Experimenting with networked control software subject to faults

Brindha Jeniefer Josephrexon and Martina Maggio

***Abstract*— Faults and errors are common in the execution of digital controllers on top of embedded hardware. Researchers from the embedded system domain devised models to understand and bound the occurrence of these faults. Using these models, control researchers have demonstrated robustness properties of control systems, and of their corresponding digital implementations. In this paper, we build a framework to experiment with the injection of faults in a networked control system that regulates the behaviour of a Furuta pendulum. We use the software framework to experiment on computational problems that cause the control signals not to be available on time, and network faults that cause dropped packets during the transmission of sensor data and actuator commands.**

I. Introduction

Embedded systems are often resource-constrained, with the hardware being able to support only a limited amount of functionality. They also usually host a given number of tasks, that should be periodically completed in order for the system to work correctly. An instance of task execution is commonly called *job*. The real-time design of these systems ensures that all jobs meet their deadlines, i.e., they complete their computation within the time specified by the system designer. However, this often leads to the system design being conservative, in terms of the execution frequency of the tasks. The conservatism can lead to low processor utilisation (i.e., the hardware is often idle) and low tasks' sampling rates (i.e., tasks – among which the controller – execute infrequently).

Moreover, when these systems need to interact with any external environment, it is often beneficial to shorten the computation periods. This is usually due to the fact that something may change in the external environment, that needs an immediate reaction. In the most likely case, the *performance* of these systems is directly linked to the tasks' periods, and executing more frequently means delivering a better performance. In the case of control systems, this performance can usually be directly linked to the quality of control. This means that executing control tasks more frequently delivers a higher control precision.

Given this setting, there is obviously a trade off in the design of embedded control systems. On one hand, executing more frequently increases the quality of control. On the other hand, however, lowering the execution period also impacts the amount of computation needed to be provided by the hardware platform.

Brindha Jeniefer Josephrexon is a master student at Saarland University, Germany. Martina Maggio is a professor at the Department of Computer Science at Saarland University, Germany, and an associate professor at the Department of Automatic Control at Lund University, Sweden.

Once the control code is written and compiled, it is possible to analyse it and determine its worst-case execution time [25]. Following similar practice for all the tasks in the system (and knowing their real-time characteristics, like priorities) allows us to determine a *worst-case response time* [8]. In principle, the system designer should ensure that the task period for the controller is greater than the worst-case response time for the control task. However, this often limits the achievable performance, and hence it is sometimes beneficial to select control task periods that do not guarantee that the control task will always complete before its deadline. When the control task does not complete, the system experiences a *deadline miss*.

Noticing that deadline misses actually do happen in practice [1], researchers started investigating if deadline misses are harmful or not for control systems [12], [15], [23], finding out that in many cases control systems can tolerate packet losses [14] and deadline misses [18] by design. Alternatively, controllers can be designed to handle packet losses [2], [10], varying sampling rates [4], [21], and deadline misses [17]. To ensure the resilience of a control system to missing deadlines, however, it is necessary to understand (and to constrain) how these deadline misses occur [19].

Examples of deadline miss models are either probabilistic [22] or constraint-based [5]. In the first case, the designer tries to find the probability that a deadline is missed in the system, using a given scheduling algorithm and for a given set of tasks. A probabilistic approach is very useful to derive the stochastic behaviour and characteristics for the control system, but it does not offer strict guarantees, as it is always possible to miss every deadline in a very long sequence, even though the probability of this occurrence is extremely low. In the second case, the designer analyses the hardware and guarantees that given constraints are satisfied. For example, it may be possible to guarantee that a control task will not miss more than 2 consecutive deadlines, if its worst-case response time does not exceed 3 times its period (and the task is allowed to continue when the corresponding deadlines are missed) [7]. One of such constrained models is the *weakly-hard model* [5], [13], [24], that stipulates contracts about the *outcomes* of task jobs. An example of a weakly-hard constraint is the (m, K) constraint model, which imposes that a task cannot miss more than m deadlines in a window of K consecutive jobs [20]. Another example of weakly-hard constraint limits the amount of consecutive deadline misses that any task in the embedded system can experience.

Control systems subject to weakly-hard real-time constraints have been analysed, casting them into the framework of switching systems and trying to either find common

Lyapunov functions to guarantee stability or using upper and lower bounds for their joint spectral radius [15], [18]. Some of this work included experiments with real systems, like [23], that were testing the resilience to a burst of deadline misses followed by a recovery period.

In this paper we created a software framework that allows us to test the theoretical insights obtained in these analysis on the practical case study of a Furuta pendulum [3]. In addition to injecting deadline misses, our framework allows us to include the possibility of injecting communication failures, and faulty data transmissions [16]. Using our software stack, we test the resilience of the Furuta pendulum controller under a variety of conditions, from probabilistic constraints on the number of deadline misses to weakly-hard constraints, and transmission errors.

II. System

In this section, we outline the details of the implemented control system. We first describe the general setup for our experimental evaluation, and then enter into detail on the fault injection. Figure 1 summarises our setup. In the figure,

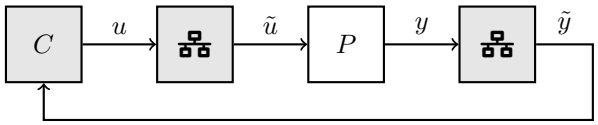

Fig. 1. Experimental setup for the control loop, including network interconnections. The plant block (white) is not subject to faults. In the controller and the signal transmission blocks (grey), we can experimentally inject faults (transmission errors and deadline misses).

we mark in gray blocks that are subject to faults that we inject, and in white the only block – the plant – in which we don't inject any fault. In principle, faults may occur in the plant, during the experiments, but are not under our control. However, we never experienced (uncontrolled) plant faults in practice.

A. Furuta pendulum and control design

Our setup consists of a Furuta pendulum built with standard components and design choices.[1] As it is common practice in real-time embedded systems [6], we design a one-step-delay state-feedback controller, hence the control signal that is calculated during the k-th execution period of the controller is applied at the beginning of the $k+1$-th controller iteration. We have access to sensor measurements that report the system state (pendulum angle, pendulum angular velocity, and velocity of the rotary joint of the pendulum base) and use literature results for both the linearised controller and an energy-based swing-up mechanism [3].

Specifically, our controller executes every $5\,ms$. When the pendulum is in the vicinity of the upright position, it uses the following control law,

$$u\left(k+1\right) = 0.375\,x_1\left(k\right) + 0.025\,x_2\left(k\right) + 0.0125\,x_3\left(k\right),$$

[1] https://build-its-inprogress.blogspot.com/2016/08/desktop-inverted-pendulum-part-1.html

where $x_1\left(k\right)$ is the pendulum angle, $x_2\left(k\right)$ is the pendulum angular velocity, and $x_3\left(k\right)$ is the velocity of the rotational joint in the base measured at iteration k. For every iteration, the controller starts with the actuation of the control signal computed during the previous execution interval, then it receives a message with the current values of x_1, x_2 and x_3, and finally it uses the remaining part of the period calculating the control signal for the following period. Clearly, it is unlikely to imagine that such a simple computation would not be able to complete in one period. However, the issue could occur in situations like security attacks that prevent the controller from completing, software bugs, or overload due to additional computation that is executed in the controller node.

The swing-up routine is performed with a first kick that sets $u\left(k+1\right) = 0.05$ and a following energy trajectory that sets $u\left(k+1\right)$ to

$$c\,\cos^4 x_1\left(k\right) \cdot x_2\left(k\right) \cdot \left(-g\,\cos x_1\left(k\right) - c\,x_2^2\left(k\right) + g\right) \cdot 10^{-2}$$

with $c = 0.0075$ and $g = 9.81$, calculated following the method in [3] with our Furuta pendulum parameters. The switch between the swing-up controller to the state-feedback one occurs when $|x_1| \leq 0.7$.

B. Fault injection models

Our experiments are aimed at testing the robustness and resilience of the control system to faults both in the computational unit, and in the network transmission, as we want our results to be representative for networked control systems.

Channel model: We model the transmission channel (and hence its possible problems) using a *Simplified Gilbert Elliot* channel model [11]. According to this model, the channel is represented with a Markov model that can be found in one of two states: transmitting (T), faulty (F). In the transmitting state, messages are delivered correctly. In the faulty state, messages are not delivered.

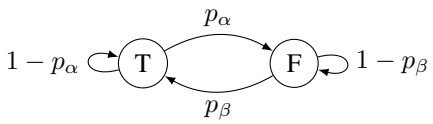

Fig. 2. Channel model: the transmitting (T) and faulty (F) states are connected by transitions that are taken at transmission time. The value of p_α represents the probability of transitioning to a faulty state from a transmitting state. The value of p_β represents the probability of the channel recovering from the faulty state to the transmitting one.

A transition in the model is taken every time a message is transmitted. In the transmitting state, the probability of a failure occurring is p_α. In that case, the channel state becomes the faulty state. Correspondingly, the probability of remaining in the transmitting state after a transmission is $1-p_\alpha$. In the faulty state, the probability of transitioning to the transmitting state is p_β and no state change occurs with $1-p_\beta$ probability. Without loss of generality, we assume the channel initial state to be T. Figure 2 illustrates the model.

Computation models: Faults can happen at the computation level. Over time, researchers have proposed different fault models for computational problems. In this investigation, we implemented the two most common ones: *probabilistic* [22] and *weakly-hard* [5]. The weakly-hard model includes four different submodels.

Before getting into the differences among the different computational fault models, there is a need to discuss what happens when a fault occurs in the computation, similarly to what is done in the literature [9], [10]. Depending on the details of the controller implementation and hardware and platform used, we investigate four different alternatives. These alternatives emerge from the need of taking two essential decisions. Suppose a fault occurred in the computation at iteration k.

1) Since our controllers are designed embedding a one step delay, at the beginning of the controller iteration $k+1$ we need to decide what control signal to apply. The two most common choices are *zero* and *hold*.
   Normally, one would expect hold to be the preferred choice. However, there are control situations in which hold is not naturally possible (for example because additional computation is required to transform the control signal from a domain to another one, which is the case with electric motors, and the lack of a control signal at the start of the $k+1$ period makes this additional computation impossible).
2) Depending on the controller implementation, the control task may be stopped by the operating system at the beginning of the next iteration (i.e., when a deadline miss occurs), or it is possible to let it continue its execution skipping the execution of the next controller iteration. These two alternatives will be denoted respectively with *kill* and *skip*.
   When skip is used, after a missed deadline, it is possible that the task completes within the following period. In principle, it would be conceivable to apply the resulting control signal as soon as it is available, but this is not the common industrial practice. On the contrary, even in case of a skipped deadline, the control signal is only applied at the following sampling instant after the task completion. While it could be beneficial to combine the continuation of the previous task with the execution of a new iteration of the controller task, this was proven harmful in terms of stability results [18] and hence we assume that if the control task overruns, then the next iteration is skipped.

Note that often the choice is linked to the specific control problem and to the choice of operating system and hardware platform, and it is not necessarily easy to select a new policy to handle computational problems. We therefore combine these choices into four different policies to analyse: Kill and Hold, Kill and Zero, Skip and Hold, and Skip and Zero. In our experimental campaign, we show the difference that these policies induce in the control performance.

If deadline misses and computational problems occur without any *regulation*, there is no hope of the control system to behave well regardless of the fault, as the system can be continuously in a faulty state. To analyse these systems fault models were introduced. As anticipated above, we implemented the injection of two very common fault models: probabilistic and weakly-hard.

Probabilistic constraints impose that for each iteration of the control task, there is a given probability to overrun the deadline. Each job of the control task is treated independently and there is no relation between a deadline outcome and the following ones. We denote with $\mathtt{p}(x)$ the probabilistic constraint where the deadline miss probability is x.

Weakly-hard constraint, on the contrary, come with guarantees on the sequence of events. According to the weakly-hard model [5], a task may satisfy one of these weakly-hard constraints (where $m, h \in \mathbb{N}$, $k \in \mathbb{N} \setminus \{0\}$, $m \leq k$, and $h \leq k$):

1) $\mathtt{am}\,(m,k)$: in any window of k consecutive jobs, at most m deadlines are missed. The term $\mathtt{am}$ stands for any miss, as the constraint specifies the number of misses in a window, and the misses can follow any pattern.
2) $\mathtt{ah}\,(h,k)$: in any window of k consecutive jobs, at least h deadlines are hit. The term $\mathtt{ah}$ stands for any hit, as the constraint specifies the number of deadline hits in a window, and the hits can follow any pattern.
3) $\mathtt{rm}\,(m)$: at most m *consecutive* deadlines are missed. The term $\mathtt{rm}$ stands for row miss, as the constraint specifies the number of consecutive misses experienced by the task. While usually these constraints are specified using a window of consecutive jobs (i.e., a sliding window), there is no need in this case to specify the window parameter, which is always equivalent to $m+1$.
4) $\mathtt{rh}\,(h,k)$: in any window of k consecutive jobs, at least h *consecutive* deadlines are hit. The term $\mathtt{rh}$ stands for row hits.

We test the control system in the presence of combinations of injected faults. For comparison, we run the control system 50 times for a duration of a minute in nominal conditions, i.e., without any fault.

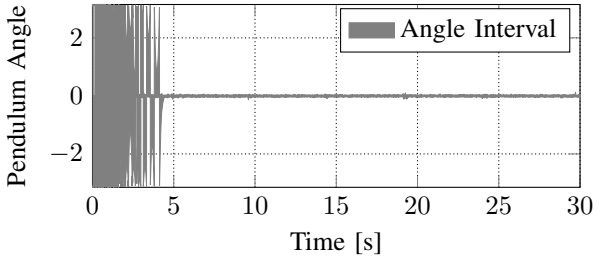

Fig. 3. Pendulum angle interval over 50 runs in nominal conditions (i.e., in absence of deadline misses and transmission failures).

In Figure 3 we show the interval of measurements in which the pendulum angle can be found, 0 corresponding to the upright position. Clearly, during the transient phase

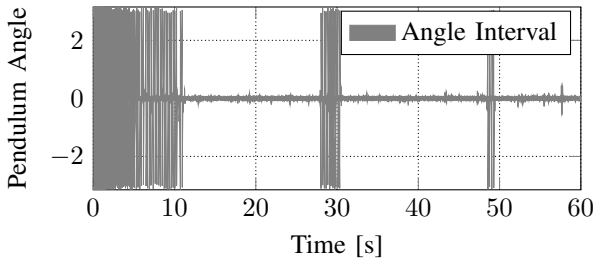

Fig. 4. Pendulum angle interval over 50 runs for $\mathtt{am}\,(2, 10)$ with Skip and Hold, when $p_\alpha = 0.2$ and $p_\beta = 0.8$.

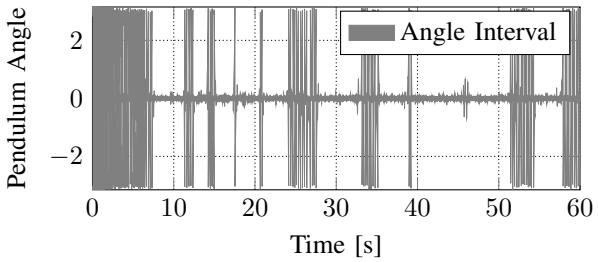

Fig. 5. Pendulum angle interval over 50 runs for $\mathtt{p}\,(0.4)$ with Kill and Hold, when $p_\alpha = 0.2$ and $p_\beta = 1.0$.

the pendulum angle can be found in different positions and the trajectory that leads to the upright position can vary. For every iteration of the controller, the figure shows the area between the maximum and minimum value experienced for the pendulum angle during the runs. As can be seen, all the runs reach the equilibrium point within than 5 seconds and then for all the runs, the pendulum remains in the vicinity of the equilibrium point. We built our testing framework to log the system state in every iteration of every run, and analyse the data according to a few performance metrics. One of these metrics is the percentage of time spent in the upright position during the entire duration of the experiment, which (for a single experiment, lasting 60 seconds, in which the pendulum is not experiencing any fault) is around 90% on average.

III. Experimental results

This section reports on our experiments conducted with the Furuta pendulum. We compare our results with the nominal control result shown in Figure 3, where (in 50 runs) the pendulum reaches the upright position in less than 5 seconds in the worst case.

Figures 4 and 5 show respectively the angle interval over 50 iterations when the software experiences respectively faults with $\{\mathtt{am}\,(2, 10), p_\alpha = 0.2, p_\beta = 0.8\}$ and $\{\mathtt{p}\,(0.4), p_\alpha = 0.2, p_\beta = 1.0\}$. In both cases, for some of the 50 runs the pendulum is not kept in the upright position for the entire duration of the experiment and it falls and experiences a swing up. Note that the figures show a summary of the 50 experiments, meaning that the angle can be found in the interval, but may be around the equilibrium point for some experiments and not in the upright equilibrium interval for some other runs.

As can be seen, it takes a longer time on average for the pendulum to reach the upright position in the first case (Figure 4), while the pendulum seems to be quicker to reach the equilibrium point in the second case (Figure 5). However, the pendulum seems to experience on average more falls during the second set of experiments.

On average, the probability of finding a deadline miss in the first set of experiments (Figure 4) is upper bounded by 0.2, since the $\mathtt{am}\,(2, 10)$ constraint imposes that one can experience only up to 2 misses in every sliding window of 10 controller iterations. On the contrary, in the second case (Figure 5) there is no upper bound to the actual deadline miss probability but on average it should be around 0.4. In the second case, however, the communication channel can experience only one message drop at a time, as $p_\beta = 1.0$ and the channel always recovers to the transmitting state after a faulty transmission. These experiments give us a sense of the faults, but it is hard, looking at the plots, to compare the effect of the different components.

Hence, we devised an experimental campaign aimed at providing a more comprehensive outlook on the effect of faults on the pendulum angle. We still perform 50 runs of the control system in different conditions. Rather than showing the pendulum angle, we calculate the percentage of time that the pendulum has spent in the proximity of the upright equilibrium, as the amount of time during each run in which the angle was in an interval of ± 0.05 from the upright equilibrium point. We then report a box plot of this quantity for the set of 50 runs, each lasting one minute. Figure 6 shows the results.

The figure is composed of four plots. In the first plot, $p_\alpha = 0.0$ and $p_\beta = 1.0$. This means that there is no communication failure, and both the sensor and the actuator values are retrieved and communicated correctly. The leftmost box plot shows the distribution of the percentage of time the pendulum spends upright over 50 runs in nominal conditions, i.e., in absence of deadline misses. The following boxes show the same values when deadlines can be missed according to the constraint indicated in the x-axis. Furthermore, the vertical lines separate the plots according to the strategy that is used to handle the miss (Kill and Hold, Kill and Zero, Skip and Hold, and Skip and Zero). Without channel failures, it is very easy to conclude that Kill and Hold is the best strategy in terms of the pendulum stabilisation.

On the contrary, Skip and Zero does not obtain satisfactory results, and the pendulum is often not controlled correctly, even for constraints that are fairly simple to handle. For the two remaining strategies, it is very obvious to see that Kill and Zero is worse that Skip and Hold. Skip and Hold is able to manage some of the constraints, with the pendulum falling often when 3 consecutive deadlines are missed and being fairly robust to other deadline-miss patterns. Generally, the Zero actuation strategy seems to be worse than the Hold alternative.

The following rows show what happen when the values

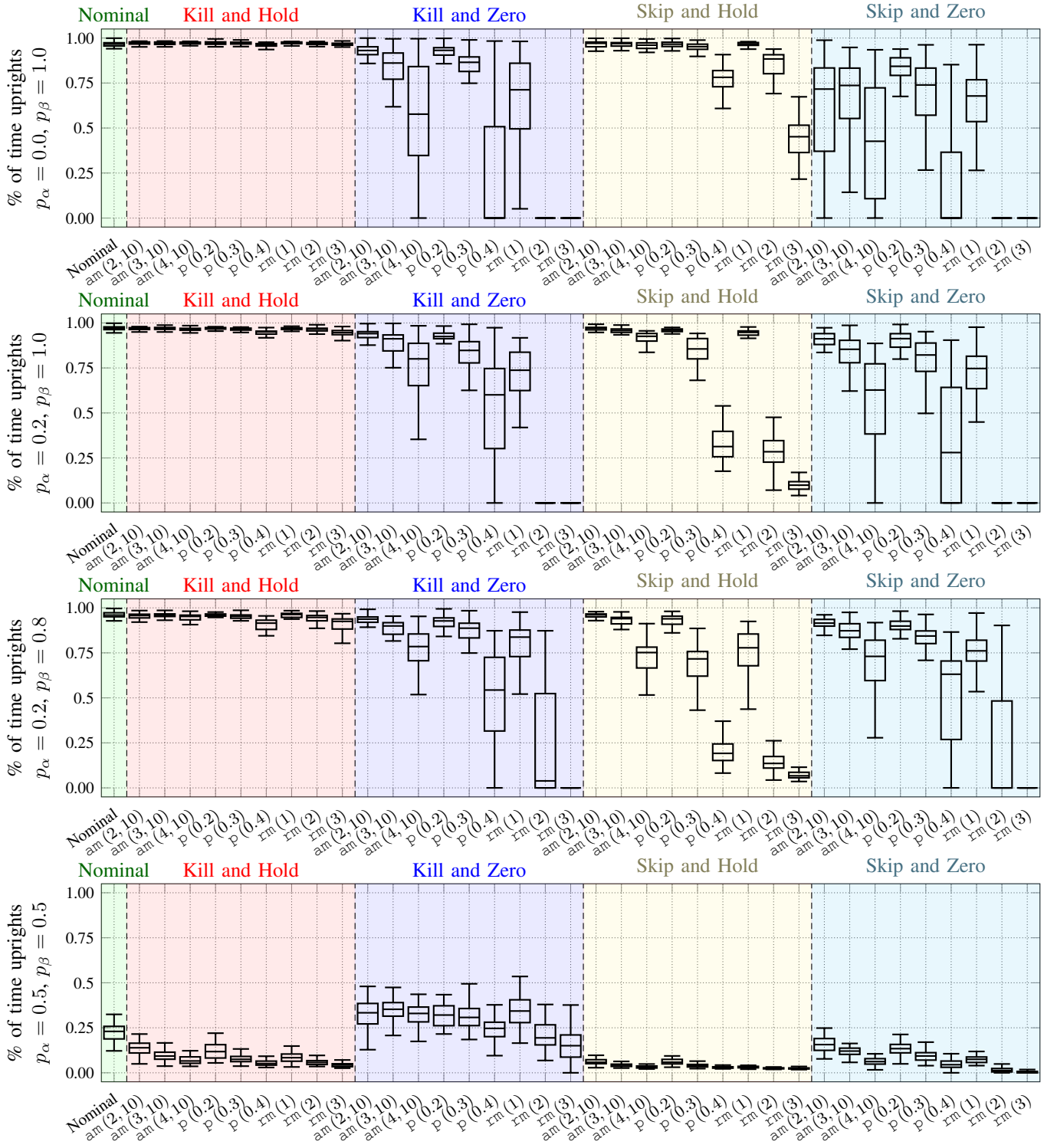

Fig. 6. Box plot showing the percentages of time spent in the upright position for different experiments with the Furuta pendulum. Each box plot summarises 50 experiments, each experiment lasts for a minute. The x-axis shows different types of constraints, from any-miss, to probabilistic, to row-miss constraints. The first box plot always shows the nominal situation in absence of deadline misses. The first row shows the case in which no channel failure occurs (the transmission channels start and remain in the good state, in which transmissions happen correctly). The other plots show the same results as the first one, with different channel models and probabilities.

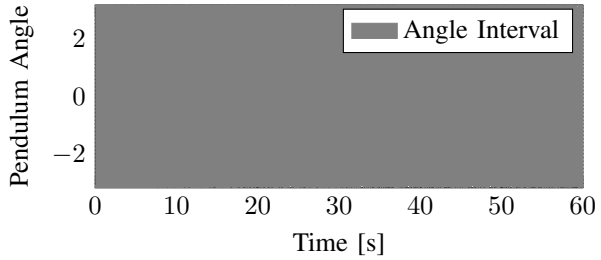

Fig. 7. Pendulum angle interval over 50 runs for $\texttt{rm}(1)$ with Kill and Zero, when $p_\alpha = 0.5$ and $p_\beta = 0.5$.

of p_α and p_β assume different values. The results without channel faults are consistent with the results obtained in the second and third rows, in which Kill and Hold stabilises the pendulum better than the other strategies.

The fourth row shows some other interesting results. The results are collected with $p_\alpha = 0.5$ and $p_\beta = 0.5$. This means that in nominal conditions (i.e., no computational faults that cause deadline misses) the chain between sensing and actuation is successful on average in 25% of the cases. This is because the probability that transmissions of sensor data and actuator commands on the channel are independent and hence the channel model is triggered twice. In this case, Kill and Zero seem to behave better even than the nominal conditions, which is slightly surprising, but seems to be mostly due to the pendulum spinning continuously and being found in the vicinity of the equilibrium more often because of moving uncontrollably. A plot of the interval of the pendulum angle over the 50 runs is in fact shown in Figure 7 and it is very obvious that the better behaviour shown in the box plot is not really much better than the other alternatives and the pendulum is not well-behaved.

IV. Conclusion

In this paper we discussed the implementation of a software framework to conduct experiments injecting faults in control systems. We have shown a long experimental campaign, lasting more than a week, in which a Furuta pendulum is controlled with different faults. We performed a statistical analysis on the results obtained from the experimental campaign and discussed what is the best strategy to mitigate faults due to computational problems.

References

[1] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis. An empirical survey-based study into industry practice in real-time systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 3–11, 2020.

[2] A. Aspeel, K. Rutledge, R. M. Jungers, B. Macq, and N. Özay. Optimal control for linear networked control systems with information transmission constraints. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 1960–1967, 2021.

[3] K. J. Åström and K. Furuta. Swinging up a pendulum by energy control. *Automatica*, 36(2):287–295, 2000.

[4] R. Atluri and Y. Kao. Sampled-data control of systems with widely varying time constants. *International Journal of Control*, 33(3):555–564, 1981.

[5] G. Bernat, A. Burns, and A. Llamosí. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50:308–321, 1999.

[6] A. Biondi and M. Di Natale. Achieving predictable multicore execution of automotive applications using the let paradigm. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 240–250, 2018.

[7] A. Biondi, M. D. Natale, and G. Buttazzo. Response-time analysis of engine control applications under fixed-priority scheduling. *IEEE Transactions on Computers*, 67(5):687–703, 2018.

[8] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Publishing Company, Incorporated, 3rd edition, 2011.

[9] A. Cervin. Analysis of overrun strategies in periodic control tasks. *IFAC Proceedings Volumes*, 38(1):219–224, 2005. 16th IFAC World Congress.

[10] S. K. Ghosh, S. Dey, D. Goswami, D. Mueller-Gritschneder, and S. Chakraborty. Design and validation of fault-tolerant embedded controllers. In J. Madsen and A. K. Coskun, editors, *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pages 1283–1288. IEEE, 2018.

[11] G. Hasslinger and O. Hohlfeld. The gilbert-elliott model for packet loss in real time services on the internet. In *14th GI/ITG Conference - Measurement, Modelling and Evalutation of Computer and Communication Systems*, pages 1–15, 2008.

[12] M. Hertneck, S. Linsenmayer, and F. Allgöwer. Efficient stability analysis approaches for nonlinear weakly-hard real-time control systems. *Automatica*, 133:109868, 2021.

[13] C. Hobbs, B. Ghosh, S. Xu, P. S. Duggirala, and S. Chakraborty. Safety analysis of embedded controllers under implementation platform timing uncertainties. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2022.

[14] R. M. Jungers, A. Kundu, and W. P. M. H. Heemels. Observability and controllability analysis of linear systems subject to data losses. *IEEE Transactions on Automatic Control*, 63(10):3361–3376, 2018.

[15] S. Linsenmayer and F. Allgower. Stabilization of networked control systems with weakly hard real-time dropout description. In *IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 4765–4770, 2017.

[16] S. Linsenmayer, B. W. Carabelli, S. Wildhagen, K. Rothermel, and F. Allgöwer. Controller and triggering mechanism co-design for control over time-slotted networks. *IEEE Transactions on Control of Network Systems*, 8(1):222–232, 2021.

[17] S. Linsenmayer, M. Hertneck, and F. Allgöwer. Linear weakly hard real-time control systems: Time- and event-triggered stabilization. *IEEE Transactions on Automatic Control*, 66(4):1932–1939, 2021.

[18] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein. Control-system stability under consecutive deadline misses constraints. In *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 21:1–21:24, 2020.

[19] P. Pazzaglia and M. Maggio. Characterising the effect of deadline misses on time-triggered task chains. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2022.

[20] P. Pazzaglia, Y. Sun, and M. D. Natale. Generalized weakly hard schedulability analysis for real-time periodic tasks. *ACM Trans. Embed. Comput. Syst.*, 20(1), dec 2020.

[21] M. Schinkel, W.-H. Chen, and A. Rantzer. Optimal control for systems with varying sampling rate. In *Proceedings of the 2002 American Control Conference*, volume 4, pages 2979–2984, 2002.

[22] G. von der Brüggen, N. Piatkowski, K.-H. Chen, J.-J. Chen, K. Morik, and B. B. Brandenburg. Efficiently approximating the worst-case deadline failure probability under edf. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 214–226, 2021.

[23] N. Vreman, A. Cervin, and M. Maggio. Stability and performance analysis of control systems subject to bursts of deadline misses. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 15:1–15:23, 2021.

[24] N. Vreman, R. Pates, and M. Maggio. Weaklyhard.jl: Scalable analysis of weakly-hard constraints. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 228–240, 2022.

[25] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. B. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Transaction on Embedded Computing Systems*, 7(3):36:1–36:53, 2008.