Available online at www.sciencedirect.com
ScienceDirect
Robotics and Autonomous Systems 00 (2023) 1–22
Journal Logo

Addressing Time Discrepancy between Digital and Physical Twins

Mirgita Frasheri[a,*], Henrik Ejersbo[a], Casper Thule[b], Cláudio Gomes[a], Jakob Levisen Kvistgaard[a], Peter Gorm Larsen[a], Lukas Esterle[a]

[a]*DIGIT, Department of Electrical and Computer Engineering, Aarhus University, Denmark*
[b]*Independent, Denmark*

Abstract

Digital twins (DTs) represent a key technology in the development, real-time monitoring and optimisation of cyber-physical systems (CPSs). Such potential emerges as a result of the real-time coupling between DTs and their physical counterparts, where it is possible to make use of operational data as it is being generated in order to aid decision-making. Harnessing this potential presents several design challenges, such as the parallel operation of the DT and its physical twin (PT), and the necessary synchronisation thereof, to ensure coherent execution of the system in ensemble. In this paper we present an approach that handles situations where a DT and its PT get out of sync as a result of disturbances in the normal operational conditions of the DT-PT system, *e.g.*, due to network degradation or temporary network drop. The purpose is to provide a best-effort functionality covering: user notification, degradation of DT to digital shadow (DS), with recovery mechanisms to re-establish the synchronisation between DT and PT.

Keywords: Cyber-Physical Systems, Digital Twins, Modelling, Time Delays, Synchronisation

1. Introduction

Cyber-Physical Systems (CPSs) operate in close interaction with the physical environment. As such, they are in general quite complex to develop, typically requiring expertise from multiple disciplines [1]. As a consequence a Model-Based Development (MBD) approach can be beneficial to master the complexity. Additionally, different parts of a CPS may depend on different mathematical basis, and are modelled using different tools, thus making MBD a natural choice for development. In order to evaluate the behaviour of a CPS as a whole, these models/solvers have to be coupled together into a joint simulation, for example co-simulation [2]. This coupling is made possible through standards that define (i) how models are packaged and (ii) the way functionality is exposed. One such standard is the Functional Mock-up Interface [3], a widely used approach, that is also adopted in this paper.

A fully modelled system, representing a digital replica of the physical system and allowing for bi-directional data exchange, is considered a digital twin (DT) while the real system is often referred to as a physical twin (PT). A DT can be used to continuously monitor a CPS [4]. Inside such a DT, a model can continuously be fed with data from sensors of the PT (the real CPS). The desired behaviour is that such a DT is able to predict when the PT no longer

*Corresponding author
Email addresses: mirgita.frasheri@ece.au.dk (Mirgita Frasheri), hejersbo@ece.au.dk (Henrik Ejersbo), casper.thule@outlook.com (Casper Thule), claudio.gomes@ece.au.dk (Cláudio Gomes), jkv@ece.au.dk (Jakob Levisen Kvistgaard), pgl@ece.au.dk (Peter Gorm Larsen), lukas.esterle@ece.au.dk (Lukas Esterle)

operates as expected. However promising, there are a number of challenges with this approach as there are inherent discrepancies between the data measured on the PT and the data processed on the DT. Two reasons for this are (i) data being discretized when converted from analogue to digital signals and (ii) potential delays between measurement and arrival of this data at the DT due to network disturbances. Specifically, when utilising the DT to make decisions and give feedback to an (semi-)autonomously operating PT receiving, the timeliness of data can be critical. As an example, consider the case of network control systems, where stability is threatened by the introduction of delays [5], or a case in which a DT will warn the PT of obstacles ahead. Any delay in this warning and it may become too late to avoid the obstacle. These call for approaches that allow to deal with such time delays between PT and DT.

In this work, we aim to address the issues of time delays and synchronisation errors at the DT level. Specifically we aim to answer the following research questions:

1. With a DT running in parallel to a PT, can approaches that detect time discrepancies between a DT and its PT be devised, serving as enablers for a DT that is sensitive and can adapt to the delays involved?
2. When discrepancies are detected, can we degrade the DT (allowing for bi-directional information exchange) to a digital shadow (DS), which only receives data from the PT but does not provide feedback (to said PT)?
3. After degradation to a DS, can we achieve re-synchronization between the DS and PT, effectively allowing to upgrade the DS back to a DT? Finally, can we enable developers and operators to tune the time it may take to re-synchronize?

The main contribution of this work is a novel approach for autonomous robots, governed by DTs, to deal with network delays when receiving data to ensure safe and continuous operation. To orchestrate different functional mock-up units (FMUs), we utilise the Maestro engine [6]. We further employ a RabbitMQ solution based on the AMQP protocol [7] for data brokerage between the PT and the DT and present our orchestration synchronisation administration (OSA) approach, for detecting time discrepancies between PT and DT and mitigating when out-of-sync conditions.

The remainder of this paper is structured as follows. The next section gives more details on the areas of co-simulation, the utilised AMQP data brokerage, called RMQFMU, and its implementation, different time synchronisation techniques, and approaches for simulating hardware-in-the-loop. Section 3 gives a formal description of the problem we are tackling in this article. Our approach to deal with time synchronisation problems with DTs and their potential mitigation is presented in Section 4. We evaluate this approach through a real-world mobile robotic system called Desktop Robotti. The experimental setup and the achieved outcomes to reduce time discrepancies are discussed in Section 5. Section 6 gives an overview of related work. Finally, Section 7 concludes the article, discusses future work, and outlines a road-map to achieve the identified open challenges.

2. Background

This section introduces co-simulation and the Functional Mock-up Interface 2.0 (FMI2) standard for co-simulation. Furthermore, it presents the enabling tools Maestro2 and RMQFMU used in the approach. Maestro2 and RMQFMU are open-source and part of the tool chain of the non-for-profit INTO-CPS Association[1] [8]. The vision of INTO-CPS Association is to offer a tool-chain that enables MBD support for the engineering of CPSs and DTs.

2.1. Co-simulation concepts and tools

In order to simulate a CPS consisting of different constituents represented by different simulation units, one can apply collaborative/coupled simulation (co-simulation) [9]. Co-simulation is a technique that brings together the individual simulation units of the CPS into a global simulation of the CPS. This is achieved through coordinated time progression and exchange of values between the constituents. Thus, a co-simulation typically consists of a repetitive procedure of setting inputs on the simulation units, making them progress in time, and retrieving their outputs. For an introduction and survey on the topic, please refer to [10].

In order to compose simulation units in a generalised fashion it is necessary that they adhere to a standard. The contribution that we propose is based on FMI2 [11, 12]. The individual simulation units of an FMI-based co-simulation

[1] Available at `https://into-cps.org` - Visited June 22, 2021.

are referred to as FMUs. Such an FMU shall implement a set of c-interfaces according to the standard, provide a static description file describing the given simulation unit, and be packaged in a certain fashion.

The realisation of a co-simulation and thereby a composition of FMUs is carried out by an co-orchestration engine (COE) employing an orchestration algorithm (OA). The OA describes (i) the order of getting outputs from the FMUs, through `getXXX`-functions, (ii) making the FMUs progress in time, *i.e.*, requesting the FMU to complete an iteration that takes it from time h to $h + \Delta h$, through the `doStep`-function, and (iii) setting inputs on the FMUs, through `setXXX`-functions, along with various extensions. One such extension is the non-standardised function `getMaxStepSize` [13], which the OA can use to get the maximum step size of the subsequent `doStep` of all FMUs implementing the `getMaxStepSize`-function. Applying this extension makes it possible to calculate the largest step agreed upon by all FMUs and thereby achieve variable time step size and, in some cases, speed up the co-simulation. This is considered below in Section 4. A typical OA is the Jacobi approach [14], which in a simulation loop iteration sets all inputs at simulated time h, prompts the FMUs to perform `doStep` with simulated time step size Δh, such that they progress to time $h + \Delta h$ and finally collects the outputs of the FMUs at simulated time $h + \Delta h$ before iterating again until a given simulated end-time h_{end} is reached. The orchestration engine used for the contribution of this paper is Maestro2[2] [15], henceforth referred to as Maestro, as it implements the Jacobi approach and can be extended through plugins. Maestro and the surrounding tool-chain is a product of the INTO-CPS project, the IP of which was transferred to the INTO-CPS association at the end of the project's run [16]. Further development of these tools, as well as maintenance and documentation is actively on-going, hence we adopt such tool-chain in our research in general, as well as specifically in this paper.

2.2. RMQFMU: a Bridge to the Real World

The RMQFMU, first proposed in [17] and later extended in [18], enables the development of advanced monitors by focusing on the problem of brokering external data from a deployed system so that it can be used in a co-simulation, in real-time. RMQFMU is a RabbitMQ client library[3] wrapped as an FMU and thus enabling data brokerage in the context of a co-simulation. In addition, the FMU also enables publishing data back to the external system, thus closing the communication loop, which is relevant when used in a DT context.

As a facilitating data broker it is important that message delays introduced by the FMU are minimal, and hence impacting an overall co-simulation environment as little as possible. To this extent the RMQFMU enables a multi-threaded configuration, with a separate client thread consuming and parsing data from the RabbitMQ server, potentially in parallel with the co-simulation stepping this and other FMUs.

In order to progress in time, the RMQFMU needs valid data, *i.e.*, data with a timestamp that matches the time to which the RMQFMU has been requested to step to. The RMQFMU can be configured with a *maxage* parameter, among others, that defines the age of data for which a message is still considered valid, and thus outputted at a given time-step. Assume the RMQFMU is attempting to reach time $h+\Delta h$, with a given *maxage*. In this condition a message is valid if its timestamp T_{msg} satisfies the following condition:

$$h + \Delta h - maxage \leq T_{msg} \leq h + \Delta h. \tag{1}$$

Note that, at any given time, there may be more than one message in the RMQFMU's internal queue that satisfies the above condition. In such a case, only the most recent message will be picked and outputted.

The fundamental steps of the RMQFMU involved in a `doStep` operation, with step size Δh, are the following:

1. The RMQFMU will access the RabbitMQ queue of the current environment state until it is able to read a message or until a timeout value is reached. In the case of a timeout, the RMQFMU exits. When accessing the RabbitMQ queue, the RMQFMU executes the following:
   (a) Read n (specified by the user) messages from the RabbitMQ queue, merge them with the FMU's internal queue and sort the result.
   (b) Check if the resulting sequence can be used to produce an output at time $h + \Delta h$ that satisfies the *maxage*, *i.e.*, fulfils equation 1. If so, then the data broker will not block. Otherwise it blocks and waits for more messages from the RabbitMQ queue.

[2] Maestro2 is the successor of Maestro1. Maestro2 provides the features of Maestro1 and more.
[3] Specifically librabbitmq-c, that can be found at `https://github.com/alanxz/rabbitmq-c`

2. If not blocked, then the RMQFMU will:
   (a) update the internal time: $h \leftarrow h + \Delta h$;
   (b) set the output to the value of the most recent message satisfying Equation 1;
   (c) drop the internal messages with timestamps in the past.

It could happen that different messages have different timing requirements. In our setup this is addressed by having as many RMQFMUs as the different timing requirements for the messages. For each time requirement, the corresponding RMQFMU would be configured with the adequate *maxage*.

2.3. Time Synchronisation Techniques

Time synchronisation is not only a fundamental problem for DTs, but for other domains such as discrete event simulation (DES) and parallel discrete event simulation (PDES) as well. Classical synchronisation approaches for PDES [19] typically fall in one of two categories, namely conservative and optimistic approaches [20, 21] (the curious reader is referred to [22] for a comparative study). While discrete event simulation is adopted in those cases where systems cannot be represented by continuous models, parallel discrete event simulation distributes the execution of such simulation across several threads with the goal of improving simulation speed [23]. Being able to run such simulations in parallel is rather helpful when they are coupled to actual hardware, and timing consistency becomes of importance.

Conservative approaches are such that no sub-system can advance to the next step unless there is a guarantee that no other event will arrive between the current step and the next, *i.e.*, such methods require some lookahead in terms of future events, but no rollback capability is required [20]. These methods can be either synchronous or asynchronous [24], where the former requires global synchronisation, whereas the latter does not. Common examples of conservative approaches are: null message [25, 26], bounded lag [27], time buckets [28], YAWNS [29], and composite synchronisation [30].

In general, the type of models, discrete or continuous (see [10] for more details), and the manner in which it is advanced in time, *i.e.*, either with uniform or variable step-size, will shape the actual implementations of the conservative techniques [31]. Specifically, for continuous models, the reader is referred to the Gauss-Seidel (serial) and Jacobi (parallel) approaches [14], whereas for discrete models the reader is pointed to master-slave, time-stepped, and global event driven, among others [32, 33].

Optimistic approaches on the other hand allow the different entities to progress in time without waiting for one another, until a conflict is detected and the offending entities are required to rollback – no matter how far back [34] – undo the actions taken and repeat the execution of those steps. Consequently, they require rollback capabilities, but no lookahead for future events is needed. An example implementation is the Time Warp mechanism [34]. This method relies on the concept of virtual time, and is similar to betting that no message will be received with an older timestamp than the one that is currently being processed. Modifications of Time Warp have been proposed, *e.g.*, to allow for optimistic execution in the cloud, where the loads among different components are asymmetric [35].

Recent research has focused on bringing together these two classes of methods by enabling dynamic switching between them [20]. This means that, if lookahead is available, then it is used to step conservatively to the next cycles. If no lookahead is available, then it leads to the assumption that it is safe to step and potential conflicts are resolved as they occur. Previous research has also considered the combination of both approaches, however in a static setting where it is not possible to switch between the two (see [36] for an example of the latter, and [20] for a comprehensive overview).

It is noted that for the DTs discussed in this paper, conservative approaches are typically implemented, specifically either the Gauss-Seidel or Jacobi algorithm. Thus, there are clear parallels to the work in DES and PDES domains. Nevertheless, there are important differences to note, as DTs are expected to run alongside their PTs, with the system as a whole operating safely. While simulations in the classical sense are not designed to run alongside hardware and potentially control or reconfigure said hardware during runtime.

2.4. Hardware-in-the-Loop Simulation

DTs are closer to hardware-in-the-loop simulations (HILs) due to the coupling of the simulation components to hardware. As such, considerations for HIL are also relevant in the context of DTs. HIL simulation is a technique that has been used for decades to test systems for which testing in their actual operation setting is too costly, *e.g.*, aircraft

and satellite control systems [37], seismic studies [38], among others. HIL simulation requires the coupling of the hardware to a real-time simulation, requiring that the simulation time follow the wall-clock time [39]. In this case the simulation needs to represent reality in an acceptable way, and the simulation is generally only validated for a set of scenarios. A critical parameter for HIL is the execution time of a single iteration of the simulation loop [37]. In order for the simulation to be accurate this value needs to be as small as possible, however long enough to allow for the completion of the different operations, *e.g.*, I/O; in other words the logic has to be processed within one time-step of the simulation, considering also the communication to the rest of the system [40]. Rankin and Jiang have proposed a verification and validation framework based on HIL simulation, where the interface hardware is external to the simulator, however it is the latter that has control over the variables [41]. A recent contribution proposes to use virtual hardware-in-the-loop instead, where most of the real-time conditions can be lifted [39].

In cases where non real-time simulations need to be coupled to the actual hardware, additional synchronisation issues emerge [42]. Classical approaches to address such issues are (i) the lock-step, in which all subsystems advance in time only when the slowest subsystem is ready to go the next step, and (ii) time-slicing, where the simulations are run for a period of time, with the hardware frozen; once the outputs are available the hardware is run for a period of time. For the latter approach, in case any discrepancies are observed, then time is rolled back to a previous point where the data was consistent. However, such an approach might not necessarily be adequate in the case of a DT that runs in real-time alongside its PT, *i.e.*, it might not be possible to perform a rollback. It is worth to note that there might be cases where the DT is able to simulate ahead in time, as compared to the PT, with rollback to the time that the PT is in being triggered by a signal from the PT itself, or even an operator.

While there are clear similarities, mostly in terms of the coupling between simulation and hardware, DTs mainly differ from HIL in that a DT is intended to be deployed alongside its PT and perform different tasks from monitoring, to reconfiguration, to sending control commands if deemed necessary.

3. Problem Description

A DT is capable of getting real-time data from its PT, as well as send control or other data to said PT if necessary. In order for such operation to be useful, the twins should be synchronised when they exchange information with one another. Should they become out-of-sync due to a disruption during operation as a result of unforeseen changes in the environment or other failures, such as a result of a time delay attack [43] then the digital twin should be able to identify such disruptions. Furthermore, the DT should take actions in order to mitigate consequences of the DT sending control data to the PT once the two are out-of-sync.

We assume that the wall-clock times (WCTs) of the DT and PT are synchronised via a clock synchronisation service (see [44]), *e.g.*, NTP, and define the time difference as a function between simulated time and wall-clock time:

$$t_{diff}(h, t_{wc}) = s2w(s) - t_{wc} \tag{2}$$

where h is simulation time and t_{wc} is the wall-clock time, and the function $s2w$ maps simulation time to wall-clock time.

We assume that in normal conditions the DT is able to keep up with the PT, *i.e.*,

$$t_{diff}(h, t_{wc}) \leq \theta_{safe}, \tag{3}$$

where θ_{safe} is a predefined threshold used to indicate the tolerated time difference between systems. If $t_{diff}(h, t_{wc}) > \theta_{safe}$ then the DT and PT are considered out-of-sync. Note that, real-time requirements depend on the actual application, *e.g.*, for human robot interaction these range between $100 - 300$ms, whereas for other HIL systems there are stricter requirements up to 10ms [23]. For the calculation of θ_{safe}, we can refer to the maximum allowed delay, computed in stability analysis of network control systems. It is well known that instability arises from delayed control commands, based on delayed sensory information. The interested reader is referred to [45] for a survey on networked control systems with communication delays. Normal conditions refer to expected values for the network delays d_{exp} between DT and PT, specifically d_{exp} refers to the one-way delay, presumed symmetric in both directions, and the computational costs of the calculations performed within the DT. This means that such factors shall be evaluated at system setup, and the DT shall be configured to cover for expected values of network and computational latency. In

some cases, this can be resolved by applying a larger time-step size to the DT. Nevertheless, such an approach does not always guarantee a faster simulation, as each FMU could potentially decide on the internal solver step size on its own, regardless of the communication step size given [46].

As mentioned, the system can be subjected to disruptions during its operation, which can result in the DT and PT becoming out-of-sync. These disruptions could be temporary or long-term/permanent. In this paper we exemplify such disruptions with temporary network degradation and temporary communication loss, with the assumption that the WCT clock synchronization assumption (made for Equation 2) is unaffected, as it is considered out of scope for this paper. This is a reasonable assumption because wall-clock synchronisation schemes require far less bandwidth and speed than typical DT applications. As such it would not be impacted during network degradation. In regards to the network drop case, where the communication is cut for some longer period of time t_{drop}, we additionally assume that t_{drop} is lower than the time it would take for the wall-clocks on the DT and PT to drift from each other.

Additionally, note the difference between the safety threshold (θ_{safe}) and the constraint on the age of data (*maxage*). The former indicates how far behind the simulation can fall before the DT should degrade to a DS, whereas the latter indicates the age of the data with which the RMQFMU can proceed to the next step when no other data is available. Note however, that a high *maxage*, would undermine the calculation of t_{diff}, since the co-simulation would be able to progress, thus keeping up with the wall-clock time, while still operating on older data that is considered valid. As such a highly sensitive application would require a small *maxage*, as opposed to cases where operation with older data can be tolerated.

In the following, we illustrate two common scenarios regarding how the DT is affected by network issues.

Temporary network degradation. Assume that due to some environmental issues, the expected network delay d_{exp} suddenly increases by some factor γ, *i.e.*, $d(t) = d_{exp} + \gamma(t)$, where $d(t)$ is the network delay experienced at time t (Figure 1a). As an example consider the case where 4G degrades to a 2G network. Additionally, assume that $\theta_{safe} = 1$ time-unit for the sake of the example, where the time-unit represents the expected wall-clock time duration for one step. As seen in Figure 1a, during normal conditions, a message sent from the PT to the DT at time t_1 will arrive between h_0 and h_1, a time for which $t_{diff}(h_1, t_1) \leq \theta_{safe}$ holds. Thereafter the DT will reach simulation time h_1, and will be able to send its reply in a timely fashion to the PT. At this point the DT is ready to attempt h_2. However during network degradation, which starts at t_2, a message sent from the PT at time t_2 will arrive with some delay γ on the DT (for the sake of the example $d_{exp} + \gamma = 2$ time-units, with γ remaining constant). Given an age constraint (*maxage*) on data equal to 1 time-unit, the DT has advanced from h_1 to h_2 with the same data available at h_1 (as in the example in Figure 1a). At h_2 there is initially no new data – given the unexpected delay – and the last data is not valid any longer due to the age constraint. Thus, the DT will wait for valid data, hanging at h_2 until the timeout (the RMQFMU timeout) is reached. This waiting mode on the DT comes as a result of the wait in the RMQFMU until valid data is available. Other FMUs, and the DT as a whole cannot continue to the next simulation step until the RMQFMU outputs such data. Meanwhile, the wall-clock is advancing, and assuming the data sent by the PT at t_2 is not lost, it will eventually arrive on the DT at t_4. In this case $t_{diff}(h_2, t_4) \leq \theta_{safe}$ does not hold any longer, as the difference between $s2w(h_2)$ and t_4 is bigger than the threshold. Additionally, any message from the DT – sent after h_3 is reached – will arrive after approximately the same delay on the PT. The consequence of this is that a message that is a response to the state of the PT at time t_2 arrives at time t_6, and at that time the content of said message may be obsolete, and should not be considered by the PT. While it may be sensible to implement mechanisms in the PT that can allow it to discard messages that come too late (thereby mitigating delays directly at the PT level), this might not be possible in situations where the CPS is retrofitted with a DT. Nevertheless, without additional preventive mechanisms, the DT will keep sending messages that are bound to be too late, meaning that the DT (i) is oblivious to the current situation in which it cannot follow the PT, and that (ii) it is burdening the connection with meaningless messages. The latter can be especially troublesome when there is a connection with limited bandwidth, where each message can create additional queuing delay.

Temporary network drop. Assume that the network connection drops temporarily, as can happen for example with WiFi connections. During this time, the PT continues to send messages to the DT, and advances in time, whilst the DT will remain fixed at the latest time-point for which it has valid data, waiting till a timeout value is reached. Assuming the connection is re-established before the timeout is reached, the messages will come in bursts (Figure 1b). While the DT is at time h_1, the PT has continued to time t_5. The DT could consider each message and step progressively,

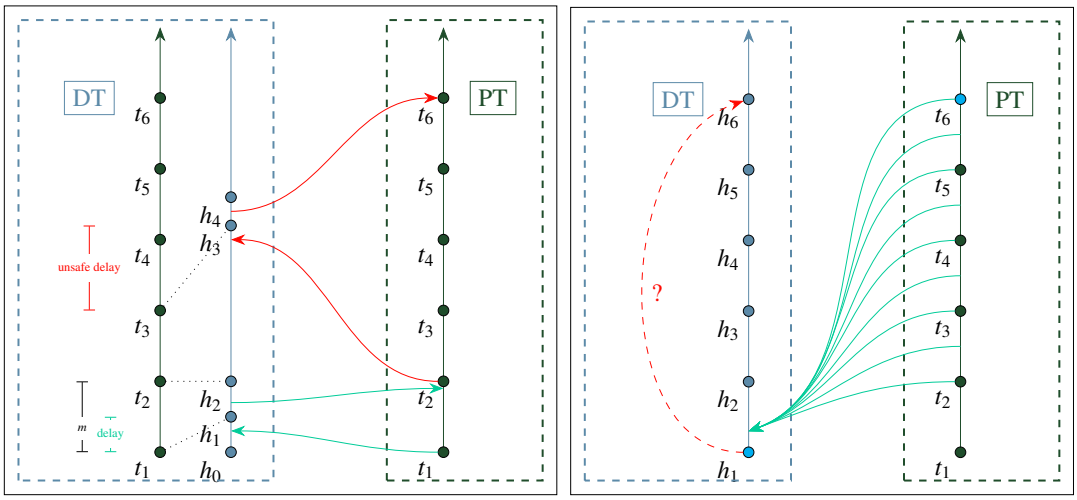


Figure 1: a) Network degradation during operation, resulting in increased network delay. Green arrows indicate transmission of messages within the expected delays, whereas the red arrows indicate abnormal conditions with increased network delay. Axes notated with t_i represent the wall-clock time, assumed to be shared by both DT and PT, m refers to the age constraint. The dotted lines show the relation between DT and PT time. b) Incoming burst of messages after network connection is re-established – the current time-step (simulation and wall-clock time) is represented as a circle filled in cyan. The dashed red-line notates a possible attempt to take a bigger step-size to go directly from h_1 to h_6, as opposed to taking the steps one by one, i.e. h_1, h_2, ..., h_6.

whilst the PT advances even further, or it could try to jump ahead in time by some given jump size that defines, in the context of this paper, how many messages can be skipped. The preferred choice in this case would depend on how the data is used, *e.g.*, (i) monitors that care only for the latest state, or (ii) monitors that care for the whole history. Independently of this, the DT is in a situation where it is behind in time, and thus should attempt to catch up as fast as it can, while not sending meaningless messages to the PT.

In both these conditions, the DT has to be 'aware' of the time difference between itself and its PT, in order to (i) notify an operator and keep them updated, (ii) disable the transmission of control commands to the PT, and (iii) attempt to jump ahead to the PT time if possible. In the next section we propose a set of mechanisms with which our DT platform can be extended such that it adapts accordingly and safely in these situations, and attempts mitigation strategies when possible.

4. Orchestration Synchronisation Administration - The OSA Approach

In this paper we propose a Maestro-based mechanism to detect and potentially resolve out-of-sync situations that may occur. Specifically, our approach enables:

1. The detection of the time difference $t_{diff}(h, t_{wc})$ between the DT and its PT, and thereafter the notification of the user/operator supervising the DT.
2. Suspending and re-enabling the communication link from the DT to the PT, depending on whether the DT and PT are out-of-sync or in-sync, respectively.
3. The specification of a mitigation strategy to be adopted when out-of-sync.

The first two functions are realised through the implementation of a time discrepancy detector FMU, namely OSAFMU, that sets its Boolean output to *True* if the DT and PT are out-of-sync, and to *False* otherwise, and its integer output to the observed time discrepancy in milliseconds. Such output is used by the RMQFMU to disable or enable, respectively, the sending of data to external components such as the PT. The third functionality is supported by Maestro, which calls the `getMaxStepSize` function on each FMU, when an out-of-sync situation is detected, and asks the FMUs to take bigger step sizes if feasible. By taking bigger step-sizes, the co-simulation can progress faster, thus reducing the gap between simulation and wall-clock time.

Time Discrepancy Detection. The OSAFMU adheres to the FMI2 standard, and as such its behaviour is encapsulated within an FMU that can be included in any FMI2-based co-simulation. It can be parametrised through the θ_{safe} and check interval Δ parameters, the former is defined in Section 3, whereas the latter specifies the interval used to check the time discrepancy. The FMI interface exposes two outputs, (i) the out-of-sync flag f_{oos}, used to notify the other FMUs that the DT and PT are not synchronised, and (ii) the actual time discrepancy t_D, that could be used by other FMUs. Additionally, the FMU has a background thread that can interact with a user/operator through another interface not bound to the FMI standard. Consequently, said user can be notified of an out of synchronisation event immediately, *i.e.*, not only at simulation step boundaries.

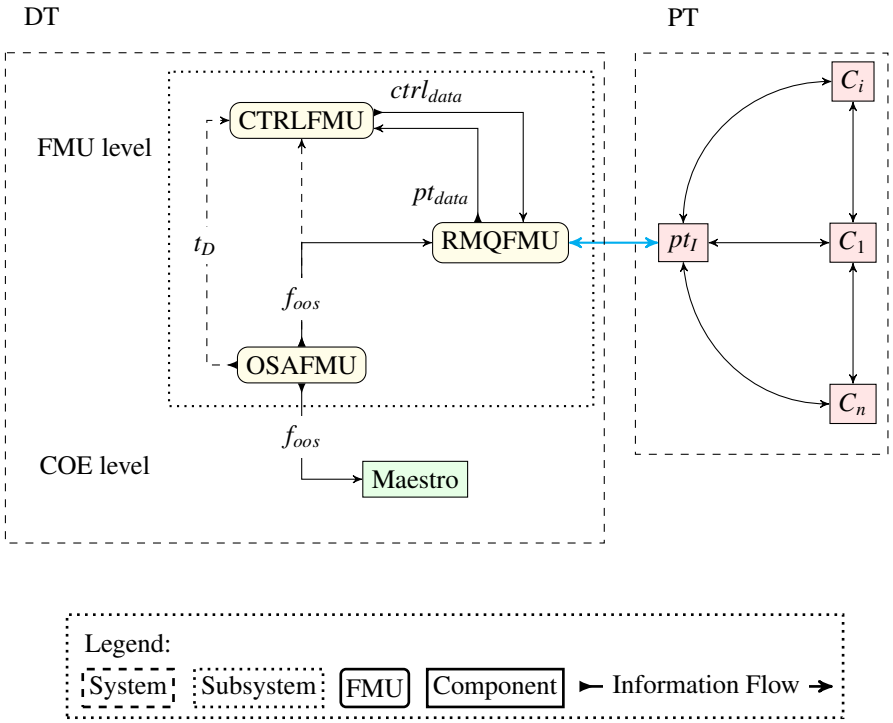


Figure 2: The DT composed of the RMQFMU, CTRLFMU, and OSAFMU, where the RMQFMU gets the data from the PT, and outputs it to the CTRLFMU (pt_{data}), which in turn sends control data to the PT through the RMQFMU ($ctrl_{data}$). The OSAFMU can set the out-of-sync flag, propagated to Maestro and RMQFMU in this case, and potentially to any other FMUs that need it. The PT is assumed to have an interface component pt_I and a set of other components $C_1 ... C_n$.

Assume a system comprised of the PT and its DT, where the DT consists of a set of FMUs (Figure 2) including (i) the RMQFMU that realises the data brokering between the DT and PT, (ii) the OSAFMU, and an arbitrary control FMU (CTRLFMU) able to send control data to the PT. The behaviour of the OSAFMU and the detailed interaction between the components of the DT is captured in Figure 3. Assume that the given system, composed of the PT, RMQFMU, OSAFMU, CTRLFMU and Maestro, has been successfully initialised. At time t_i the PT sends a message with timestamp $t_{PT}(i) = t_i$ to the DT. The message will arrive on the DT side, either before, after, or during a `doStep` call of the RMQFMU of a given simulation loop iteration. In the latter case, the message might not be processed yet, *e.g.*, the RMQFMU could be setting its outputs with the data it had under its disposal before message $t_{PT}(i)$ arrived. In other cases, this message will be processed and if it is valid with respect to the rules enforced by the RMQFMU, it will be available to other FMUs after the subsequent `doStep`. Maestro, as co-orchestration engine, coordinates the execution of the co-simulation and asks FMUs to perform their steps. The background thread in the OSAFMU continuously calculates the time difference by evaluating its own wall-clock (t_{dtpt}) against the current simulation time (h_{dt}). Should this difference be above a predefined threshold (θ_{safe}) then the OSAFMU immediately notifies the operator. This is done to ensure that the user gets the information as soon as possible, and does not have to wait for the completion of the `doStep` by every FMU. Within its `doStep`, the OSAFMU keeps track of the simulation time, and

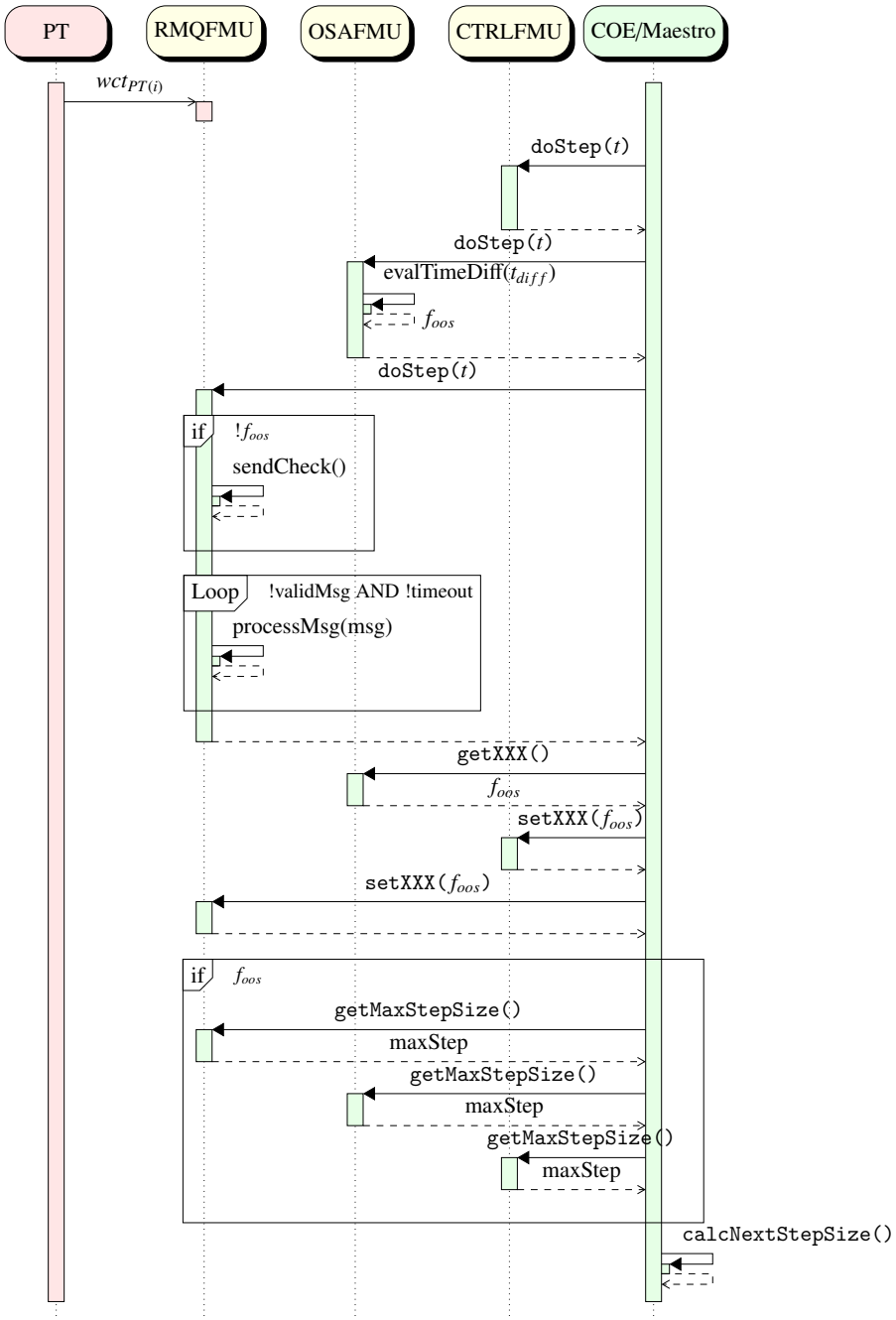


Figure 3: Sequence diagram of the PT and DT execution. The dotted arrows indicate that the transfer of data is not done directly, but through the COE/Maestro via `getXXX()` and `setXXX()`.

sets its outputs based on the information it has in the current step. If an out-of-sync condition has been identified by the OSAFMU thread, then the out-of-sync flag is set to 1. This can be used by any number of FMUs in the system, and is decided by the user during co-simulation configuration which FMUs will get this value. On the other hand, the RMQFMU is required to have this information at all times, such that it can stop sending messages to the PT when the out-of-sync condition is met; the `sendCheck` function checks whether there are inputs that should be sent to the PT at every time-step. Maestro uses the out-of-sync flag as well to decide whether a mitigation strategy is set in motion.

In this paper we consider a mitigation strategy that attempts to make the FMUs take bigger step-sizes until synchronisation is re-established, specifically through the `getMaxStepSize` function call. Note that, `getMaxStepSize` is not part of the FMI standard, and is a non-standardised addition that allows more flexibility in stepping the FMUs in a co-simulation. In case this function is not supported by the orchestration algorithm, the OSAFMU can still be used to provide its other functionalities, *i.e.*, (i) identify out-of-sync and notify user, thus enabling (ii) the cut of the DT to PT link until both are synchronised again. After each FMU returns the biggest step size it can take, Maestro will set the next step size as the minimum of these values. At every `doStep`, the OSAFMU will continue to check the time difference. In case the time difference goes below the safety threshold, then the outofsync flag is set to 0, and other FMUs are notified that it is allowed to send messages back to the PT. The user is also notified of such occurrence.

Tackling Network Degradation. During network degradation, the messages start coming at a slower rate than initially expected. As a result, the DT progresses in time at a lower real-time rate due to RMQFMU waiting for messages as described in Section 2.2. Even so, the DT can still progress its simulation time with older data, if the latter is still valid according to the maxage parameter, *i.e.*, $s2w(h_{dt}) \leq t_1 + maxage$, where t_1 is the time-stamp of the latest data available in the DT. The OSAFMU component is always running in the background and will notify the user as soon as $t_{diff}(h, t_{wc}) > \theta_{safe}$. Should that be the case, the following `doStep` call will result in the f_{oos} flag being set. The f_{oos} flag is transferred to the other FMUs at the end of the simulation loop iteration. When RMQFMU receives the f_{oos} flag it will stop sending messages to the PT, until the f_{oos} is unset. Note that, if there is no valid data for the RMQFMU, there is no way to take a bigger step-size, because there is no message from the PT to synchronise to. What can be done in this case is simply to degrade to DS functionality, and notify the user that the DT and PT are out-of-sync.

Tackling Message Bursts. In this case, the PT continuously sends messages at times $[t_1..t_5]$ to the DT. Assume that around time t_1 the WiFi connection dropped, then the co-simulation could only progress as long as the last received message is valid, *i.e.*, $s2w(h_{dt}) \leq t_1 + maxage$. Thereafter, the co-simulation would stall at the simulation time corresponding to time $t_1 + maxage$, until the timeout value of the RMQFMU is reached. Should this happen, the RMQFMU will exit and the co-simulation will shutdown. In the following we assume that the WiFi is reconnected before the timeout condition is met. Note that the OSAFMU is running in the background, and as soon as the time difference is above a given threshold, it will notify the user, and prepare the flag to be set on the next `doStep` call. Once the connection is re-established, the messages will be delivered to the RMQFMU queue, and the RMQFMU will be able to go to the next step. In this case, when Maestro calls a `getMaxStepSize` on each FMU, the RMQFMU will report a step size that reflects the latest message in the queue. After getting the maximum step-size from each FMU, Maestro will estimate the size of the next step, and enter the next loop iteration. This process will continue until the systems are back in sync, where the out-of-sync flag would be set to 0, and the full DT capability would be re-enabled.

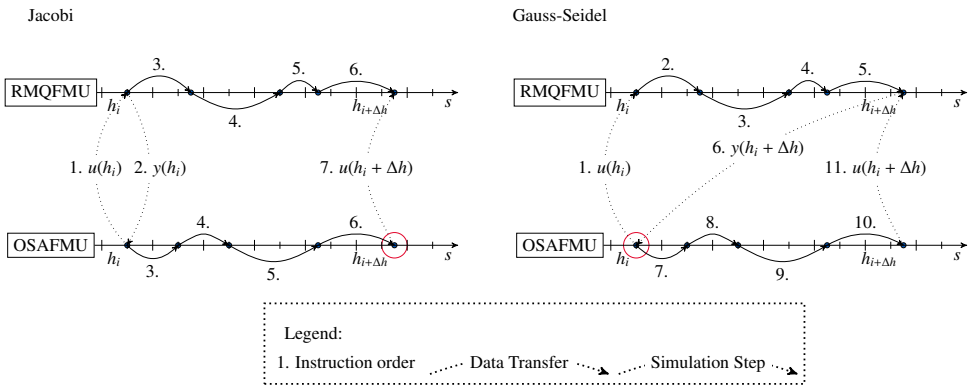


Figure 4: Jacobi (left) and Gauss-Seidel (right) approaches. Marked with red circles are the time-points for which the out-of-sync flag set at time h_i from OSAFMU is propagated to RMQFMU, where the step-size is equal to H. The figure is adapted from [47]. $u(h_i)$ represents the set input to the FMU at time h_i; $y(h_i)$ represents the output gotten from the FMU at time h_i.

Jacobi vs. Gauss-Seidel orchestration. Let us assume the system is at a time $h_{i-\Delta h}$ and attempting to reach step h_i, when there is lack of data. As aforementioned, the RMQFMU will wait for a timeout value before it exits. In a Jacobian setting (Figure 4 on the left), the other FMUs might have completed their `doStep` by this point. In such a case the OSAFMU would complete its step, effectively reaching h_i immediately while the safety threshold is still not violated. The moment the RMQFMU gets data, it will complete its `doStep`, reach the time h_i, and output the state of its buffer. Thereafter, the co-simulation will attempt to reach time $h_{i+\Delta h}$. By this time, the OSAFMU will be able to detect the discrepancy, as it reaches time $h_{i+\Delta h}$ (arrows $3-6$). Assuming the flow of information continues normally, the RMQFMU will also reach $h_{i+\Delta h}$ in a timely manner (arrows $3-6$). At time $h_{i+\Delta h}$, the set out-of-sync flag will be received on the RMQFMU end (arrow 7). The mitigation strategy, if possible, will be carried out just before calling `doStep` for $h_{i+2\Delta h}$. Similarly, the f_{oos} flag will be available to the RMQFMU as input for $h_{i+2\Delta h}$. In a Gauss-Seidel setting (Figure 4 on the right), the RMQFMU will perform its step first after the flow of data is reinstated, reaching h_i. Thereafter, the OSAFMU will attempt to reach h_i, and will be able to detect that the safety threshold has been violated. As a result, the f_{oos} flag will be available to the RMQFMU as input already at time h_i (arrow 1) before it attempts to reach $h_{i+\Delta h}$ (arrows $2-5$). In this case, the mitigation strategy, if applicable, will be carried out when the RMQFMU is attempting to reach $h_{i+\Delta h}$ (arrows $2-5$), followed by the OSAFMU (arrows $7-10$).

5. Evaluation

In this section we describe our case-study, and detail the setup of our experiments in which we replicate the network conditions already described in Section 3, *i.e.*, network degradation, and temporary network drop. Thereafter, we present the results and discuss the suitability of our approach in DT platforms.

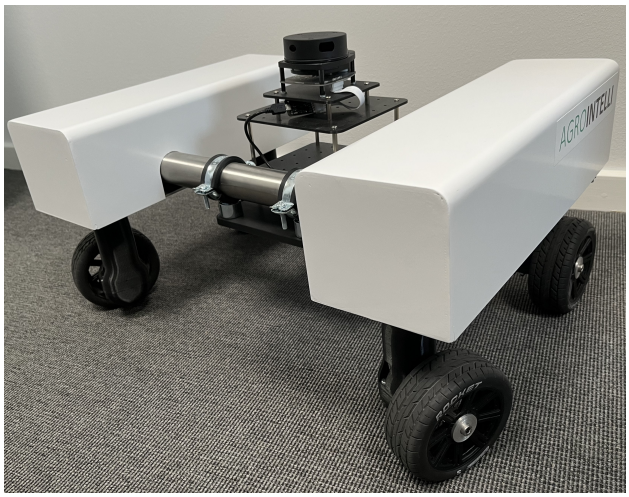


Figure 5: The Desktop Robotti.

5.1. Case-Study: The Desktop Robotti

The evaluation of the OSA approach is performed on the Desktop Robotti case-study, for which we are building a DT able to follow and monitor its behaviour (see preliminary results [48]). The Desktop Robotti (Figure 5) is a miniature version of the Robotti field robot [49], developed by AgroIntelli, and enables us to experiment with various ideas in the lab, at a low cost.

A teleoperation interface running on a virtual machine with ROS2 middleware [50] enables a user to operate the Desktop Robotti, effectively acting as the system's controller. Movement commands consist of the desired speed and steering angle, and are initially calculated for a virtual bicycle model (for a detailed derivation the reader is pointed to [51]). This model is used to describe in a simple a way the motion of a 4-wheel differential drive vehicle, by placing a virtual wheel in the centre of the front and rear axes of said vehicle. After the calculations are performed for the

two virtual wheels, these are translated into values for each wheel (when turning, each wheel must rotate at different speeds). The control loop for each rear wheel runs on an Arduino UNO3, which communicates through a serial link with the ROS2-node running on a Raspberry Pi 4 Model B+ (RPi) that continuously sends control parameters. The steering servo motors on the other hand, are directly coupled to the RPi, and are controlled by a dedicated ROS2-node. In order to get position data, the robot is coupled with the MarvelMind indoor positioning system (IPS). There are 4 stationary beacons which act as references for a single mobile beacon placed on the robot. The Desktop Robotti is also equipped with a 2D-Lidar, that supplies the planar scans of the surroundings. Thus, it is possible to track how the robot moves with respect to its initial pose.

5.2. Experimental Design

The Desktop Robotti is coupled to a co-simulation that consists of the RMQFMU, OSAFMU, and a counter FMU (similar to Figure 2). The latter outputs an integer, that increases step-wise for every co-simulation step, and is fed as input to the RMQFMU. The Desktop Robotti, driven along a straight path, will publish its position (x, y) every $100ms$ to the co-simulation, whereas the co-simulation, configured with a default step of 100ms, publishes in every such step the value of the counter. Note that, in a more realistic setting the counter FMU could be replaced by an FMU that sends control data to the robot, or other feedback that can trigger specific behaviours on said robot. For the purposes of this paper, a counter FMU that continuously sends data to the robot is deemed sufficient, as the focus is on how the DT behaves in an out-of-sync situation, rather than what actual data is sent to the robot.

In order to simulate network degradation and temporary dropout, we use a relay node able to introduce desired delays in the communication between the robot and the co-simulation, *i.e.*, between the PT and its DT. The relay node gets data from other nodes in the PT through the ROS network, via publish/subscribe mechanisms, and exchanges data with the DT through the RabbitMQ server. Normally this node simply publishes information back and forth between the DT and PT, as fast as it can. For the purpose of our tests, two other modes of operation have been added to the relay node that can be activated for configurable periods of time, *e.g.*, start at t_i and end at t_j (naturally $t_i < t_j$). In the network degradation case, if within the activation period, the relay node waits for a delay of γ, before sending a message to the DT. When the activation period is over, *i.e.*, $t > t_j$ then the relay node resumes its usual behaviour and sends data as soon as it can. Note that the delay cannot be bigger than the RMQFMU timeout. In the second case, if within the activation period, the relay node collects the messages as they come, and publishes them in a batch after the delay of γ. After the activation period is over, the relay node resumes its normal operation. Similarly to the first case, such delay cannot be bigger than the RMQFMU timeout value. Since we are interested in evaluating the response of the DT to network issues, we focus on when messages are received on the DT side. We do not consider for example a burst of messages on the PT side.

The goals of these experiments are to show that:

1. The DT does not send data when in an out-of-sync situation.
2. The DT can recover when possible after a temporary network dropout.

The parameters of interest in our scenarios, *i.e.*, network degradation and network dropout, are the degradation delay γ, safety threshold θ_{safe}, and the jump size J (only relevant for the network drop scenario). To set these parameters, we need to calculate the observed baseline network delay γ_{min}, as well as the calculated maximum allowable network delay γ_{max}. Parameters γ and θ_{safe} then have to be between γ_{min} and γ_{max}, where the former is simply a physical constraint, and the latter is a necessary constraint to ensure safe operation, *e.g.*, by avoiding collisions. We show in the paragraphs below how these parameters are calculated for the experiments shown in this paper.

Configuration of the Network Degradation Scenario. In order to adopt a realistic γ and θ_{safe} in our experiments, we computed γ_{min} based on empirical measurements of network delay, and computed the maximum allowable delay in the system γ_{max} based on a bicycle kinematic model approximating the Desktop Robotti geometry.

Before we detail how γ_{min} was computed, it is worth characterising the different delays between the components in the system. The involved components are illustrated in Figure 6. The DT and parts of the ROS network run on a PC, and communicate with one another through the RabbitMQ server. The ROS components on the PC communicate with the RPi (mounted on the Desktop Robotti) through WiFi, whereas the positioning data from the proprietary positioning system comes in through a serial connection. The RPi communicates with the micro-controller (MCU)

via a serial link. The delay of the control loop is broken down into sensor delay (IPS → Simulation) which corresponds to γ_{min}, and actuation delay (Simulation → ROS and ROS → RPi). The typical values for these delays were measured empirically, over 10 independent runs of the Desktop Robotti. The results are summarized in Table 1. Note that, the values corresponding to the delay from Simulation → ROS are not measured. We expect these to be similar to ROS → Simulation, since there are no sources for asymmetry in our experiments. Based on our measurements we estimate a mean value for $\gamma_{min} = 249.04ms$ with standard deviation $\sigma_{\gamma min} = 95.74ms$, as a result of the addition of the IPS to position system delay plus the position system to simulation delay. The considerable delay at the IPS is affected by its internal processing. It is possible to increase the update rate at the loss of accuracy.

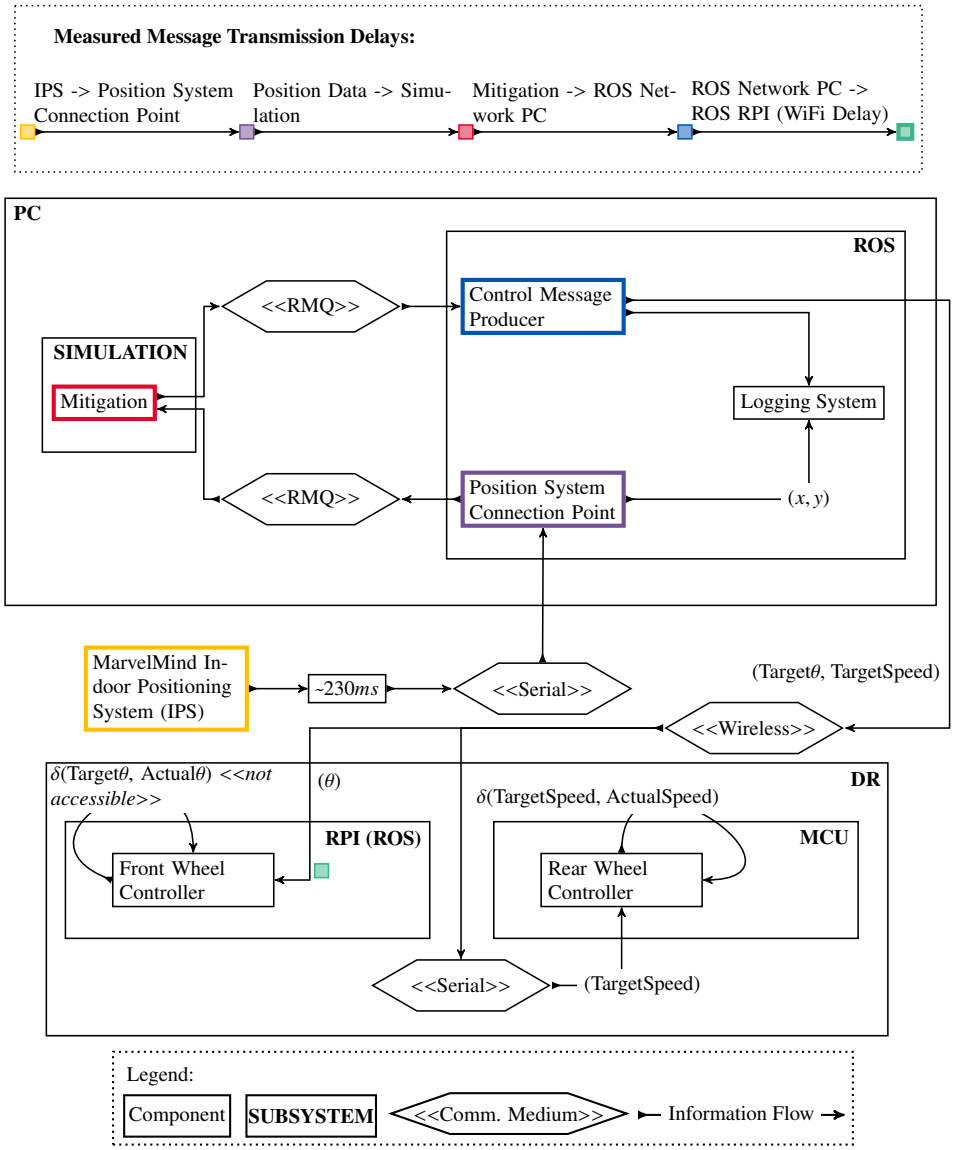


Figure 6: Illustration of the components and information flow in the system comprising the DT (the co-simulation) and the ROS network, and PT, i.e., the Desktop Robotti. In the figure the target headings and speeds represent what is given by the user, whereas the actual headings and speeds refer to what is actually achieved by the robot; θ represents the heading angle; whereas δ represents the difference between actual and target values.

The calculation of γ_{max} must be done via simulation because it corresponds to a safety limit. In order to see that, note that we must contrive a scenario where excessive network delay may lead to an unsafe behaviour, such as a collision with an obstacle. To avoid damaging the real Desktop Robotti and because the real Desktop Robotti makes

Table 1: Results for empirically measured communication delays. The communication paths refer to the components in Figure 6. Note that the second row is in italics because it is not measured directly. The values from the fourth line are used instead, as it is assumed that the delay is symmetrical.

Communication Path	Mean	Std. Dev	Min	Max
IPS → Position System (ROS)	237.64 ms	95.18 ms	98.09 ms	436.13 ms
Position System (ROS) → Simulation	*11.40 ms*	*10.38 ms*	*8.96 ms*	*112.80* ms
γ_{min}	249.04 ms	95.74 ms	97.05 ms	548.93 ms
Simulation → ROS	11.40 ms	10.38 ms	8.96 ms	112.80 ms
ROS → RPI	23.99 ms	5.55 ms	15.75 ms	35.23 ms

use of the Lidar to prevent collisions, we must approximate this value from simulations using a bicycle kinematic model in an obstacle avoidance scenario, where Lidar is not being used. We stress that γ_{max} is an absolute worst case delay that in practice is rarely achievable if all components of the Desktop Robotti are functioning correctly.

The simulation flow is as follows:

1. we assume that the Desktop Robotti is not using lidar, so the only sensors for location are given by the IPS, where the locations of the obstacles are known;
2. the Desktop Robotti moves in straight line, and the simulation predicts the future position of the Desktop Robotti (a prediction of 15 simulation steps (1.5s) into the future is done, where each simulation step is equal to $100ms$);
3. the simulation receives the position of the Desktop Robotti (fed as historical data) with a delay;
4. if the simulation detects that the future Desktop Robotti will collide with an obstacle, it warns Robotti to turn and avoid the obstacle;
5. the amount of steering is proportional to how close the Desktop Robotti is to colliding with the obstacle;
6. the delay occurs in the warning message, so naturally there will be a delay for which the warning arrives too late and leads to a collision;
7. for more realism, we place different obstacles, laid in a way that the Desktop Robotti will steer clear from one obstacle and head straight into another obstacle.

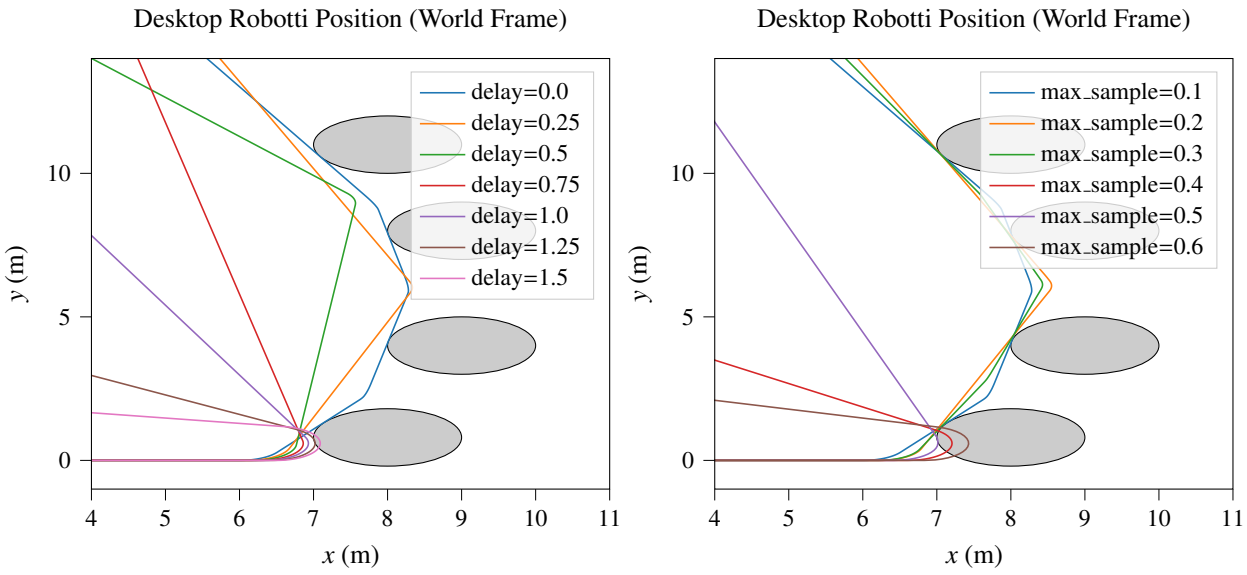
We repeated the above simulation for increasing delay values (we assume same delay for Step 3 and Step 6), until we observe a collision. We fixed the prediction to 15 simulation steps (roughly 0.24m ahead of the Desktop Robotti) and the speed of the Desktop Robotti to 0.16m/s. The results summarised in Figure 7a show that $\gamma_{max} \approx 1.25s$, as it is the maximum value for which the robot does not collide. For $\gamma_{max} \approx 1.5s$ we can observe that the robot collides. Note that, in the figure, the oval objects represent the obstacles, the curves represent the trajectories, whereas the different colours represent the different delays. The longer the delay, the shorter the time to turn, and thus divert from the obstacles.

Based on γ_{min} and γ_{max}, we set γ to a value 20% lower than γ_{max} – a noticeable delay is preferred in our experiments without resulting in a crash, and θ_{safe} to a value 20% higher than γ_{min} – the smallest possible value for the threshold is preferred for higher accuracy.

Configuration of the Network Drop Scenario. The jump size J is bounded by how many messages can be safely skipped without causing the system to become unsafe. To estimate this value, we use the same simulation scenario as the Network Degradation, but instead of having a delay, we successively increase the communication step size until a collision is observed. More precisely, in this case messages in-between are being dropped, and are never processed, as opposed to inducing a delay, where all messages are processed, albeit after the imposed delay. This is a reasonable approximation because, in the worst case, messages being skipped represent a simulation that samples the system at a lower rate than normal. Assume this step limit is MAX_SAMPLE, then the number of messages J that can be safely skipped is given by:

$$J \cdot \Delta h \leq MAX\_SAMPLE, \tag{4}$$

where Δh represents the co-simulation step size and system sample size (i.e. the sensor sampling interval). In other words, if we sample the system with MAX_SAMPLE without collisions occurring, then collisions should also not



(a) (b)

Figure 7: a Position of the robot in (x, y) with different delays in the message handling. b Position of the robot in (x, y) with different MAX_SAMPLEs. In both figures, shaded areas indicate obstacles.

Table 2: Measured and estimated parameters for the network degradation and drop scenarios The calculated values are $\gamma = 0.8 \cdot \gamma_{max}$, and $\theta_{safe} = 1.2 \cdot \gamma_{min}$.

Scenario / Parameters	γ_{min}	γ_{max}	θ_{safe}	γ	step-size	maxage	MAX_SAMPLE
Network Degradation	$249.04ms$	$1.25s$	$299ms$	$1s$	$100ms$	$300ms$	n/a
Network Drop	$249.04ms$	$1.25s$	$299ms$	$1s$	$100ms$	$300ms$	$300ms$

occur if we simulate the system with sample step size Δh but always skipping J messages. The results summarized in Figure 7b show that $MAX\_SAMPLE \approx 0.3$, which means we can in the worst case skip 3 messages.

Note that, in the presented results (Figure 7b) there is no orderly progression for the estimated MAX_SAMPLEs. This would depend on which message is being dropped and which goes through and is processed. In some cases, one could be lucky such that the important messages (noting the presence of an obstacle for example) are retained, while in other cases the retained messages are not useful, thus leading the robot to collision. In the current setup, the messages being dropped are always in the middle, and the edge messages are kept. The analysis could be improved by selecting randomly in each step which messages are being dropped.

5.3. Results

Based on the analysis described in the previous paragraphs, we set the parameters for both scenarios as follows: $\gamma = 1s$, $\theta_{safe} = 299ms$ for network degradation, and $\gamma = 1s$, $\theta_{safe} = 299ms$, and $J = 3$ messages, thus the maximum step size is equal to $300ms$, for network drop (see Table 2). Moreover, in both cases we consider a frequency of sending data equal to $100ms$, where γ_{min} is observed at a mean value of $249.04ms$ (see Table 1). We set the step-size of the co-simulation at $100ms$, and a *maxage* equal to $300ms$ that provides some buffer, as the timestamps of the incoming messages might not exactly match the simulation step, *e.g.*, $0.62s$ instead of $0.6s$. Each set of experiments is repeated 5 times.

Normal Operation. In order to have a base-line, we run our experiments in a normal mode, *i.e.*, without any induced delay in the system (Figure 8). It is possible to observe that in 4 out of 5 runs the system remains well beneath θ_{safe}, whereas in Run 4 the delays are such that the threshold is violated more often. We can observe from Figure 9a that

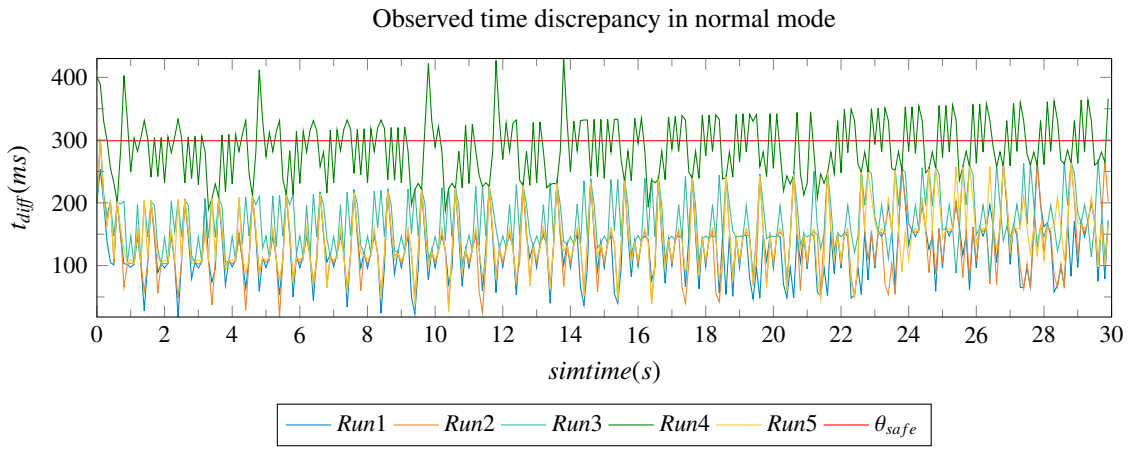


Figure 8: Results for an execution in normal mode for a step-size of $100ms$, frequency of sending data of $100ms$, *maxage* of $300ms$, and an experiment duration of $30s$, where θ_{safe} is represented by the red line.

these violations are mostly one off, but in some cases there can 2, 3, or 4 consecutive steps that violate the thresholds. This can be due to the irregularities in the network that are outside of our interference.

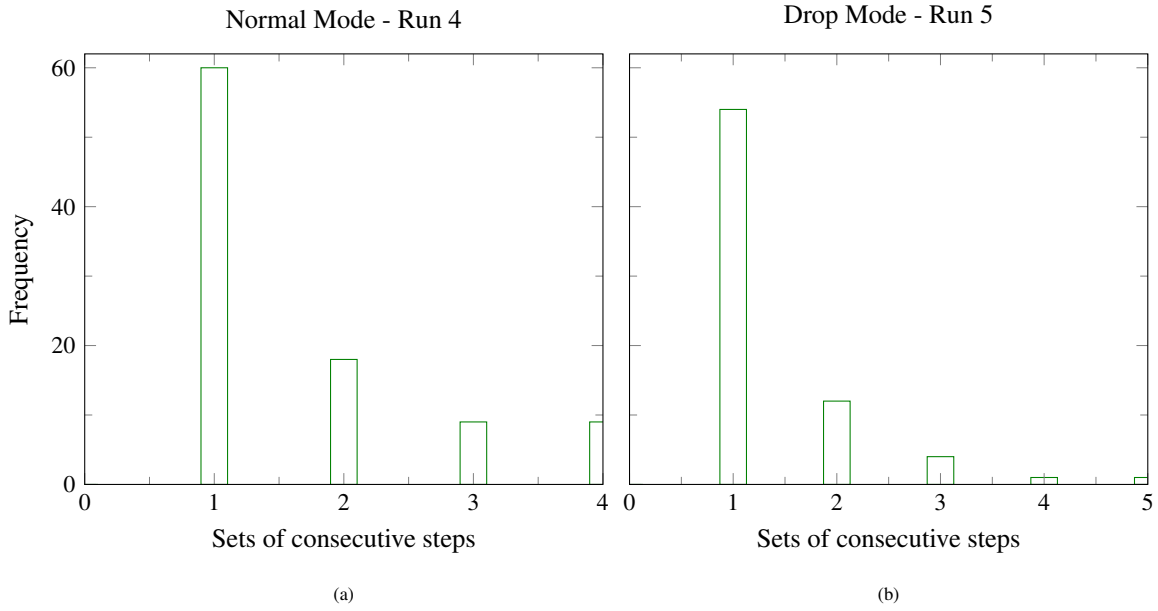


Figure 9: Histograms displaying the frequency of sets of consecutive steps with values for t_{diff} over θ_{safe} over 2 complete simulation runs, shown in (a) for Run 4 in normal mode, and (b) for Run 5 in drop mode. In both a) and b) in the majority of cases the consecutive steps violating the θ_{safe} are lower than the calculated $MAX\_SAMPLE = 300ms$.

Network Degradation. In the degraded scenario[4] the delay $\gamma = 1s$ is induced such that for a duration of $5s$, messages are sent every second; note that messages themselves are time-stamped $100ms$ apart. This means that the delay will accumulate, as messages come $1s$ apart, whereas the DT is able to progress only by $100ms$. This is visible in Figure 10,

[4]A video recording of the degraded scenario can be found at `https://drive.google.com/file/d/1_WWHDM6l1tgRhWLBB-yeZkCAdMQE-OBO/view`

where the maximum t_{diff} is at around $5s$. Once the normal conditions are re-instated, the DT picks up the pace and is able to catch-up with the wall-clock time. It does so in 60.8 time-steps of $100ms$ averaged over the 5 runs. This is

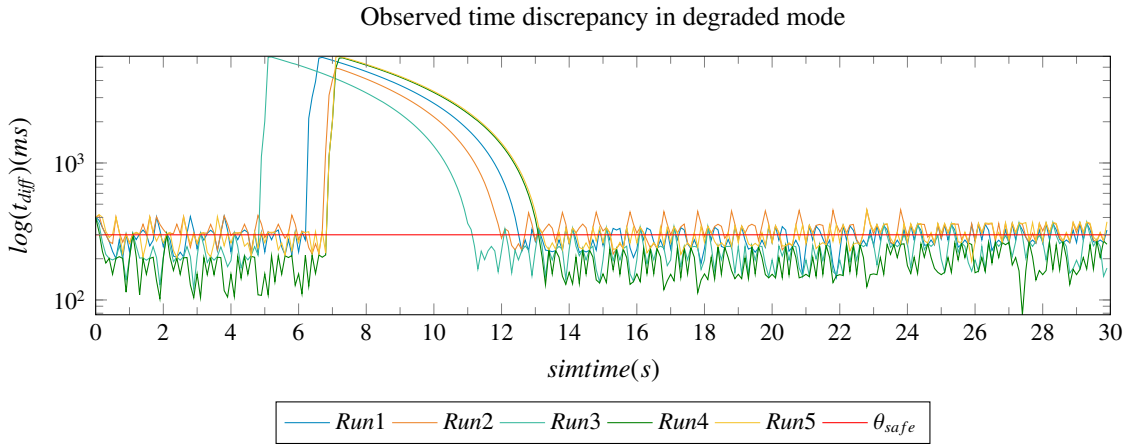


Figure 10: Results for an execution in degraded mode for a step-size of $100ms$, frequency of sending data of $100ms$, *maxage* of $300ms$, an experiment duration of $30s$, $\gamma = 1s$, $\theta_{safe} = 299ms$, and $J = 100ms$.

possible when the DT is able to run faster than real-time, where messages are processed fast enough, allowing it to progress fast in time (simulation time). As in the normal mode, there are still cases outside of the degrade period, in which the threshold is violated. In any case, the DT successfully degrades into a DS mode, with no messages being propagated to the PT, and the user is notified immediately. We can observe in Table 3, the total number of generated messages (always 300, since we always step by $0.1s$, the number of messages received on the ROS side, the number of messages skipped during normal conditions when the threshold was exceeded by 10s of milliseconds (these are not necessarily in consecutive order), and the number of messages skipped during the induced delay (these are consecutive one after the other).

Table 3: Results for the network degradation scenario, consisting in number of messages generated and received, number of messages overall, and number of messages skipped during the degraded period.

Run#	Msgs Generated	Msgs Received	Msgs Skipped	Msgs skipped (delay)
1	300	144	93	63
2	300	70	178	52
3	300	216	21	63
4	300	234	4	62
5	300	112	124	64

Network Dropout. In the dropout scenario, the delay $\gamma = 1s$ is induced such that during $1s$ no messages are sent to the DT. After the dropout period is over, the accumulated messages are sent in a batch as fast as possible. Thereafter, the publishing continues as in normal mode. It is possible to observe that the DT catches up fast to the wall-clock time, however the speed of the simulation is not the only factor at play here. In this case, the DT attempts to take bigger step-sizes, with the maximum set at J, and thus will try to step by $300ms$ at a time, once the burst of messages comes. On average among the 5 runs, it takes 3.8 step-sizes to get in sync, where the step-size is at $300ms$. Similarly to the normal and degraded mode, the threshold is violated during normal conditions in some cases, the worst case is observed in Run 5 (Figure 9b). Similar to Figure 9a, the majority of these violations are one off, while in some cases there are 2, 3, or 4 consecutive steps in violations. Note that the case with 5 consecutive steps corresponds to the drop period. As in the degraded mode, the DT successfully degrades to a DS. We can observe in Table 4 that the

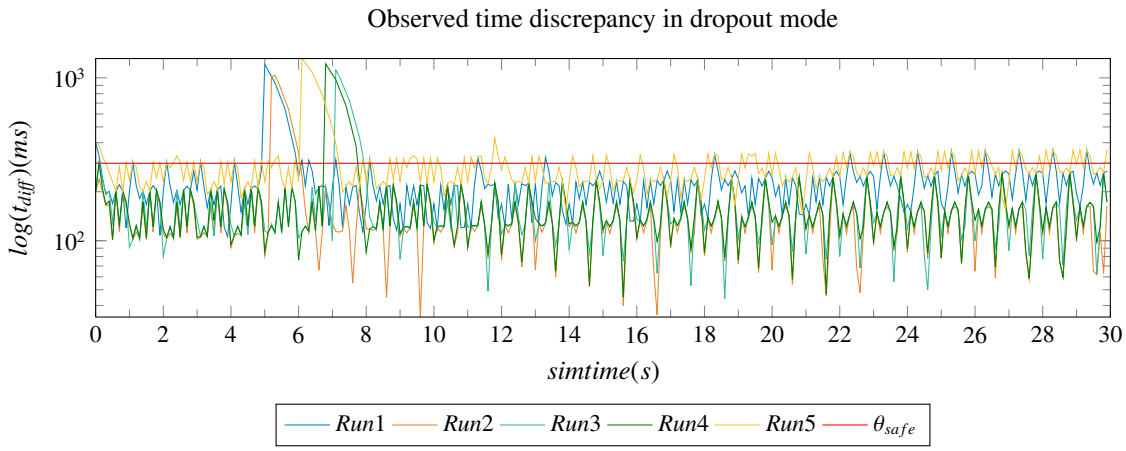


Figure 11: Results for an execution in drop mode for a step-size of 100ms, frequency of sending data of 100ms, *maxage* of 300ms, an experiment duration of 30s, $\gamma = 1s$, $\theta_{safe} = 299ms$, and $J = 300ms$.

total number of generated messages is not the same. This is due to taking some step-sizes equal to 0.3s, as opposed to only 0.1s as in the degradation scenario. In this case, only a handful of messages are skipped during the induced delay. Outside the dropout period, a couple of messages are skipped due to the threshold being exceeded by 10s of milliseconds (Runs $2-4$), whereas in Run 1 and 5, the threshold has been exceeded 21 and 90 times respectively.

Table 4: Results for the network drop scenario, consisting in number of messages generated and received, number of messages overall, and number of messages skipped during the degraded period.

Run#	Msgs Generated	Msgs Received	Msgs Skipped	Msgs skipped (delay)
1	294	269	21	4
2	296	290	1	5
3	296	289	2	5
4	292	286	2	4
5	294	199	90	5

The speedup achieved by the mitigation has been estimated as well. It is possible to observe in Table 5 that the speedup varies from 1.95 to 2.6 in these runs. The estimation was performed as follows for each run independently. First the number of steps completed, and time to get back in sync were noted down, as well as the time for each individual step (the sync time was divided by the number of steps). The time to sync was calculated using the timestamps of the `doStep` calls extracted from the logs. Thereafter, it was calculated how many steps would have been taken if only 100ms steps were possible, and how much time that would have taken, using the time per step. Finally, the time saved was estimated by subtracting the achieved sync time from the estimated total time, whereas the speedup achieved was calculated by dividing the time saved over the estimated total time.

6. Related Work

The synchronisation problem for co-simulation coupled with HIL has been formalised in recent research [31], where the authors also define real-time (RT) synchronisation conditions for the correct coupling of the two. Thereafter, it is shown how to setup the step sizes for both the RT simulator and HIL component, based on expected network delay, such that the RT conditions can be upheld. The focus of this work is on the setup phase of such a co-simulation, as opposed to our proposal that focuses on mitigation strategies when systems get out-of-sync during actual operation. Obermaier *et. al* on the other hand propose a set of metrics that allows the evaluation of real-time capabilities of

Table 5: Results for observed time measurements in the network drop scenario, where (i) the period represents the simulation time during the drop conditions, (ii) steps done represents the steps performed until the system was back in sync, (iii) all steps represents the number of steps that would have been taken if no bigger steps than the default (100ms) were taken, (iv) sync time is the time it took the system to get back in sync, (v) time/step is the time it took for each individual step, (vi) estimated total time is the calculated value on how much time it would have taken to get back in sync if all steps were performed, (vii) using the time/step as a measure, (viii) saved is the amount of time that was saved by taking bigger step sizes, and (ix) the speedup is calculated as the total estimated time over the sync time.

Run#	Period	Steps Done	All Steps	Sync Time	Time/Step	Est. Tot. Time	Saved	Speedup
1	5.0-6.0 (s)	5	11	17 (ms)	3.4 (ms)	37.4 (ms)	20.4 (ms)	2.2
2	5.2-6.1 (s)	6	10	15 (ms)	2.5 (ms)	25(ms)	10 (ms)	2.5
3	7.1-8.0 (s)	6	10	15 (ms)	2.5 (ms)	25 (ms)	10(ms)	2.5
4	6.8-8.0 (s)	5	13	14 (ms)	2.8 (ms)	36.4 (ms)	22.4 (ms)	2.6
5	6.1-7.2 (s)	6	12	16 (ms)	2.6 (ms)	31.2 (ms)	15.2 (ms)	1.95

different systems, *i.e.*, either soft or firm real time, within the context of PDES (parallel discrete event simulation) [23]. Specifically, they calculate the maximum time difference (between simulation time and wall-clock time) among the simulation units, for all simulation steps, as well as the difference of the former metric with minimum time difference (between simulation time and wall-clock time) among the simulation units. These two metrics are then used to determine the real-time capabilities with respect to the relation to wall-clock time, and the synchronisation among the simulation units.

A recent study has reviewed DT applications in the manufacturing domain, *i.e.*, manufacturing execution systems (MES), reporting on the gaps between what are theoretical features of DTs and the current state-of-art [52]. Additionally, the authors state that the DT-PT communication is not bi-directional with the field in current applications, with the exception of three ad-hoc approaches that enable the DT to PT link [53, 54, 55]. A simulation model for a DT based on MATLAB/Simulink was also proposed by the authors [52]; while still in the digital shadow state, the authors argue that the technical setup to allow the communication from DT to PT is present. Coronado *et. al* focus on a DT for the shop floor, realised with the MTConnect protocol, an open source protocol intended for getting data from manufacturing equipment [56], for MES [53]. Hu *et. al* also adopt the MTConnect protocol to create a cloud-based DT, such that different components can be monitored and operated from the cloud, as a result of a user request [54]. They compare this with the sensor-server DT approach. It seems that commands can go from the DT to the physical machine, however the focus is more on the creation of the DT per user request, in terms of the information model with respect to the physical machine. Also it seems that the sync between the different parts of their system is performed at the configuration centre part of their knowledge resource module. Shahriar *et. al* use the MTComm protocol to enable the control of the system from the digital twin for operations "Move axes of Bukito machine from the Digital-Twin" and "Operate electrical power supply of Bukito machine from the Digital-Twin" [55]. In the authors own words: "MTComm is a service-oriented method to establish connection from the cloud to the machine tools. It offers several RESTful web services to monitor, diagnose, and operate machine tools over the Internet."

In recent years, techniques have been proposed that deal with network delays by (i) having the simulation/HW sharing not only their current states, but also their predicted state for a sequence of future steps that can be used in case of delays [57]; and (ii) predicting the outputs of other components locally, while also considering varying delays [58]. Furthermore, the HLA standard for distributed co-simulation [59] includes a mechanism that allow simulation units to transmit dead-reckoning models to each other, that act as delay compensators.

7. Concluding Remarks

In this paper we propose a mechanism that enables a co-simulation based DT to detect time discrepancies between the DT and its PT during run-time, that arise as a result of network degradation. Such information is used to degrade to a DS mode when a (user configurable) threshold is exceeded. Although not considered in this paper, such methods could in principle also be useful in attack prevention, where the time difference comes as a result of time delay attacks [43]. In addition to being able to fall back to a DS mode, the DT attempts to recover when possible by

attempting bigger step-sizes that aim to diminish the difference between the DT time and the wall-clock time. In the experiments ran for the purpose of this paper, the DT always manages to catch up to the PT. Nevertheless, it might not always be possible to bring the simulation forward. During a persistent degraded scenario the delay will keep accumulating. As such, the DT will not be able to take bigger step-sizes because there would be no new data in the RMQFMU; the capability to run faster than real-time is of no use either. Whereas, during a dropout scenario, after a burst of messages come in, the RMQFMU would be able to jump far ahead to the latest message in the queue. However, other FMUs could have stricter limitations on the maximum step-size that can be taken, slowing as a result the pace with which the DT gets in sync. Additionally, in the results presented in this paper, it is possible to see that even during a normal run the calculated threshold was exceeded on occasion. This pin-points to the need for a more thorough network evaluation for the estimation of the time discrepancy threshold, also considering potential variability during normal conditions. The delays within the DT itself, and the platform that it is run on should also be factored in, to ensure a higher accuracy. In general, getting the system back in-sync can prove to be quite complex. Being able to take bigger step sizes might not always be possible, thus the re-synchronisation would rely on how fast the a simulation step is computed as compared to the real-time such step represents, *e.g.*, 100ms in simulation should be performed in much less than 100ms real-time. These are obvious limitations, that leave DT developers with the task of improving their solvers in terms of time performance, as well as flexibility in being able to take bigger step-sizes, that can both come at the cost of accuracy, the usual trade-off in such cases.

There are a few directions for future work. (I) We would like to investigate the reasons for the time discrepancies, *i.e*, being able to tell whether it is due to the network; if not then identifying which components in the system are responsible for such discrepancies, thus providing a better understanding of the simulation-reality gap. (II) It is of interest to consider situations in which a degraded mode is persistent, in terms of how the DT should behave, and how an operator can be involved in making decisions, *e.g.*, terminate/restart DT etc. (III) There can also be different ways in which the safety threshold θ_{safe} is calculated. As an example, by considering the round-trip bound of a message, and estimating the delays that incur in the system components. It is also of interest to compare these alternative methods, and evaluating in series of experiments, the most reliable.

Acknowledgements

We acknowledge the Poul Due Jensen Foundation that funded our basic research for engineering of digital twins. We would also like to thank Gill Lumer-Klabbers and Jacob Odgaard Hausted for all their contributions for the Desktop Robotti case-study. In addition we would like to thank Innovation Foundation Denmark for funding the AgroRobottiFleet project and ITEA for funding the UPSIM project.

References

[1] E. A. Lee, Cyber Physical Systems: Design Challenges, Tech. Rep. UCB/EECS-2008-8, EECS Department, University of California, Berkeley (Jan 2008).
URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html

[2] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, Co-simulation: a Survey, ACM Comput. Surv. 51 (3) (2018) 49:1–49:33.

[3] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J. Peetz, S. Wolf, G. I. T. I. Gmbh, D. L. R. Oberpfaffenhofen, The Functional Mockup Interface for Tool independent Exchange of Simulation Models, in: 8th International Modelica Conference, Munich, Germany, 2011, pp. 105–114.

[4] M. Grieves, J. Vickers, Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems, in: F.-J. Kahlen, S. Flumerfelt, A. Alves (Eds.), Transdisciplinary Perspectives on Complex Systems, Springer International Publishing Switzerland, 2017, pp. 85–113.

[5] W. Zhang, M. Branicky, S. Phillips, Stability of networked control systems, IEEE Control Systems Magazine 21 (1) (2001) 84–99. doi:10.1109/37.898794.

[6] C. Thule, K. Lausdahl, C. Gomes, G. Meisl, P. G. Larsen, Maestro: The INTO-CPS co-simulation framework, Simulation Modelling Practice and Theory 92 (2019) 45 – 61. doi:https://doi.org/10.1016/j.simpat.2018.12.005.
URL http://www.sciencedirect.com/science/article/pii/S1569190X1830193X

[7] V. M. Ionescu, The analysis of the performance of rabbitmq and activemq, in: 2015 14th RoEduNet International Conference-Networking in Education and Research (RoEduNet NER), IEEE, 2015, pp. 132–137.

[8] J. Fitzgerald, C. Gamble, P. G. Larsen, K. Pierce, J. Woodcock, Cyber-Physical Systems Design: Formal Foundations, Methods and Integrated Tool Chains, in: Proceedings of the Third FME Workshop on Formal Methods in Software Engineering, Formalise '15, IEEE Press, 2015, pp. 40–46, event-place: Florence, Italy.

[9] R. Kubler, W. Schiehlen, Two methods of simulator coupling, Mathematical and Computer Modelling of Dynamical Systems 6 (2) (2000) 93–113. doi:10.1076/1387-3954(200006)6:2;1-m;ft093.

[10] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, Co-Simulation: A Survey, ACM Computing Surveys 51 (3) (2018) 1–33, place: New York, NY, USA Publisher: ACM. doi:10.1145/3179993.
URL http://doi.acm.org/10.1145/3179993

[11] Modelica Association, Functional Mock-up Interface for Model Exchange and Co-Simulation, https://fmi-standard.org/downloads/ (December 2020).

[12] T. Blochwitz, M. Otter, J. Åkesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, A. Viel, Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models, in: Proceedings of the 9th International Modelica Conference, The Modelica Association, 2012, pp. 173–184. doi:10.3384/ecp12076173.
URL https://lup.lub.lu.se/search/ws/files/5428900/2972293.pdf

[13] D. Broman, C. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, M. Wetter, Determinate composition of FMUs for co-simulation, in: 2013 Proceedings of the International Conference on Embedded Software (EMSOFT), IEEE, 2013, pp. 1–12. doi:10.1109/EMSOFT.2013.6658580.
URL http://ieeexplore.ieee.org/document/6658580/

[14] J. Bastian, C. Clauß, S. Wolf, P. Schneider, Master for co-simulation using FMI, in: Linköping Electronic Conference Proceedings, Linköping University Electronic Press, 2011. doi:10.3384/ecp11063115.

[15] C. Thule, M. Palmieri, C. Gomes, K. Lausdahl, H. D. Macedo, N. Battle, P. G. Larsen, Towards reuse of synchronization algorithms in co-simulation frameworks, in: Software Engineering and Formal Methods, Springer International Publishing, 2020, pp. 50–66. doi:10.1007/978-3-030-57506-9_5.

[16] A. Bagnato, E. Brosse, I. Quadri, A. Sadovykh, Into-cps: An integrated "tool chain" for comprehensive: model-based design of cyber-physical systems, in: ICSSEA 2015 Proceedings, 2015, this publication is part of the Horizon 2020 project: Integrated Tool chain for model-based design of CPSs (INTO-CPS), project/GA number 644047.; null ; Conference date: 27-05-2015 Through 29-05-2015.

[17] C. Thule, C. Gomes, K. Lausdahl, Formally Verified FMI Enabled Data Broker: RabbitMQ FMU, SummerSim '20, ACM New York, NY, USA, p. to appear.

[18] M. Frasheri, H. Ejersbo, C. Thule, L. Esterle, Rmqfmu: Bridging the real world with co-simulation for practitioners, in: H. Macedo, C. Thule, K. Pierce (Eds.), Proceedings of the 19th International Overture Workshop, 2021, pp. 66–80, null ; Conference date: 22-10-2021 Through 22-10-2021.

[19] R. M. Fujimoto, Parallel and distributed simulation systems, Vol. 300, Citeseer, 2000.

[20] D. R. Jefferson, P. D. Barnes, Virtual time iii: Unification of conservative and optimistic synchronization in parallel discrete event simulation, in: 2017 Winter Simulation Conference (WSC), IEEE, 2017, pp. 786–797.

[21] C. Köhler, Connection between Hardware and Simulation, Vieweg+Teubner, Wiesbaden, 2011, pp. 29–54. doi:10.1007/978-3-8348-9916-3_3.
URL https://doi.org/10.1007/978-3-8348-9916-3_3

[22] C. D. Carothers, K. S. Perumalla, On deciding between conservative and optimistic approaches on massively parallel platforms, in: Proceedings of the 2010 Winter Simulation Conference, IEEE, 2010, pp. 678–687.

[23] C. Obermaier, R. Riebl, A. H. Al-Bayatti, S. Khan, C. Facchi, Measuring the realtime capability of parallel-discrete-event-simulations, Electronics 10 (6) (2021) 636.

[24] A. Park, R. M. Fujimoto, K. S. Perumalla, Conservative synchronization of large-scale network simulations, in: 18th Workshop on Parallel and Distributed Simulation, 2004. PADS 2004., IEEE, 2004, pp. 153–161.

[25] K. M. Chandy, J. Misra, Distributed simulation: A case study in design and verification of distributed programs, IEEE Transactions on software engineering (5) (1979) 440–452.

[26] R. E. Bryant, Simulation of packet communication architecture computer systems., Tech. rep., MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE (1977).

[27] B. Lubachevsky, A. Schwartz, A. Weiss, An analysis of rollback-based simulation, ACM Transactions on Modeling and Computer Simulation (TOMACS) 1 (2) (1991) 154–193.

[28] J. S. Steinman, Breathing time warp, in: Proceedings of the seventh workshop on Parallel and distributed simulation, 1993, pp. 109–118.

[29] P. M. Dickens, D. M. Nicol, P. F. Reynolds Jr, J. M. Duva, Analysis of bounded time warp and comparison with yawns, ACM Transactions on Modeling and Computer Simulation (TOMACS) 6 (4) (1996) 297–320.

[30] D. M. Nicol, J. Liu, Composite synchronization in parallel discrete-event simulation, IEEE Transactions on Parallel and Distributed Systems 13 (5) (2002) 433–446.

[31] T. L. Nguyen, Q. T. Tran, Y. Besanger, et al., Synchronization conditions and real-time constraints in co-simulation and hardware-in-the-loop techniques for cyber–physical energy system assessment, Sustainable Energy, Grids and Networks 20 (2019) 100252.

[32] W. Li, M. Ferdowsi, M. Stevic, A. Monti, F. Ponci, Cosimulation for smart grid communications, IEEE Transactions on Industrial Informatics 10 (4) (2014) 2374–2384.

[33] V. H. Nguyen, Y. Besanger, Q. T. Tran, T. L. Nguyen, et al., On conceptual structuration and coupling methods of co-simulation frameworks in cyber-physical energy system validation, Energies 10 (12) (2017) 1977.

[34] D. R. Jefferson, Virtual time, ACM Transactions on Programming Languages and Systems (TOPLAS) 7 (3) (1985) 404–425.

[35] A. Malik, A. Park, R. Fujimoto, Optimistic synchronization of parallel simulations in cloud computing environments, in: 2009 IEEE International Conference on Cloud Computing, IEEE, 2009, pp. 49–56.

[36] R. M. Fujimoto, Time management in the high level architecture, Simulation 71 (6) (1998) 388–400.

[37] J. A. Ledin, Hardware-in-the-loop simulation, Embedded Systems Programming 12 (1999) 42–62.

[38] C. Gomes, G. Abbiati, P. G. Larsen, Seismic Hybrid Testing using FMI-based Co-Simulation, in: Proceedings of the 14th International Modelica Conference, Linköping University Electronic Press, Linköpings Universitet, online, 2021. doi:10.3384/ecp21181287.

[39] J. Reitz, A. Gugenheimer, J. Roßmann, Virtual hardware in the loop: Hybrid simulation of dynamic systems with a virtualization platform,

in: 2020 Winter Simulation Conference (WSC), IEEE, 2020, pp. 1027–1038.
[40] D. Bullock, B. Johnson, R. B. Wells, M. Kyte, Z. Li, Hardware-in-the-loop simulation, Transportation Research Part C: Emerging Technologies 12 (1) (2004) 73–89. doi:https://doi.org/10.1016/j.trc.2002.10.002.
URL https://www.sciencedirect.com/science/article/pii/S0968090X03000792
[41] D. J. Rankin, J. Jiang, A hardware-in-the-loop simulation platform for the verification and validation of safety control systems, IEEE Transactions on nuclear science 58 (2) (2011) 468–478.
[42] D. M. Lane, G. J. Falconer, G. Randall, I. Edwards, Interoperability and synchronisation of distributed hardware-in-the-loop simulation for underwater robot development: issues and experiments, in: Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164), Vol. 1, IEEE, 2001, pp. 909–914.
[43] G. Bianchin, F. Pasqualetti, Time-Delay Attacks in Network Systems, Springer International Publishing, Cham, 2018, pp. 157–174. doi:10.1007/978-3-319-98935-8_8.
URL https://doi.org/10.1007/978-3-319-98935-8_8
[44] M. Lévesque, D. Tipper, A survey of clock synchronization over packet-switched networks, IEEE Communications Surveys Tutorials 18 (4) (2016) 2926–2947. doi:10.1109/COMST.2016.2590438.
[45] W. P. M. H. Heemels, A. R. Teel, N. van de Wouw, D. Nešić, Networked control systems with communication constraints: Tradeoffs between transmission intervals, delays and performance, IEEE Transactions on Automatic Control 55 (8) (2010) 1781–1796. doi:10.1109/tac.2010.2042352.
[46] C. Gomes, C. Thule, P. G. Larsen, J. Denil, H. Vangheluwe, Co-simulation of Continuous Systems: A Tutorial, Tech. Rep. arXiv:1809.08463, Belgium (2018). arXiv:1809.08463.
[47] C. Gomes, H. Vangheluwe, Co-simulation of continuous systems: a hands-on approach, in: 2019 Winter Simulation Conference (WSC), IEEE, 2019, pp. 1469–1481.
[48] G. Lumer-Klabbers, J. O. Hausted, J. L. Kvistgaard, H. D. Macedo, M. Frasheri, P. G. Larsen, Towards a digital twin framework for autonomous robots, in: SESS: The 5th IEEE Int. Workshop on Software Eng. for Smart Systems, COMPSAC 2021, IEEE, 2021.
[49] F. Foldager, O. Balling, C. Gamble, P. G. Larsen, M. Boel, O. Green, Design Space Exploration in the Development of Agricultural Robots, in: AgEng conference, Wageningen, The Netherlands, 2018.
[50] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot operating system 2: Design, architecture, and uses in the wild, Science Robotics 7 (66) (2022) eabm6074. doi:10.1126/scirobotics.abm6074.
URL https://www.science.org/doi/abs/10.1126/scirobotics.abm6074
[51] J. Woodcock, C. Gomes, H. D. Macedo, P. G. Larsen, Uncertainty Quantification and Runtime Monitoring Using Environment-Aware Digital Twins, in: Leveraging Applications of Formal Methods, Verification and Validation: Tools and Trends, Vol. 12479 of Lecture Notes in Computer Science, Springer International Publishing, 2021, pp. 72–87. doi:10.1007/978-3-030-83723-5_6.
[52] C. Cimino, E. Negri, L. Fumagalli, Review of digital twin applications in manufacturing, Computers in Industry 113 (2019) 103130.
[53] P. D. U. Coronado, R. Lynn, W. Louhichi, M. Parto, E. Wescoat, T. Kurfess, Part data integration in the shop floor digital twin: Mobile and cloud technologies to enable a manufacturing execution system, Journal of manufacturing systems 48 (2018) 25–33.
[54] L. Hu, N.-T. Nguyen, W. Tao, M. C. Leu, X. F. Liu, M. R. Shahriar, S. N. Al Sunny, Modeling of cloud-based digital twins for smart manufacturing with mt connect, Procedia manufacturing 26 (2018) 1193–1203.
[55] M. R. Shahriar, S. N. Al Sunny, X. Liu, M. C. Leu, L. Hu, N.-T. Nguyen, Mtcomm based virtualization and integration of physical machine operations with digital-twins in cyber-physical manufacturing cloud, in: 2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), IEEE, 2018, pp. 46–51.
[56] R. Lynn, E. Wescoat, D. Han, T. Kurfess, Embedded fog computing for high-frequency mtconnect data analytics, Manufacturing Letters 15 (2018) 135–138.
[57] J. Alfonso, J. M. Rodriguez, J. C. Salazar, J. Orús, V. Schreiber, V. Ivanov, K. Augsburg, J. V. Molina, M. Al Sakka, J. A. Castellanos, Distributed simulation and testing for the design of a smart suspension, SAE International Journal of Connected and Automated Vehicles 3 (12-03-02-0011) (2020) 129–138.
[58] Y. Zheng, M. J. Brudnak, P. Jayakumar, J. L. Stein, T. Ersal, A predictor-based framework for delay compensation in networked closed-loop systems, IEEE/ASME Transactions on Mechatronics 23 (5) (2018) 2482–2493.
[59] K.-C. Lin, Dead reckoning and distributed interactive simulation, in: Distributed Interactive Simulation Systems for Simulation and Training in the Aerospace Environment: A Critical Review, Vol. 10280, International Society for Optics and Photonics, 1995, p. 1028004.