Adaptive Workload Adjustment for Cyber-Physical Systems using Deep Reinforcement Learning*

Shikang Xu*, Israel Koren and C. Mani Krishna

Department of Electrical and Computer Engineering University of Massachusetts at Amherst, Amherst, USA

ARTICLE INFO

Keywords:
Cyber-physical systems
energy consumption
adaptive fault-tolerance
deep reinforcement learning

ABSTRACT

Reducing computational energy consumption in cyber-physical systems (CPSs) has attracted considerable attention in recent years. Associated with energy consumption is a heating of the devices. Device failure rate increases exponentially with increase temperature, so that high energy consumption leads to a significant shortening of processor lifetime.

Reducing thermal stress without harming application safety and performance is the goal of this work. Our approach is to abort control tasks dispatch when this is judged, by a neural network, to not contribute to either safety or performance. This technique is orthogonal to others that have been used to reduce energy consumption such as dynamic voltage/frequency scaling and adaptive use of redundancy. Simulation experiments show that this approach leads to a further reduction in device aging when used in conjunction with these prior techniques.

1. Introduction

This paper introduces an adaptive task dispatching algorithm for cyber-physical systems (CPSs) with high reliability requirements. The objective is to lengthen the operational lifetime of embedded processors in such applications by reducing the thermal stress on the computational platform without compromising on the quality of control provided or the safety of the application.

Motivation: Embedded applications take up most of the processors manufactured today. Many CPS applications use dozens, if not hundreds, of processing cores. When these processor boards are disposed of, they create a significant environmental problem [1, 2]. With cyber-physical systems (CPSs) growing in number and complexity, this problem is likely to become worse. Any improvement in the operational lifetime of embedded processors will therefore make a positive contribution to sustainability by (a) requiring fewer replacements over time and (b) requiring fewer onboard Line Replaceable Units in the first place, to provide enough redundancy to meet reliability requirements.

Furthermore, embedded applications have been migrating to highly cost-sensitive domains. When the typical CPS was a military aircraft, the cost of the cyber subsystem was not a limiting consideration. Today, when a typical CPS is more likely to be a commercial product, cost pressures are considerable. (For example, increasing the price of a car by just $1,000 can have a measurable impact on its fate in the marketplace.) Processors in such CPSs often have to work in poorly ventilated or otherwise difficult environments, where poor heat dissipation results in elevated temperatures that accelerate device aging and shorten useful lifetimes.

*This work has been partially supported by by the National Science Foundation under grant CNS-1717262

*Corresponding author
ORCID(s):

Organization of This Paper: This paper is organized as follows. Section 2 contains technical background information about conventional CPSs. Section 3 covers the connection between the controlled plant dynamics and safety requirements to the demands on the computational platform. It describes how information concerning the physical plant can be used to provide more lightweight fault-tolerance without sacrificing safety. Section 4 describes our adaptive control-task dispatching algorithm, which uses Q-learning(a reinforcement learning algorithm) to effectively and safely abort unnecessary control update calculations. Two case studies are presented in Section 5 to evaluate the benefits of this approach. We conclude with a brief discussion in Section 6.

2. Technical Background

CPS Structure: The structure of a typical CPS is shown in Figure 1. The computer (cyber subsystem) is in the feedback loop of the controlled plant. It receives inputs from sensors and the human operator (if any). Sensors provide raw or preprocessed data about the state of the operating environment and of the controlled plant. The cyber subsystem consists of a set of cores with associated memory, which drives actuators and then the controlled plant based on the current states and the inputs of sensors.

Computational Workload: The computational workload includes peripheral activities, such as environment and state sensing, pre-processing sensing data and communication. In this paper, we focus on the workload related with the control algorithm since it dominates the overall workload. The computational workload consists of *periodic* and *aperiodic* tasks to update the actuator values in order to meet operational needs. Periodic tasks consist of *jobs* that are released at regular intervals; for aperiodic tasks, there is a minimum interval between invocations.

Given tasks' worst-case execution times, real-time deadlines, and dispatch rates, the questions of assigning them

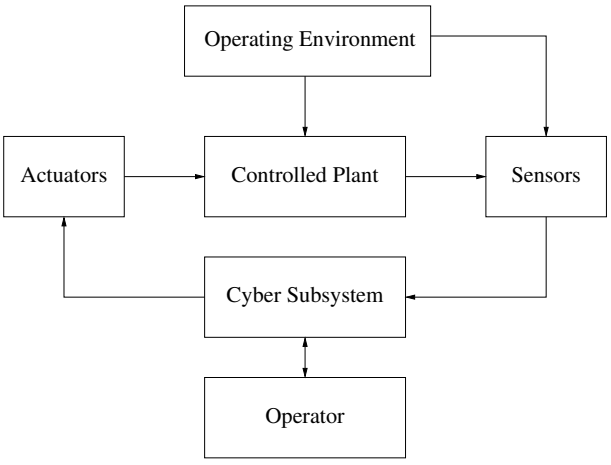


Figure 1: Basic CPS Structure

to cores and scheduling them has attracted a great deal of research [3, 4, 5]. Scheduling techniques under computational energy concerns [6] while taking thermal considerations into account [7] have also been recently developed; however, many challenges remain in these critical areas.

Reliability Imperatives: When a CPS is life-critical and therefore has to be ultra-reliable, fault-tolerance must be used. Massive redundancy and hardware and/or software diversity is traditionally the way in which this has been done, especially in aviation. This approach is followed since quite often specifications require the cyber subsystem of CPS failure probability to be very low, much lower in many cases than that of individual components. For example, the fatal failure rate for an aviation control system is traditionally set at 10^{-9} per hour [8].

The Controlled Plant and Its State Space: The controlled plant operates in a given state-space. This consists of physical variables of relevance. For example, aircraft state variables include (among others) pitch angle, pitch rate, and angle of attack. State-space equations are written to express the plant dynamics; for computerized control, these are typically a set of difference equations.

Safe operation of the plant is defined by a *safe state space* (SSS). This is the state-space region over which the plant must be kept to enable safe operation. For example, the pressure and temperature within a chemical reactor vessel must be kept under specified limits. The SSS is specified based on plant dynamics and other considerations; it is assumed to be given as input to any discussion of cyber-side reliability. Leaving the SSS at any time constitutes an unacceptable plant-failure event, whose probability must be kept under (very low) specified limits.

The SSS and plant dynamics provide a precise way in which the needs of the application are fed through to define the cyber system dependability and quality of control.

Zero-Order Hold (ZOH): We will be assuming a Zero-Order Hold model in this paper [9]. This is widely used in CPSs and consists of actuators getting updates at distcrete time instances and holding steady between updates. That is, the control surfaces are held piecewise constant, with required changes taking place between the pieces.

Quality of Control (QoC): The controlled plant actuators are set to maximize the QoC which is generally some function of the divergence of the plant's trajectory over its state space from the optimal. This is under the constraint that the plant remains in its SSS throughout. QoC depends on several controllable factors: (a) quality of sensor data, (b) rate at which the control surfaces are updated, (c) quality of the algorithm used to compute actuator settings, and (d) range of actuator capability. The dependability of the cyber system is evaluated by its ability to keep the plant within SSS throughout the period of operation.

Heating Accelerates Cyber Failure: The processor hardware failure rate is strongly dependent on operating temperature. A variety of physical processes, such as electromigration, hot carrier injection, and bias temperature instability, are much more strongly active at higher temperatures. Indeed, we often model the device failure rate as exponentially dependent on temperature; under the widely used Arrhenius model, failure rate is proportional to $\exp(-E_a/[kT])$ where E_a is a constant, called the activation energy, k the Boltzmann constant and T the absolute temperature. A generally quoted (and *very* approximate) rule of thumb is that device lifetime halves for every 10^oC increase in operating temperature (calculated using the exponential relation mentioned above).

Operating temperature depends on two factors: the amount of power consumed by the chip (turned into heat) and the effectiveness with which this heat is conducted away. This temperature can be estimated by means of a heat-flow model. The most popular of these consists of building a thermal equivalent electronic circuit, where the analogy is drawn between heat and charge flow. The rate at which heat can be dissipated to the ambient is represented by a thermal resistance, $R_{thermal}$; the amount of heat required to raise the temperature of a given node by 1^oC by thermal capacitance, $C_{thermal}$. If the power consumption at time t is denoted by $P(t)$, we have the differential equation of the node temperature at time t, $T(t)$, as follows:

$$C_{thermal}\frac{dT(t)}{dt} = P(t) - \frac{T(t) - T_{ambient}(t)}{R_{thermal}}$$

Here, $T_{ambient}(t)$ is the ambient temperature at time t. See [10] for details.

Since high temperature accelerates aging, we can define a *Temperature Accelerated Aging Factor*, TAAF(I), over an interval, I, as follows:

$$\text{TAAF}(I) = \frac{\text{Effective Aging over interval } I}{\text{Duration of interval } I}.$$

Ways to calculate this aging factor can be found in [11].

Controlling Power Consumption: To keep the power consumption low, we can do four things:

- (*a*) Use a heterogeneous set of processor cores. Tasks are then evaluated to see on which core type they consume the least energy. For instance, some algorithms may have greater affinity to GPUs than to general-purpose architectures.
- (*b*) Use Dynamic Voltage and Frequency Scaling (DVFS). The energy consumed tends to go down roughly quadratically with the supply voltage at the cost of a roughly linear increase in execution time. Voltage and frequency can be adjusted appropriately to reduce energy consumption while still meeting real-time task deadlines.
- (*c*) Only carry out redundant calculations when this is required to ensure the safety of the controlled plant. In other words, adapt the amount of fault-tolerance provided to the current needs of the application.
- (*d*) Reduce the rate at which control tasks are dispatched. This has an obvious effect on the computational workload; the challenge is to do this in such a way as to not compromise application safety or meaningfully diminish application performance.

Note that doing (*d*) increases the scope for (*b*). When we reduce task dispatch rate, we are increasing the available slack in the task schedule. Some of this slack can then be used to scale the voltage to run the processor slower and more energy-efficiently without missing any deadlines.

Controlling control task dispatch rate appropriately is the focus of this paper.

3. Task Dispatch Rate Tradeoffs

There are two key tradeoffs in changing the task dispatch rate, one obvious and the other less so. The first is the impact on the QoC; the second on the amount of redundant computations required.

Impact on Quality of Control: Delay in updating the actuator settings corresponds to feedback delay. From basic control theory, we know that such delay pushes the poles of the plant closed-loop transfer function towards the right half-plane, degrading plant performance and making it less stable. Beyond a point, aviation-failure stability is lost altogether. This problem is accentuated when there are significant disturbances from the operating environment. We therefore have a lower bound to the task dispatch rate, below which it is not safe to go.

However, as we keep increasing control task dispatch rate, diminishing returns set in and the rate of QoC improvement keeps dropping.

Also, CPSs typically have a large number of actuators. Their impact on the plant is correlated (e.g., have a multiplicative effect). That is, the impact on QoC of changing the update rate of one actuator depends on the update rate of many, if not all, of the others. In some cases, the actuators can be grouped into sets, where such correlative behavior happens within sets but the sets themselves are largely uncorrelated from one another.

Impact on Computational Workload: The impact of changing periodic task dispatch rate on the computational workload is not monotonic. The reason is that the amount of redundancy required changes as the plant nears the edge of its Safe State Space, SSS.

To make this precise, we break up the SSS into two disjoint parts, S_1 and S_3, i.e., $S_1 \cup S_3 = SSS$ and $S_1 \cap S_3 = \emptyset$. Then, a point $x \in SSS$ is said to be in S_1 if and only if the plant will remain in SSS through the subsequent task dispatch period even if its controls are set arbitrarily wrong over that period. S_3 covers the rest of SSS.

Therefore, if the plant is in S_1, there is no need for redundant calculations. Even if a failure in the computational units happens in some component (and failures are mostly transient), the plant will not become unsafe as a result of an incorrect actuator output. If, however, it is in S_3, then redundancy is required to maintain plant safety to its prescribed (high) level. (To determine whether it is in S_1 or S_3, a classifier-based approach works well: see [12].)

What does this have to do with periodic task dispatch rate and the computational workload? To keep the description simple, consider a plant with a single actuator: the idea is the same for more complex systems. Let the fraction of time the plant spends in S_1, S_3 with a task dispatch rate of ρ be denoted by $\phi_1(\rho)$ and $\phi_3(\rho)$, respectively. Let the average computational workload without and with redundancy, respectively, be W_1, W_3. Then, the total average computational workload is proportional to $W_{tot}(\rho) = \rho(\phi_1(\rho)W_1 + \phi_3(\rho)W_3) = \rho W_3 - \rho\phi_1(\rho)(W_3 - W_1)$ (since $\phi_1(\rho) + \phi_3(\rho) = 1$). Now, $W_3 \gg W_1$; typically $W_3 \geq 3W_1$. Since $\phi_1(\rho)$ is a monotonically non-decreasing function of ρ, $W_{tot}(\rho)$ is a U-shaped curve: it has a high value for small ρ since the plant spends a large fraction of its time in S_3. As ρ increases, this fraction drops and the amount of redundant calculations goes down. Beyond a certain point, the beneficial impact of an increase in $\phi_1(\rho)$ is more than offset by the impact of an increase in dispatch rate, and the workload goes up again.

The problem of setting a suitable update rate is complicated by the fact that in a real system, there are multiple actuators which (as mentioned earlier) interact with one another in their impact on QoC. Disturbances (e.g., turbulence, noise) in the operating environment also play a part. Rather than do an analytical search for the appropriate dispatch rates, we instead propose in this paper to use a machine learning approach.

4. Task Dispatch with a Q-Learning Agent

4.1. Outline of Approach

Our approach can be summarized as follows.

- For each periodic control task, τ_i, set a baseline dis-

patch rate, ψ_i. This is the maximum rate at which jobs of τ_i are released; the minimum time between successive invocations is $\Delta_i = 1/\psi_i$. This step is determined by the control engineer and treated as input by our algorithm.

- Define *tentative release epochs* $\xi_{i,1}, \xi_{i,2}, \cdots$, spaced Δ_i apart.
- At time $\xi_{i,j}$ for each j, the system will decide whether or not to actually release a new iteration of τ_i or not. Let x be the current state of the controlled plant.
  - If $x \in S_3$, the task will be released.
  - If $x \in S_1$, then use a deep Q-learning agent to decide whether or not to release τ_i.

The deep Q-learning agent has to be trained suitably, to ensure high quality decisions. It it important to note that while deep Q-learning has been used as the control algorithm for some systems or even some CPSs, the deep Q-learning agent used here is limited to tuning the control workload in CPSs, i.e., adjusting the workload dispatch rate, based on the physical states of the CPSs.

4.2. Sub-state-space Identification

As explained in the background section and the section above, one of the most important steps is to identify which sub-state-space the states of the CPS is in. The dispatch rate of workload can be adjusted to achieve power and thermal stress reduction only when the state of CPS is in S1. There are multiple ways to do this.

The most intuitive and simplest way is to use a table to save the states in S1 and to consult the table when the sub-state-space identification is needed. However, this approach is impractical for CPS as the states of CPS consist of continuous physical variable and it is impossible to maintain a table that can precisely cover the whole state space. The sub-state-space identification is a typical classification problem, which can be efficiently solved using machine learning. Thus, in this research, we use machines learning techniques to obtain sub-state-space classifiers. If the state space is of low dimensionality, decision tree(s) can be used. With high dimension state space, neural networks can be used. The training of sub-state-space classifier will happen offline. In actual deployment, the offline-trained classifier is executed. During operation of CPS, the execution of the offline-trained classifier will introduce minimal overhead [13].

4.3. Training of Deep Q-Learning Agent

Deep Q-learning is a neural network approximation to optimization by Markov Decision Theory (MDT). In MDT, we take actions in each state of a Markov chain and calculate the expected reward from the present to infinity (in one version; in others, one can have a finite time horizon). Actions are taken to maximize this expected reward. Denoting by $V(i)$ the expected such reward if the present state is i, we can write the MDT equation:

$$V(i) = \max_a \{ R(i,a) + \alpha \sum_j p_{i,j}(a) V(j) \}$$

where $R(i,a)$ is the immediate reward for taking action a in state i and $p_{i,j}(a)$ is the probability of making the transition from state i to j as a result of taking action a. $\alpha \in [0,1)$ is a discount factor which expresses the amount by which future rewards are discounted at the present time. See [14] for more details. If $R(\cdot,\cdot)$, $p_{i,j}(\cdot)$, and α are known, then it is fairly simple to obtain $V(\cdot)$ by means of value iteration. Then, the optimal action to take in state i is whatever action maximizes the right-hand side of the above equation. The action, in our case, would be to either release or not release a task iteration.

However, such an approach is impractical for our purposes. To begin with, we do not know $R(\cdot,\cdot)$ or $p_{i,j}(\cdot)$ ahead of time. Even if we did, we could not afford to store $V(i)$ given that any reasonably fine-grained sampling of the controlled plant state space would result in too large a table. The obvious solution is to learn these quantities as we go along and use a neural network in which to (approximately and implicitly) store the optimum action for each state. This is done by the well-known deep-Q learning algorithm [15].

We define our immediate reward as the weighted sum of the QoC and the TAAF, both calculated over the interval to the next decision epoch for this task. Weights have to be provided by the user, based on the relative importance of each quantity.

The weights of the neural network that is used to determine the optimum action are obtained by training it on a simulation of the controlled plant and its operating environment. (Note that further training can also be done in parallel with actual plant operation, based on the observed or recorded behavior of the controlled plant. This would follow the same procedure as described here.)

The training process is shown in Algorithm 1.

The algorithm is run for a number of training cycles. It starts with selecting, at random, a starting state for the controlled plant in S_1. The first iteration of every control task is executed to set up the simulation just prior to the entry into the **for** loop (line 3).

After some housekeeping steps (lines 5 and 6), it simulates the system by taking an action. This action is specified by the function `getAction()`.

The controlled plant is then simulated starting from state s and with action a; as a result, at the next decision epoch, it is found in state s' having accrued over the past interval, a reward of r. (If $s' \in S_3$, meaning that the controlled plant has wandered into an area requiring fault tolerance, a large penalty is applied to discourage such a step.) The history is updated and the plant state is reset to this new state, s'. This process continues until the plant leaves S_1. (A failsafe loop-exit criterion if it does not leave S_1 for a very long time is not shown here.) The neural network weights and the exploration factor are then updated using the standard approach in Q-learning [15].

Algorithm 1 Training of Q-Learning agent

ϵ=1 //*exploration factor*
CPS //*system under control*
for each training cycle **do**
    s=random selected state in $S1$
    CPS.reset(s)
    h=[] //*record states/rewards history*
    while $s \in S_1$ **do**
        a =getAction(s)
        s',r = CPS.sim(a)
        if $s' \notin S_1$ **then**
            r=large negative value
        end if
        h.append([s,a,s',r])
        $s = s'$
    end while
    Update Weights for the Q-Value network based on h
    Update ϵ value
end for
getAction(s,ϵ)
  b =random number in (0,1]
  if $b > \epsilon$
    return action with highest Q value on state s
  else return random action

5. Case Studies

In this section, we use two case studies to compare the workload adjustment approach presented in this paper against a baseline algorithm. We first outline the baseline algorithm before describing the reward function used. Then, our case studies are introduced: involving (*a*) quadcopter control and (*b*) adaptive cruise control in cars.

5.1. Baseline Algorithm

The baseline algorithm we use to compare the present approach appeared in [13]. It also uses adaptive fault tolerance based on the S_1 and S_3 subspaces, but relies on a more static approach.

The pseudocode for this baseline algorithm is shown below.

Algorithm 2 State Based Adaptive Control Selection

Input: $S1$ associated with different control period, $S1(P_1)$,$S1(P_2)$...$S1(P_k)$ (P_1<P_2<...<P_K).
At the end of every frame, with system state s:
if $s \in \cup_{i=1}^{k} S_1(P_i)$ **then**
    $p = \max\{i | s \in S_1(P_i)\}$
else
    $p = P_1$
end if

It is a lightweight greedy algorithm. All control task periods are assumed to be identical. This is frame-based control, with a single dynamically adjustable frame duration, and each control task releasing one iteration each frame. The duration of the frame can be adapted based on the current state of the controlled plant.

The algorithm selects from among a menu of allowed periods (i.e., frame lengths), $P_1 < P_2 < \cdots < P_k$. Its objective is to use the largest among these periods for which no fault-tolerance is required *for the present task iteration*, if such a selection is possible. As mentioned before, it is greedy, in that no consideration is given to the impact of the period selection on *future* task iterations.

It uses a classifier, which is offline trained using machine learning techniques (see [13] for details), to rapidly determine if the current state is in S_1 for any of the allowed frame lengths. If this is so, it selects the longest frame length (i.e., the greatest period) which satisfies this requirement. (Note that S_1 tends to shrink as the frame length increases. The reason is that increasing the frame length tends to increase the controlled plant feedback delay which degrades the QoC and tends to increase the chance of the plant ending up closer to the edge of the SSS and in subspace S_3.) If, on the other hand, the current state is in S_3 even for the smallest period, P_1, this is the one that is chosen.

Experimental results reported in [13] suggest significant performance improvements for this baseline approach over the traditional non-adaptive application of fault-tolerance. The improvements listed for the current algorithm are those above and beyond those delivered by this baseline.

5.2. Reward Function

The reward function we use is a weighted sum of QoC and TAAF. QoC is the negative of the Integrated Squared Tracking Error (ISTE). In particular, if $s(t)$, $s_d(t)$ are the current and desired states at time t, respectively, then we have ISTE = $\int_0^T ||s(t) - s_d(t)||^2 dt$ where T is some specified integration period. The Quality of Control is the negative of ISTE.

In the two case studies, we assume a processor energy consumption of 15 watts when busy (0.5 watts when idle, no Dynamic Voltage and Frequency Scaling), an ambient temperature of 23^{o}C, and the thermal equivalent circuit parameters of $R_{thermal} = 2\Omega$, $C_{thermal} = 0.5F$. The parameters related to thermal calculation are obtained from simulation data.

5.3. Case Study 1: Quadcopter Hovering

The application in this section is a quadcopter hovering in position, trying – in the face of atmospheric disturbances – to retain its location (vertical and horizontal) and body position (rotation, pitch and yaw angle). The quadcopter has 6 degrees of freedom. There are 12 state variables: x,y,z indicating the location of the quadcopter; vx,vy,vz indicating the rate of change, i.e., speed, along these axes; θ,ϕ,ψ indicating the body position and l,q,r indicating the rotational speed of the quadcopter along its three axes. The first three groups of variables are with respect to the earth frame; the last group of variables are with respect to the quadcopter's body frame. The rotation speed along body axis can be transformed into the body position change rate using a simple transition ma-

trix.

Quadcopter dynamics can be fairly accurately linearized at the desired state with its continuous-time state equation being given by $\dot{x}(t) = Ax(t) + Bu(t)$ where A and B are sparse 12×12 and 12×4 matrices, respectively [16].

The quadcopter has four propellers; each propeller is separately controlled. Its motion is managed by applying differential controls to these; i.e., the actuators are the motors driving the propellers. The execution of the control task per propeller takes 5 ms. Model predictive control is used; the control tasks seek to adjust controls to minimize the distance of the actual plant state from the desired state, for that instant in time [17]. The matrices A and B are:

$A =$

$$\begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -0.5 & 0 & 0 & 0 & 9.81 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.5 & 0 & -9.81 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$B = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0.06 & 0.06 & 0.06 & 0.06 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
1.5 & 0 & -1.5 & 0 \\
0 & 1.5 & 0 & -1.5 \\
0.1 & -0.1 & 0.1 & -0.1
\end{bmatrix}$$

In this case study, a neural network classifier is used to determine the SSS and S_1, S_3 spaces; the approach taken is the same as in [13]. The frame period is 5 ms, i.e., all four control tasks have this period.

In the training of the deep Q-learning agent, the state of the system includes the physical state of the quadcopter as well as the latest control signals produced by the control tasks, i.e. 16 variables. The neural network used here has three hidden layers with size 16, 32 and 16, respectively.

The operating environment subjects the quadcopter to disturbances which arrive as a Poisson process; the quadcopter has to counteract these to retain its hovering position. The value of each element in the disturbance is statistically independent of the others and follows a normal distribution with mean equal to half the peak amplitude and standard deviation set to $1/6$ of the peak amplitude. The disturbance can be modeled as additive to the control applied to to the quadcopter (both the control from the controller and the disturbance are vectors of the same dimension, and the sum is element wise.) In Figures 2 and 3, the legend eXrY indicates an environmental disturbance peak amplitude (i.e., maximum value) of X with a Poisson process arrival rate of Y per second. For example, e1r2 means a disturbance of peak amplitude 1 arriving as a Poisson process with arrival rate 2 per second.

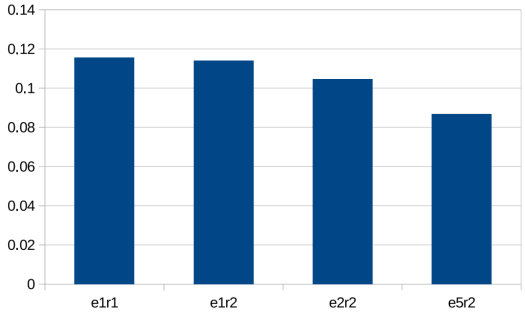


Figure 2: TAAF Improvement

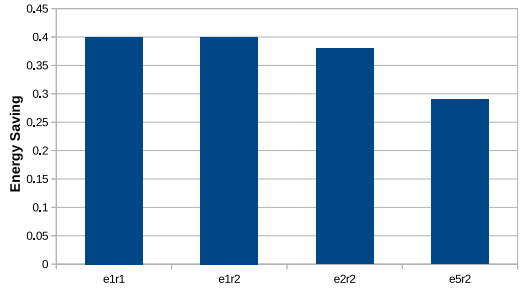


Figure 3: Energy Saving

Figure 2 shows the improvement in TAAF for the cores, over the baseline algorithm. The x-axis indicates the environment in which the controlled plant operates. We obtain a 9% to 11% improvement in lifetime; we stress that this is added improvements to that already experienced with the adaptive fault-tolerant baseline algorithm. The total energy consumption savings over the baseline are shown in Figure 3.

In Figure 4, we compare the Integrated Squared Tracking Error exhibited by (a) the present algorithm and (b) the baseline adaptive algorithm. The smaller this quantity the better the quality of control. We can see that this algorithm provides a noticeable improvement in control quality compared to the baseline approach (while still reducing thermal stress on the processors). For example, in the e1r1 case, almost 100% of the time, we have ISTE below 3 $m^2 s$, while the corresponding number for the baseline algorithm is just over 70%.

Overall, the present machine learning algorithm shows performance advantages over the baseline algorithm. This baseline, as we have pointed out, was itself shown to be a significant improvement, with respect to energy consumption and thermal stress, over the traditional non-adaptive fault-tolerance approach for cyber-physical systems.

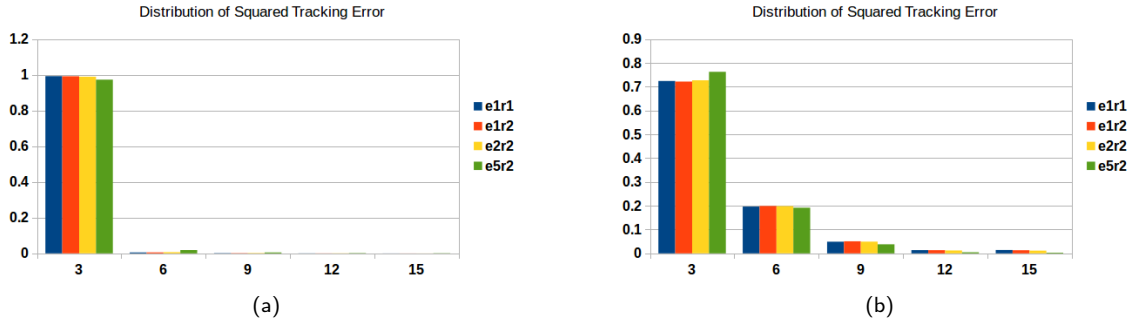


Figure 4: (a) Distribution of QoC using Deep Q-learning (b) Distribution of QoC using control period based classifier

5.4. Case Study 2: Adaptive Cruise Control

This case study is adaptive cruise control (ACC) in a car (called the *following car*) trying to maintain a safe distance from the car immediately in front of it (called the *leading car*) with varying speed. The Constant Time Gap (CTG) algorithm is used [18]. CTG seeks to keep the distance between cars equal to a constant time gap multiplied by the car speed, so that the inter-car separation linearly increases with speed; such an approach is known to have advantages with respect to keeping traffic flow smooth [19].

We follow the CTG cruise control model from [18]. Under CTG, the desired acceleration of the following car, a, is given by

$$a = -\frac{1}{h}(\dot{s} + s + hv) \tag{1}$$

where h is the time gap (input to ACC), s is the observed separation between successive cars, and v the speed of the following car. This acceleration is invoked when the following condition is satisfied:

$$s \geq hv + l + \frac{\dot{s}}{2D} \tag{2}$$

where l is the length of the leading car and D is the deceleration during coasting (no throttle and no braking).

In this case study, we assume that the maximum acceleration of the car is $3m/s^2$ (maximum throttle) and $-10m/s^2$ (maximum braking). The following car should be able to stop at least 10 meters behind the car in front if the latter comes to a stop. For the Q-learning algorithm, the relevant state of the following car is the current inter-car separation from the one in front and its speed. Available periods for the ACC task are 100 ms, 500 ms, and 1000 ms.

As before, the safe state space and S_1, S_3 subspaces are obtained from offline analysis and a classifier is used by a car to rapidly identify which subspace it is in. The subspaces for this case are shown in Figure 5.

Figure 6 shows a case where the leading car brakes hard from 25 m/s to 11 m/s (before accelerating again). The ACC task in the following car responds and quickly cuts its own speed to maintain a safe inter-car separation (Figure 6(a)). During the key period of its response, the TAAF improvement of the current algorithm over the baseline ranges from

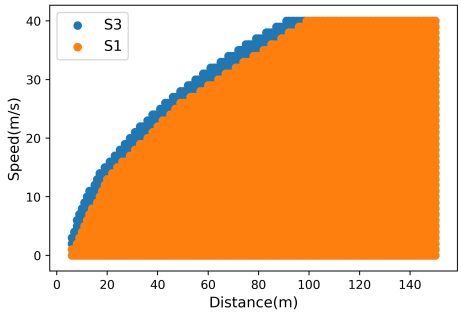


Figure 5: Case Study ACC: SSS and S1 volume

12% to 25% (Figure 6(b)). As the demands on the ACC task drop with time, this improvement also dies away, as expected.

6. Conclusion

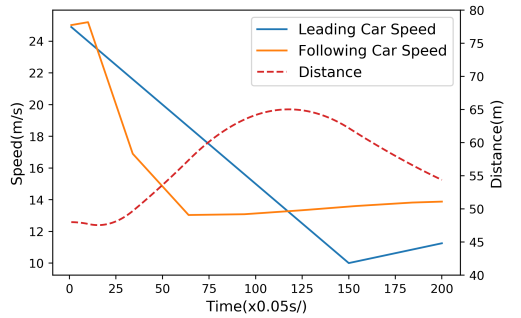
The current approach in CPSs is to update most actuator settings at a constant period. However, this can result in wasteful computation, releasing tasks for execution even when these do not result in meaningful improvements in plant performance.

We have presented an algorithm which uses machine learning techniques to adaptively dispatch control tasks in a cyber-physical system to reduce thermal-induced aging of the processors while retaining levels of application safety and performance.

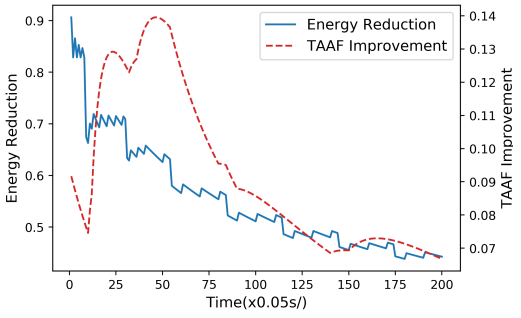
A machine intelligence approach allows the system to adapt to fault-tolerant requirements that vary with the progress of the controlled plant through its state-space. It has the advantage of not requiring accurate prior knowledge of the operating environment since it uses experience to adapt its behavior. Experimental results have shown that this approach provides additional savings on top of that achieved by using adaptive fault-tolerance.

References

[1] M. Heacock, C. B. Kelly, K. A. Asante, L. S. Birnbaum, Å. L. Bergman, M.-N. Bruné, I. Buka, D. O. Carpenter, A. Chen, X. Huo,



(a)



(b)

Figure 6: Case Study ACC (a)Physical States of System (b) TAAF Improvement and Energy Reduction

et al., E-waste and harm to vulnerable populations: a growing global problem, Environmental health perspectives 124 (5) (2016) 550–555.

[2] B. H. Robinson, E-waste: an assessment of global production and environmental impacts, Science of the total environment 408 (2) (2009) 183–191.

[3] R. I. Davis, A. Burns, A survey of hard real-time scheduling for multiprocessor systems, ACM Comput. Surv. 43 (4). doi:10.1145/1978802.1978814.

[4] C. M. Krishna, Real-Time Systems, 1st Edition, McGraw-Hill Higher Education, 1997.

[5] J. W. S. W. Liu, Real-Time Systems, 1st Edition, Prentice Hall PTR, USA, 2000.

[6] M. Bambagini, M. Marinoni, H. Aydin, G. Buttazzo, Energy-aware scheduling for real-time systems: A survey, ACM Trans. Embed. Comput. Syst. 15 (1). doi:10.1145/2808231.

[7] C. Krishna, I. Koren, Thermal-aware management techniques for cyber-physical systems, Sustainable Computing: Informatics and Systems 15 (2017) 39 – 51. doi:https://doi.org/10.1016/j.suscom.2017.05.005.

[8] Y. Zhang, Y. Li, D. Su, L. Jin, Advanced flight control system failure states airworthiness requirements and verification, Procedia Engineering 80 (2014) 431 – 436, 3rd International Symposium on Aircraft Airworthiness (ISAA 2013). doi:https://doi.org/10.1016/j.proeng.2014.09.101.

[9] K. J. Åström, B. Wittenmark, Computer-Controlled Systems (3rd Ed.), Prentice-Hall, Inc., USA, 1997.

[10] Wei Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, M. R. Stan, Hotspot: a compact thermal modeling methodology for early-stage vlsi design, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 14 (5) (2006) 501–513.

[11] C. Krishna, Ameliorating thermally accelerated aging with state-based application of fault-tolerance in cyber-physical computers, IEEE Transactions on Reliability 64 (2015) 4–14. doi:10.1109/TR.2014.2363154.

[12] Y. Xu, I. Koren, C. Krishna, Adaft: A framework for adaptive fault tolerance for cyber-physical systems, ACM Transactions on Embedded Computing Systems 16 (2017) 1–25. doi:10.1145/2980763.

[13] S. Xu, I. Koren, C. M. Krishna, Enhancing dependability and energy efficiency of cyber-physical systems by dynamic actuator derating, in: Sustainable Computing: Informatics and Systems, to appear, 2020.

[14] S. Ross, Applied Probability Models with Optimization Applications, Dover Books on Mathematics, Dover Publications, 2013.

[15] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, Adaptive Computation and Machine Learning series, MIT Press, 2018.

[16] F. D. Sabatino, Quadrotor control: modeling, nonlinearcontrol design, and simulation, Dissertation, KTH, School of Electrical Engineering (EES).

[17] M. Islam, M. Okasha, M. M. Idres, Dynamics and control of quadcopter using linear model predictive control approach, IOP Conference Series: Materials Science and Engineering 270 (2017) 012007. doi:10.1088/1757-899x/270/1/012007.

[18] R. Rajamani, Vehicle Dynamics and Control, Springer, Boston, MA, 2012.

[19] V. Milanés, S. E. Shladover, J. Spring, C. Nowakowski, H. Kawazoe, M. Nakamura, Cooperative adaptive cruise control in real traffic situations, IEEE Transactions on Intelligent Transportation Systems 15 (1) (2013) 296–305.