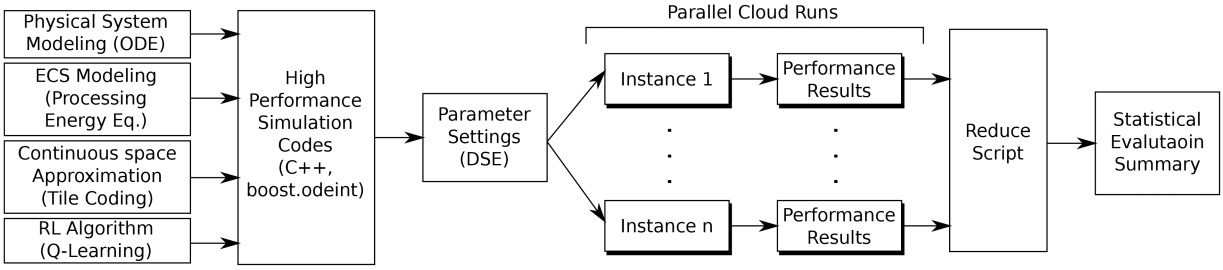

Fig. 6 *Cloud-based evaluation framework*

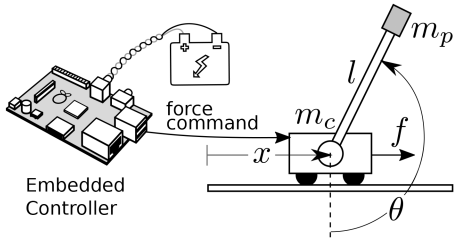

Fig. 7 *Cart-pole case study*

performance metrics and penalise it with large negative rewards in case of failures.

3. Choose an RL algorithm to learn the optimal policy. For example, Q-learning algorithm explained in Section 3 can be used.
4. Choose an approximation method for the continuous state space. For example, the tile coding described in Section 3 is one of the possible approaches. Recent deep neural network representation method is an alternative for more complex systems [7].
5. Develop the simulation codes for the physical system and ECS. Also, implement the RL algorithm and reward calculations. Next, integrate all the mentioned components and performance measure output generation codes.
6. Choose system and RL parameters based on available heuristics or by iterative design-space exploration using proposed framework. Some example of these parameters are ϵ, α, number of tilings and number of grids for each continuous state-space dimension.
7. Build executable of the simulation, launch independent parallel instances to run the simulation models for different random seeds.
8. Use the 'reduce' script to extract statistical information such as average or maximum performance.

5 Case study 1: cart-pole swing up task

In this section, we follow the steps listed in A-ECS workflow for the cart-pole swing up task. We apply the cloud-based evaluation framework described in Section 5.3 to show the performance improvement using the variable sampling time.

Consider the cart-pole system depicted in Fig. 7. The processor, powered by a battery, can generate force commands applied to the cart. The control task is to swing up the pole from fall position to upright position and keep the pole upright for the longest time period possible using the limited energy in the battery. We explain the steps to develop the VS-ECS to balance this system using a digital controller.

The first step is to define different elements of the RL framework:

Environment: In RL, the environment is defined as the part of system which can be influenced by the agent. By this definition, the environment is the physical system in the classical cart-pole example because the applied force is the only action available to the agent. In A-ECS, the concept of environment had to be extended to include properties of the controller itself because the agent can change the controller parameters dynamically.

Agent: In A-ECS, the agent is the embedded controller. More precisely, the 'fixed' elements of the controller is the agent and the 'varying ' elements are considered part of the environment as explained before.

Actions: A-ECS supports two set of actions: physical system actions and controller parameter tuning actions. In the cart-pole example, the force command to the cart is the physical action and the sampling time is the controller parameter action. Both actions are continuous variables, but for simplicity they are defined as discrete quantities in the case study. The force can be zero, or maximum force in any of two directions (right and left in Fig. 7). The sampling time can be chosen from some bounded number of available choices. Therefore, we define the action vector consisting of force command and sampling time as

$$\boldsymbol{a} = (f \quad h)^{\mathrm{T}} \tag{13}$$

System state variables: The system state variables are

$$s = (x \quad \dot{x} \quad \theta \quad \dot{\theta} \quad e)^{\mathrm{T}} \tag{14}$$

where x is the cart position, θ is the pole angle and e is the current battery energy.

Policy π: The policy π is defined as mapping of system state to optimal actions, that is the force applied to cart and the next sampling time as a function of current physical system state and the battery storage.

Reward: In the classical cart-pole example, the reward is defined as positive value (e.g. one) if the pendulum is in the upright position with some tolerance ($\pi - \delta < \theta < \pi + \delta$) and a large negative value if the pole falls. In our example, we define the reward as the time period that the controller can keep the pole in upright position. By this definition, the agent tries to use longer sampling times to save processing power to be able to balance the pole for a longer time. We also add a term proportional to the angular distance of pole to the upward position to encourage swinging the pole.

Next, we chose Q-learning and tile coding approximation function to implement the RL algorithm as described earlier in this paper, while other state-of-the-art approaches can be used as well.

The next step is to simulate the physical system and processing power consumption. In the next subsections, we describe the details of simulation models that we implemented in C++ to use them in the cloud-based evaluation process. The final steps apply the simulation code in the cloud computing platform and summarise the results by the analysis scripts. In Section 4, the results of proposed VS-ECS applied on the cart-pole case study are given.

5.1 Cart-pole dynamics

Now we explain the modelling of the physical system that is implemented in the simulation model used in the evaluation framework in Section 5.3. The cart-pole system is modelled using dynamics differential equations. The kinematic and potential of the cart-pole system is derived by

$$T = \frac{1}{2}m_{\mathrm{c}}\dot{x}^2 + \frac{1}{2}m_{\mathrm{p}}\left((\dot{x} + l\dot{\theta}\cos\theta)^2 + l^2\dot{\theta}^2\sin^2\theta\right) \tag{15}$$

$$U = -m_{\mathrm{p}}gl\cos\theta \tag{16}$$

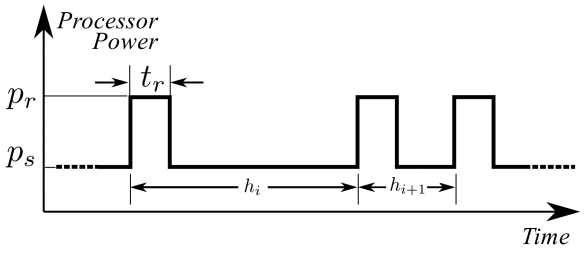

Fig. 8 *Processing power temporal modelling*

Table 1 Parameter value settings for the experiment

Parameter	Value
x range	±6 m
θ range	$\pm 2\pi$, rad
θ balance range	$\pi \pm 0.1$, rad
L	20 cm
m_c	1 kg
m_p	0.1 kg
k_1	0, N/m/s
k_2	0, Nm/1/s
ϵ	0.001
α	0.7
f_{max}	200 N
t_r	200 µs
p_r	220 mW
p_s	16 µW
battery capacity	0.3 J
No. of tiling grids/dimension	7
No. of tilings	40

The Lagrangian using (16) can be written as

$$L = \frac{1}{2}(m_c + m_p)\dot{x}^2 + m_p l\dot{\theta}\dot{x}\cos\theta + \frac{1}{2}m_p l^2\dot{\theta}^2 + m_p gl\cos\theta \tag{17}$$

We can write the differential equations of the pole motion as

$$(m_c + m_p)\ddot{x} + m_p l\ddot{\theta}\cos\theta - m_p l\dot{\theta}^2\sin\theta = F - k_1\dot{x} \tag{18}$$

$$m_p l\ddot{x}\cos\theta + m_p l^2\ddot{\theta} + m_p gl\sin\theta = -k_2\dot{\theta} \tag{19}$$

Finally, solving (19) for the linear acceleration of cart and angular acceleration of pole we have

$$\begin{aligned}\ddot{x} = &\frac{1}{m_c + m_p\sin^2\theta} \\ &\times \left(f - k_1\dot{x} + m_p\sin\theta(l\dot{\theta}^2 + g\cos\theta) + k_2\dot{\theta}\cos\theta\right)\ddot{\theta}\end{aligned} \tag{20}$$

$$\begin{aligned}\ddot{\theta} = &\frac{1}{l^2(m_c + m_p\sin^2\theta)} \\ &\times (-(m_c + m_p)k_2\dot{\theta}/m_p - lf\cos\theta + lk_1\dot{x}\cos\theta \\ &-(m_c + m_p)gl\sin\theta - m_p l^2\dot{\theta}^2\sin\theta\cos\theta)\end{aligned} \tag{21}$$

Equations (20) and (21) are highly non-linear but we can still RL framework to control the physical system with this non-linear dynamics.

5.2 Processing power modelling

Now we explain the simulation modelling of processing power, which is directly applied for the results provided in Section 5.3. We assume that the processor of the ECS is in sleep mode between each two successive control routine invocations. The mentioned idle time can be used to do other processing tasks, but as we are focused on the power consumption of the control task we can simply assume that the controller is in sleep mode in idle time to save energy. The power consumption scheme with this assumption is shown in Fig. 8. The total processing energy is modelled by

$$\begin{aligned}E_{\text{proc}} &= \sum_{i=1}^{N}(p_r t_r + p_s(h_i - t_r)) \\ &= N(p_r - p_s)t_r + p_s T\end{aligned} \tag{22}$$

while the total time T is defined as sum of all N sampling times:

$$T = \sum_{i=1}^{N} h_i \tag{23}$$

For a fixed total time T, that means that longer sampling times, h_i results in lower total steps N, and lower N value in (22) results in lower E_{proc} which means higher power efficiency.

5.3 Simulation results

In this subsection, we describe the results for two experimental setups to investigate the efficiency of the proposed VS-ECS approach. The first setup is a swing-up and balance task, the second setup addresses the balance only of the cart-pole example. We also implement and simulate ETC for the first setup as the baseline method. We will compare the results with our proposed method in the next subsection. For each setup, we conducted three experiments with different sampling schemes:

- Fixed sampling time (with value h_1),
- Fixed sampling time (with value h_2), and
- Variable sampling time (with values either h_1 or h_2 decided in real time and in each control step).

We use the simulation models and the Q-learning algorithm explained previously to run the experiments. Table 1 lists system parameters that are used in the experiments. All experiments are done for 25 million steps. After every batch of 10,000 steps, the framework evaluates the control policy learned by the RL agent. This is done by running the greedy policy and calculating the return which is defined as the time period in which the agent was able to keep the pole almost in upward position ($\pi - 0.1 \le \theta \le \pi + 0.1$) before the battery energy is completely depleted. An additional term in the return function encourages the agent to swing up the pole. The overall return is determined by the following equation:

$$r = 10^{-6}(\pi - |\pi - \theta|) + \begin{cases} h_i & |\pi - \theta| \le 0.1 \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

where h_i is selected sampling time at time step i.

For each experiment, we study the average balancing time and the identified maximum balancing time. Due to the invariant energy supply, a longer balancing time indicates a lower average power consumption, and therefore is desirable.

5.3.1 Swing-up and balance task: For the swing-up and balance experiment, each episode starts from the state where cart-pole system is still with $x = 0$ and $\theta = 0$ and ends whenever the battery energy is fully depleted or one of x or θ passes the allowable range listed in Table 1. $h_1 = 10$ and $h_2 = 100$ ms are used for the experiments. Conventional fixed sampling-time approaches have a desirable performance for the h_1 and fail in most cases when using h_2. We expect that the variable sampling time can achieve a higher performance by switching between the two sampling times in real time.

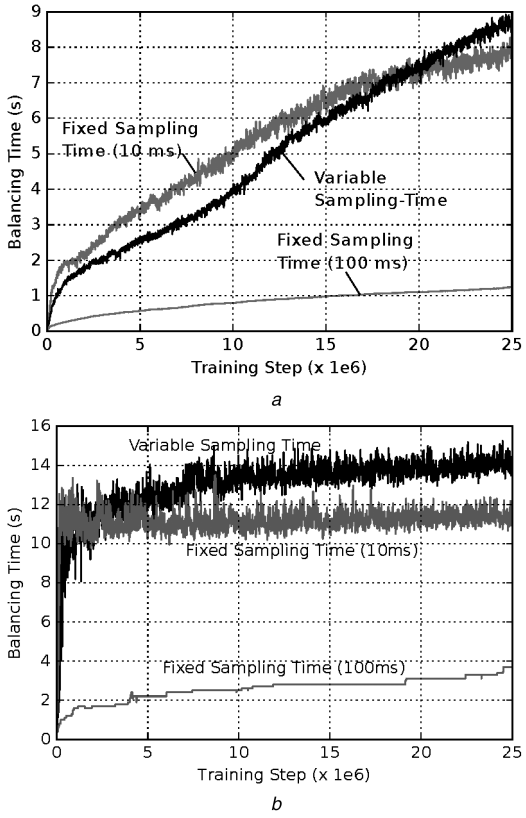

Fig. 9 *Learning curves for the average balancing time for the swing-up and balance task*

(a) Average, ***(b)*** Maximum balancing time (return) achievable by the ECS in swing-up and balance task

Fig. 9*a* shows the learning curves for the average balancing time for the swing-up and balance task. The plots are generated by averaging results of 200 runs on 32 instances launched by the cloud-based evaluation tool. The total simulation time was around 6 h on Intel Xeon E5-2600 processors. We see that the larger fixed sampling time results in a short total balancing time. The reason is the severe instability due to the large sampling time. The VS-ECS performance is lower at short term, but starts to outperform the fast fixed sampling time after around 19×10^6 learning steps, since it can utilise the two modes of operations.

At the beginning, the agent should act fast to move to pole to upright position quickly, but after that the system dynamics is slow around the balancing point and the agent should do small corrections with a slower rate that the swing up phase (Fig. 10). Therefore the probability of selecting longer sampling time is higher in the balancing state.

The benefit of VS-ECS is more obvious when we look at the maximum balancing time, shown in Fig. 9*b*. VS-ECS identifies better settings already after less than 10×10^6 learning steps and is able to balance the pole about 2 s longer than with the fast fixed sampling rate.

5.3.2 Balance-only task: The second setup considers the upright balancing time only. The control problem in this case is less complex since the pole is already close to upright positions at the start ($\pi - 0.04$). In contrast to the previous example, both applied fixed sampling times ($h_{\text{slow}} = 10$ and $h_{\text{slow}} = 1$ ms) can be used to control the system. In the new experiment setup, the allowable balance range is tighter ($\pi - 0.05 \leq \theta \leq \pi + 0.05$).

Fig. 11 shows the learning curves for the average and maximum balancing time. It can be seen that the faster fixed sampling time exhausts the battery earlier, caused by the higher processing power. However, the results also show that in average the slower fixed sampling time outlasts VS-ECS, while the maximum balancing time in both cases are equal. The results confirm that a fixed sampling scheme is more suited in system with unimodal

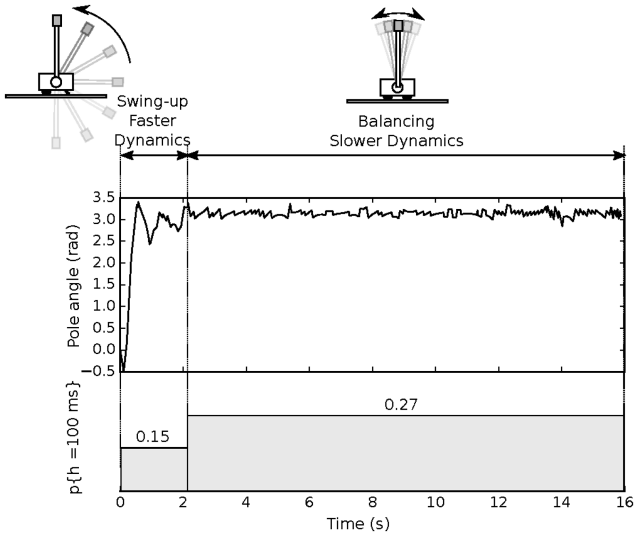

Fig. 10 *Two modes in swing-up and balance task shown by the pole angle time plot. Probability of selecting longer sampling time by A-ECS is shown in the lower time plot*

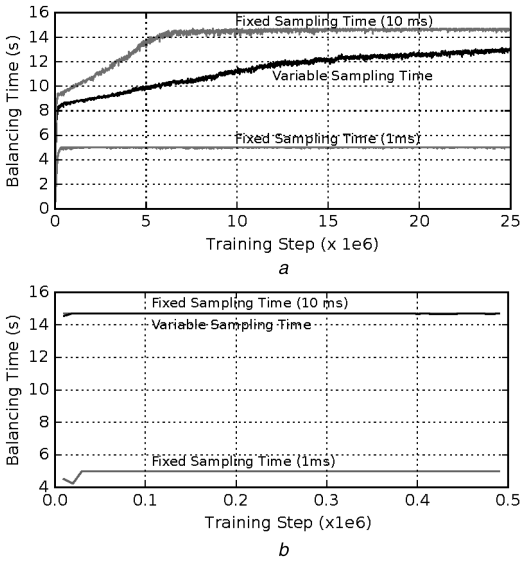

Fig. 11 *Learning curves for the average and maximum balancing time*

(a) Average, ***(b)*** Maximum balancing time (return) achievable by the ECS in balance only task

dynamics, since the system does not need to switch between modes. In these cases, VS-ECS requires learning to converge to the static behaviour of the optimum static systems.

5.4 Comparison to ETC

In this subsection, we compare the performance of proposed RL-based VS-ECS controller with an event-triggered controller (ETC) as the state-of-the-art non-uniform sampling solution for the cart-pole example. The performance metric is the balancing time as discussed in the previous subsection. The ETC for cart-pole system designed based on the method described in [16, 20]. ETC is realised by implementation of two functions, *event function* and *feedback function*. The event function, $\varepsilon: \chi \times \chi \to \mathbb{R}$, where χ is the state space indicates if a new control calculation is needed ($\varepsilon \leq 0$) or not ($\varepsilon > 0$). The first argument of ε is the memorised state of the system at last control update and the second argument is the current state. The feedback function, $\gamma: \chi \to \mathcal{U}$, is the state-feedback control law, where $\mathcal{U}$ is the control input space. Shortcoming of ETC is the required evaluation of the event function on a regular basis to detect the control update event, whereas in VS-ECS the embedded controller can go to sleep between successive control updates.

Table 2 Comparison of VS-ECS (proposed in this paper) and ETC designed by the concepts proposed in [16]

	Balancing time, s			
	Accurate physical model		Inaccurate physical model	
	Single sampling time	Adaptive sampling time	Single sampling time	Adaptive sampling time
VS-ECS	11.2	14.1	9.6	13.2
ETC	14.1	14.7	unstable	9.25

The controller which is proposed in [20] has two modes: *Swing up* and *Stabilisation*. The controller starts at Swing up mode and after it reaches a specific angle (close enough to upright position) switches to stabilisation mode. The controller uses energy control in swing up mode and linear quadratic regulator in the stabilisation mode using a linearised model. The event and feedback functions are derived for both modes in [20]. The ETC controller has a number of tunable parameters. We selected the optimal settings by exhaustive search of the design space in each case.

The physical parameters of the cart-pole systems are the same as for the previous experiments (Table 1). To study the improvement of control object metric by using ETC, the experiment is also repeated for an invariant sampling time controller with fixed period of 10 ms same as VR-ECS experiments. The results of the ETC simulations are compared with the VS-ECS maximum performance (Section 5.3.1) in Table 2. ETC's extended balancing time is caused by ETCs continuous control signal, while in RL approach, discrete control is used to limit the state-space dimension, resulting in more variation of the pendulum angle. However, in the RL-based approach adaptive rate results in more balancing time improvement comparing to ETC which is the main contribution of this paper.

The robustness of VS-ECS and ETC is also compared by an experiment where the system is designed for cart weight of 1 kg but the actual cart weight 0.7 kg. The results are also shown in Table 2. Here we see that ETC cannot stabilise the system, since ETC relies on an exact system model, while CS-ETC stabilises the system. It should be noted that the reduced balancing time for VS-ECS is not caused by model errors but by the additional weight of the pole.

6 Case study 2: Mountain Car problem

6.1 Problem definition

The Mountain Car example [21], discussed in this section, evaluates VS-ECS for a system with non-linear dynamics. In the original Mountain Car example, the goal is to control the acceleration of a car inside a valley in order to move it to the top of the mountain (Fig. 12). However, the maximum acceleration of the car is limited and it cannot be driven to the top of mountain in a single pass and the car has to go back and forth a number of times to get enough momentum to reach to the desired destination.

In the original Mountain Car problem, there is no computational overhead limitation and the car has not to stop at the destination and the goal is to reach to the destination on top of mountain in minimum time. Therefore, the RL reward is defined as follows:

$$r = \begin{cases} -1 & \text{car has not reached to the destination} \\ 0 & \text{otherwise} \end{cases} \quad (25)$$

In this experiment, we define two different goals to the original problem to make it a more difficult control task: (i) reach the destination with minimum computational overhead. (ii) car should almost be stopped (i.e. its speed should be less than a threshold) at the destination. To realise the minimum computational objective, a computational budget is defined that is decreased by one each sample time. Hence, once the car nearly stops at the destination the higher remaining computational budget means less computational overhead and should be rewarded. Using the explained problem

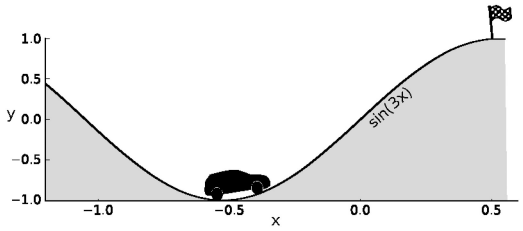

Fig. 12 *Mountain Car example*

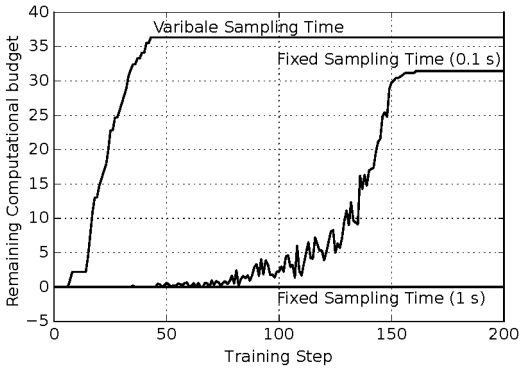

Fig. 13 *Training curves for three different scenarios in Mountain Car example*

definition, the reward defined in (25) is modified to the following reward function:

$$r = \begin{cases} 0 & x > 0.5 \\ 0 & b \le 0 \\ b & |v| < 0.1 \quad \text{and} \quad 0.44 < x < 0.45 \\ -10^{-4} & \text{otherwise} \end{cases} \quad (26)$$

where x is the car position, v is the car speed and b is the computational budget. The small negative reward in the last case included to encourage faster task completion.

6.2 Simulation results

We run the Q-learning algorithm with tile coding function approximation for three different sampling schemes: (i) fixed 0.1 s, (ii) fixed 1 s and (iii) variable 0.1 or 1 s chosen by RL agent. Fig. 13 shows the results of simulation for these scenarios. The y-axis shows the return which is the remaining computational budget in the case of successful task completion and the y-axis is the training step number. All simulations are performed with starting computational budget of 60. For the fixed 1 s, the agent is unable to command the car to reach and stop at the destination since fixed time step of 1 s is too coarse for the precise control needed to accomplish the problem objective. The agent is able to accomplish the task by the finer time precision of 0.1 s. However, using a VS-ECS in which sampling time is chosen among the fine and coarse sampling times, we can get higher efficiency of around 17%.

7 Conclusions

In this paper, we demonstrated the suitability of RL to adapt software properties of the ECS at run time. The benefit of our adaptable online approach is a reduced average power consumption of the ECS and the overall CPS, which leads to an extended life time of battery powered CPSs. Specifically for the cart-pole example we determined an extension of the life time by up to 20% compared to a system with optimal fixed settings. While our approach could not improve the power efficiency of a fixed optimal system in every case, all our experiments could achieve at least an equivalent performance. We further could outperform all sub-optimal fixed settings in all investigated scenarios, and improve the tolerance of the system to model uncertainties.

We presented a framework that facilitates the application of our approach to generic CPSs. We also presented an exploration tool that allows a designer to systematically evaluate the impact of different system settings and control modes.

While the work at hand is a very promising first step to use RL for the online software configuration of ECS, the work contains a range of limitations that might be interesting to address in future work. In our experiments, we used a dual-modal system. Adding more modes might further improve the system properties. Also applying our framework to consider voltage and frequency settings as well as resource allocations in addition to sampling rates is a promising next step. Finally, combining our approach with existing RL frameworks that focus on control properties [15] could further improve the overall design quality and system performance of CPSs, while reducing the complexity of designing the system.

8 References

[1] Juan, D.C., Garg, S., Park, J., *et al.*: 'Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling'. 2013 Int. Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013, pp. 1–10

[2] Buini, H.M., Peter, S., Givargis, T.: 'Including variability of physical models into the design automation of cyber-physical systems'. 2015 52nd ACM/EDAC/IEEE Design Automation Conf. (DAC), 2015, pp. 1–6

[3] Sala, A.: 'Computer control under time-varying sampling period: an LMI gridding approach', *Automatica*, 2005, **41**, (12), pp. 2077–2082

[4] Cervin, A., Velasco, M., Martí, P., *et al.*: 'Optimal online sampling period assignment: theory and experiments', *IEEE Trans. Control Syst. Technol.*, 2011, **19**, (4), pp. 902–910

[5] El Tantawy, S., Abdulhai, B., Abdelgawad, H.: 'Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): methodology and large-scale application on downtown Toronto', *IEEE Trans. Intell. Transport. Syst.*, 2013, **14**, (3), pp. 1140–1150

[6] Kara, E.C., Berges, M., Krogh, B., *et al.*: 'Using smart devices for system-level management and control in the smart grid: A reinforcement learning framework'. 2012 IEEE Third Int. Conf. Smart Grid Communications (SmartGridComm), 2012, pp. 85–90

[7] Lillicrap, T.P., Hunt, J.J., Pritzel, A., *et al.*: 'Continuous control with deep reinforcement learning'. 2015 arXiv preprint arXiv:150902971

[8] Neema, H., Lattmann, Z., Meijer, P., *et al.*: 'Design space exploration and manipulation for cyber physical systems'. IFIP First Int. Workshop on Design Space Exploration of Cyber-Physical Systems (IDEAL), 2014

[9] Henriksson, D., Cervin, A.: 'Optimal on-line sampling period assignment for real-time control tasks based on plant state information'. 44th IEEE Conf. Decision and Control, 2005 and 2005 European Control Conf. CDC-ECC'05, 2005, pp. 4469–4474

[10] Simon, D., Robert, D., Sename, O.: 'Robust control/scheduling co-design: application to robot control'. 11th IEEE Real Time and Embedded Technology and Applications Symp., 2005. RTAS 2005, 2005, pp. 118–127

[11] Albertos, P., Salt, J.: 'Non-uniform sampled-data control of MIMO systems', *Annu. Rev. Control*, 2011, **35**, (1), pp. 65–76

[12] Balluchi, A., Murrieri, P., Sangiovanni Vincentelli, A.L.: 'Controller synthesis on non-uniform and uncertain discrete–time domains', in Morari, M. (Ed.), '*Hybrid systems: computation and control*' (Springer, 2005), pp. 118–133

[13] Khan, S., Goodall, R.M., Dixon, R.: 'Non-uniform sampling strategies for digital control', *Int. J. Syst. Sci.*, 2013, **44**, (12), pp. 2234–2254

[14] Albertos, P., Crespo, A.: 'Real-time control of non-uniformly sampled systems', *Control Eng. Pract.*, 1999, **7**, (4), pp. 445–458

[15] Khan, S.G., Herrmann, G., Lewis, F.L., *et al.*: 'Reinforcement learning and optimal adaptive control: an overview and implementation examples', *Annu. Rev. Control*, 2012, **36**, (1), pp. 42–59

[16] Marchand, N., Durand, S., Castellanos, J.F.G.: 'A general formula for the stabilization of event-based controlled systems'. 2011 50th IEEE Conf. Decision and Control and European Control Conf., 2011, pp. 8199–8204

[17] Watkins, C.J., Dayan, P.: 'Q-learning', *Mach. Learn.*, 1992, **8**, (3-4), pp. 279–292

[18] Sutton, R.S., Barto, A.G.: '*Reinforcement learning: an introduction*' (MIT Press, 1998)

[19] Ahnert, K., Mulansky, M.: 'Odeint-solving ordinary differential equations in C++', arXiv preprint arXiv:11103397, 2011

[20] Durand, S., Castellanos, J.F.G., Marchand, N., *et al.*: 'Event-based control of the inverted pendulum: swing up and stabilization', *J. Control Eng. Appl. Inform.*, 2013, **15**, (3), pp. 96–104

[21] Moore, A.W.: 'Efficient memory-based learning for robot control'. 1990