GRANT AGREEMENT N°223866

Deliverable	D04.05
Nature	Report
Dissemination	Public

D04.05 - Feasibility mock-ups of feedback schedulers

Report Preparation Date	31/DEC/2011 Project month: 40
Authors	Daniel Simon (NeCS-INRIA) Kevin Planchet (NeCS-INRIA) Émilie Roche (NeCS-INRIA) David Muñoz de la Peña Sequedo (Univ. of Sevilla) Olivier Sename (GIPSA-lab)
Report Version	V1
Doc ID Code	INRIACO01_D04.05_31DEC2011_V1
Contract Start Date	01/09/2008
Duration	41 months
Project Coordinator :	Carlos CANUDAS DE WIT, INRIA, France

Theme 3:

Information and Communication Technologies

SUMMARY

Control and computation co-design deals with the interaction between feedback control laws design and their implementation on a real execution resource. Control design is often carried out in the framework of continuous time, or under the assumption of ideal sampling with equidistant intervals and known delays. Implementation on a real-time execution platform introduces many timing uncertainties and distortions to the ideal timing scheme, e.g. due to variable computation durations, complex preemption patterns between concurrent activities, uncertain network induced communication delays or occasional data loss.

Analyzing, prototyping, simulating and guaranteeing the safety of complex control systems are very challenging topics. Models are needed for the mechatronic continuous system, for the discrete controllers and diagnosers, and for network behavior. Real-time properties (task response times) and the network Quality of Service (QoS) influence the controlled system properties (Quality of Control, QoC). To reach effective and safe systems it is not enough to provide theoretic control laws and leave programmers and real-time systems engineers just do their best to implement the controllers.

The characterization of implementation induced distortions on feedback controllers is usually beyond the capabilities of traditional design tools like Matlab/Simulink or Scilab, which basically target simulation. Advanced networked control, or control under limited computational resources, require a fine grain control over real-time resources which are most often not provided by general purpose control design tools.

This report first describes, through the detailed design of a quadrotor drone controller, the main features of ORCCAD, an integrated development environment aimed to bridge the gap between advanced control design and real-time implementation (Borrelly et al. (1998), Simon (2011)) whose motivations and mainlines has been already given in a companion report (Jimenez et al. (2010)). In particular this controller exploits multi-tasking/multi-rate capabilities of the design tools to perform experiments with flexible and feedback scheduling algorithms. Exception processing in also exploited in the design to carry out diagnosis and fault tolerant control. Besides control design and implementation, a real-time (hardware-in-the-loop) simulation has been designed to assess the control design with a simulated target rather than with the real plant.

Using this HIL structure, several experiments using flexible real-time control features are reported. As far as the computation resource must be managed, varying the control sampling rates is of prime interest. The real-time control and simulation testbed is used to implement and test

several versions of the Extended Kalman Filter modified to comply with input data loss. A second set of experiments uses the so-called (m,k)-firm scheduling, where at least m control tasks instances out of k must be mandatory executed, and the others are optionally executed only in case of sufficient resource. Finally the Autonomous Underwater Vehicle testbed (as described in the FeedNetBack WP8) has been the support for testing varying sampling controller, where the sampling interval is considered as a varying parameter in a LFR system representation associated with a $\mathcal{H}_\infty$ controller. This later testbed also provides a first approach in combining two approaches studied in previous deliverables, i.e. inner control loops based on LPV varying sampling as depicted in Roche, Sename, Seuret, Simon and Varrier (2010) and outer loops based on Model Predictive Control as studied in Muñoz de la Peña Sequedo and Raimondo (2010).

Contents

1 Objectives

To develop effective and safe real-time control for complex applications (e.g. automotive, aerospace...), it is not enough to provide theoretic control laws and leave programmers and real-time systems engineers just do their best to implement the controllers. Complex systems are implemented using digital chips and processors, they are more and more distributed over networks, where sensors, actuators and processors are spread over a naturally asynchronous architecture. For efficiency reasons a given processor is often shared between several computing activities under control of a real-time operating system (RTOS). The control performance and robustness of such system can be very far from expected if implementation induced disturbances are not taken into account. However the detailed understanding of such complex hybrid systems is up-to-now out of the range of purely theoretic analysis.

Analyzing, prototyping, simulating and guaranteeing the safety of these systems are very challenging topics. Models are needed for the mechatronic continuous system, for the discrete controllers and diagnosers, and for network behavior. Real-time properties (task response times) and the network Quality of Service (QoS) influence the controlled system properties (Quality of Control, QoC).

Safety critical real-time systems must obey stringent constraints on resource usage such as memory, processing power and communication, which must all be verified during the design stage. A hard real-time approach is traditionally used because it guarantees that all timing constraints are verified. However, it generally results in oversizing the computing power and networking bandwidth, which is not always compatible with embedded applications. Thus a soft real-time approach is preferred, since it tolerates timing deviations such as occasionally missed deadlines. However, to keep confidence in the system safety, the interactions between the computing system and the control system must be carefully designed and observed whenever the soft real-time approach is used.

Understanding and modeling the influence of an implementation (support system) on the QoC (Quality of control) is a challenging objective in control/computing co-design. Mainly based on case studies several results exist, e.g. (Cervin (2003)), to study the influence of an implementation on the performances of a control system. However real-life size problems, involving non-linear systems and uncertain components, still escape from a purely theoretic framework. Hence the design of an embedded real-time control application needs to be progressively developed along the use of several tools ranging from classic simulation up-to the implementation of run-time code on the target.

The main development phases (summarized in Figure 1) are :

Continuous time simulation A first step is the choice of a control structure, based on a model of the plant and on the control objective. The control toolbox now contains many tools to state and solve a large variety of control problems applied to various plants. Besides control theory, preliminary design and tests often rely on simulation tools such as Matlab/Simulink or Scilab/Scicos. Control design usually needs two different models of the plant. The control algorithm design is based on a simplified **control design model**, which goal is to capture the essential aspects of the plant dynamics and behavior. This abstraction must be simple enough

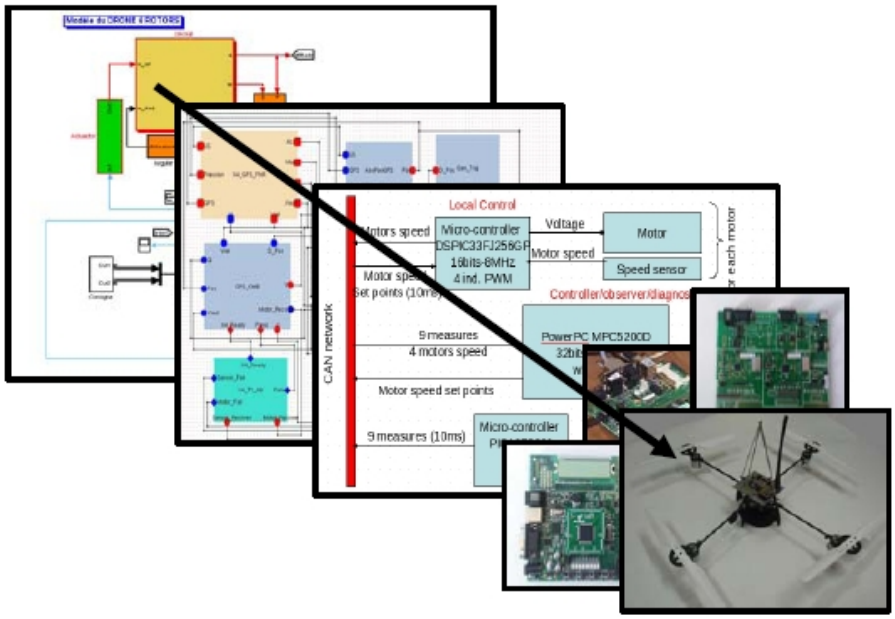

Figure 1: Design flow.

to derive a closed-loop controller, e.g. it can be a linearized model of the plant leading to a LQ controller. Then the robustness of the design w.r.t. the realistic plant should be assessed via simulations handling a complete **simulation models** of the plant, including its known non-linearities together with a model of the uncertain parameters. The aforementioned simulation tools handle some modeling capabilities, e.g. the Matlab physical modeling toolbox. These models can also be generated from external tools and languages, e.g. Siconos and Modelica for models of mechanical systems and robotics.

These preliminary simulations are able to rough out the choice of the control algorithm and firstly evaluate its adequation with the plant structure and control objectives. They are often carried out in the framework of continuous time, as non-linear control is also often designed in continuous time. When done, the evaluation of digital control and discretization consequences is basic, e.g. modeled via a single numerical loop sampled at a fixed rate.

Simulation including the real-time architecture Further to these preliminary simulation, modeling and simulating the implementation platform, i.e. the real-time tasks and scheduler inside the control nodes, and the network between the nodes, allows for a new step towards the design and evaluation of the embedded system. Among others the TRUETIME free toolbox is a Matlab/Simulink-based simulator for real-time control systems (Ohlin et al. (2007)). TrueTime eases the co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics. With this toolbox the real-time operating system's and network's features are handled by high level abstractions, contrarily to other tools such as SimEvent, or as Opnet (Opnet Technologies Inc (2011)), which allows users to

simulate the network in a more detailed (and costly) way. TRUETIME allows users to simulate several types of networks (Ethernet, CAN, Round Robin, TDMA, FDMA, Switched Ethernet and WLAN or ZigBee Wireless networks) and new models can be added.

Hardware-in-the-loop simulation Even with a specialized toolbox like TRUETIME, the previous step remains simulations still far from reality. In particular the models of the real-time components (RTOS and network) must be kept simple (high-level) to avoid prohibitive simulation times, and execution or transmission times are based on assumptions and worst cases analysis. Before experiments using the real system a useful next step is to implement the so-called *hardware-in-the-loop* test-bench. In this case, the plant is still simulated (numerical integration of the non-linear plant model), but now the control algorithms and real-time software are executed on the embedded target as real-time tasks communicating through the real network.

Experiments Finally, once the real-time simulations made the designers confident enough in both the control algorithm and its implementation, experiments with the real plant controlled by the embedded hardware and software can begin while minimizing the risk of early failure. Experiments results can be further used to update the simulation models or even to come back on the choice or dimensioning of some of the system's components.

2 Orccad framework

ORCCAD is a model-based software environment dedicated to the design, the verification and the implementation of real-time control systems[1]. In addition to control law design, the specification and validation of complex missions involving the logical and temporal cooperation of various controllers along the life of a control application (Borrelly et al. (1998)), (Törngren et al. (2006)) can be achieved.

The ORCCAD methodology is bottom-up, starting from the design of control laws by control engineers, to the design of more complex missions.

The first step in designing a control application is to identify all the necessary elementary tasks involved. Then, for each of the tasks, various issues are considered, both with an automatic control viewpoint (such as regulation problem definition, control law design, choice of relevant events, specification of recovery behaviors, etc.) or with an implementation viewpoint (such as the decomposition of the control law into real-time tasks, and selection of timing parameters). Finally, all the real-time tasks are mapped on a target architecture. During this design, the control engineer may take advantage of many degrees of freedom to meet the end-user requirements, and ORCCAD aims at allowing the designer to safely exploit these degrees of freedom through guided design.

ORCCAD proposes a controller architecture which is naturally open, since the access to every level by different users is allowed: the application layer is accessed by the end-user (mission specialist), the control layer is used by the control expert, and the system layer is accessed by the system engineer. ORCCAD provides formalized control structures, which are coordinated using

[1] `http://orccad.gforge.inria.fr`

the synchronous paradigm, specifically using the Esterel language: while the control laws are often periodic (or more generally cyclic) and programmed using real-time tasks under control of a real-time scheduler, the discrete-event controller manages the set of control laws and handles exceptions and mode switching. Both activities run under the control of a real-time operating system (RTOS).

The main entities used in the ORCCAD framework are:

- Algorithmic Modules (MA) which represent functions (e.g. controllers, filters, etc.), encoded as pieces of C code;
- Temporal Constraints (TC), the real-time tasks which implement modules (several modules can be gathered in a single TC);
- Robot Tasks (RT), the control tasks representing basic control actions encapsulated by a discrete-event controller;
- Robot Procedures (RP), a hierarchical composition of already existing RTs and RPs, to incrementally build more complex structures, from elementary executable actions to the full control application.

The RTs characterize continuous-time closed-loop control laws, along with their temporal features and the management of associated events. From the application perspective, the RT set of signals and associated behavior represent the external view of the RTs, hiding all specification and implementation details of the control laws. The RPs, which are more complex actions, can then be composed from RTs and other RPs in a hierarchical fashion leading to structures of increasing complexity. At the top level, RPs are used to describe and implement a full mission specification. At mid-level, they can be used to fulfill a single basic goal through several potential solutions, e.g. a nominal controller supplemented by the recovery substitutions associated with Fault Detection and Diagnosis.

Once a control application has been entirely designed, and for some parts formally verified, a run-time code can be automatically generated for various real-time operating systems, such as Linux in the present case. It is assumed that the underlying RTOS supports preemption and fixed priorities, so that the run-time can be fast ported on others Posix systems, and on systems relying on the same abstractions, as already done for Xenomai.

The relevance of such a modeling and design tool in the framework of control and computation co-design has been emphasized in a previous report D01.04 (Jimenez et al. (2010)). The control design process under Orccad is enlighten in the next section through the design of a multi-rate controller for a quadrotor mini-drone.

3 Testbed: a multi-rate quadrotor drone controller

3.1 Structure

Figure 2 describes the control and diagnostic setup used for testing networked control systems fault-tolerant control, diagnosis and weakly-hard scheduling policies. The main characteristics of

the testbed, together with the description of the control and diagnosis algorithms can be found in (Berbra et al. (2009)) and (Berbra et al. (2010)) and are not detailed in the following sections, which focus on the software design and implementation process.

The following block-diagram represents one Robot-Task, i.e. one control algorithm whose components are executed repetitively during the RT execution.

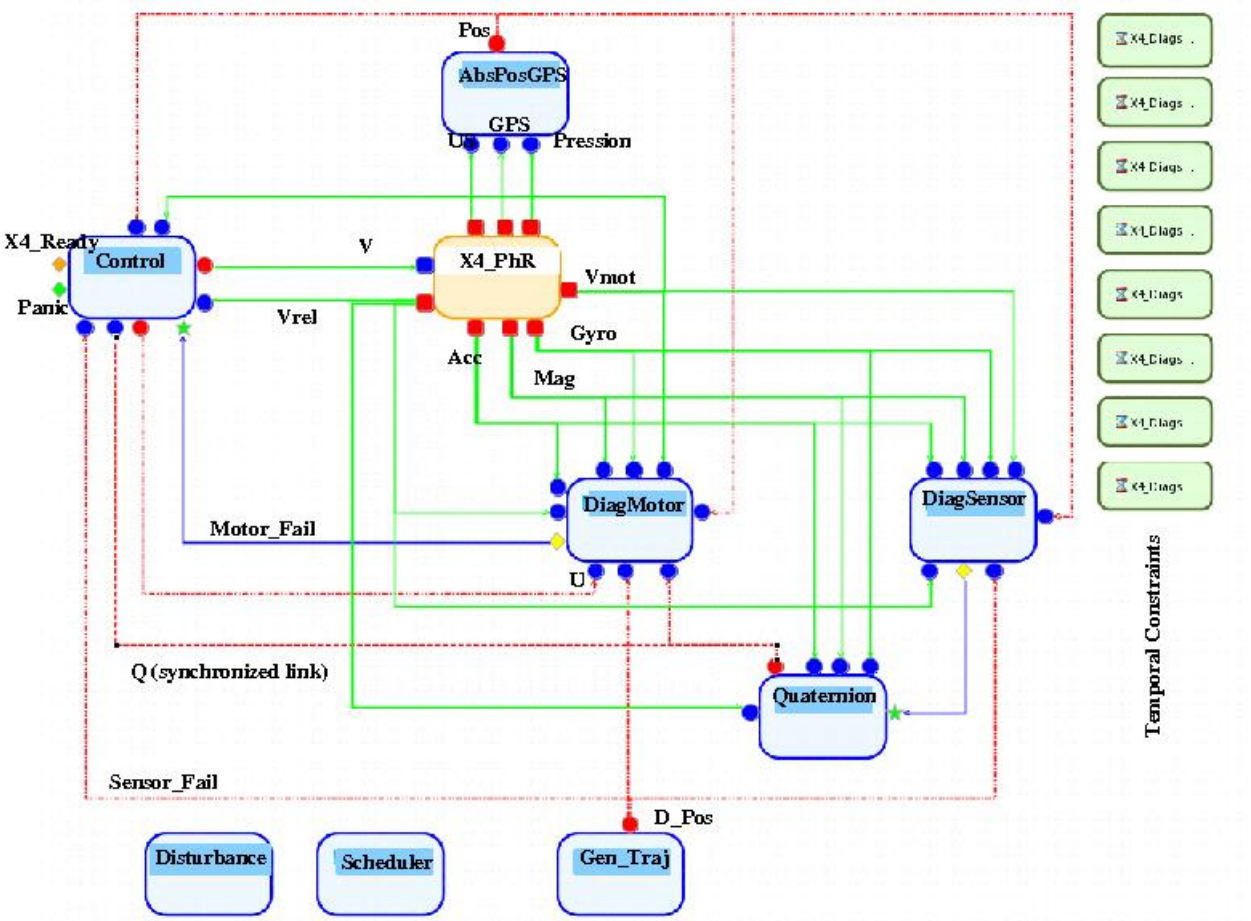

Figure 2: X4 drone control and diagnosis block-diagram.

The blue boxes represent user-provided algorithmic modules (MA) interconnected by their input/output ports. Each module implements a user defined function encoded in C, such as a controllers, filters, ... MA are made of three functions (Figure 3) :

init(...) is executed once when the RT is started. It is used to perform various initialization, e.g. memory allocation or initial values measurement. It can emit events, e.g. a T2 exception to signal a bad or timed-out initialization or a pre-condition to signal a successful completion.

compute(...) repetitively executes the code of the function. It can emit various kind of exceptions to signal particular values of the state of the system, e.g. threshold crossing, or a post-condition to signal a normal termination.

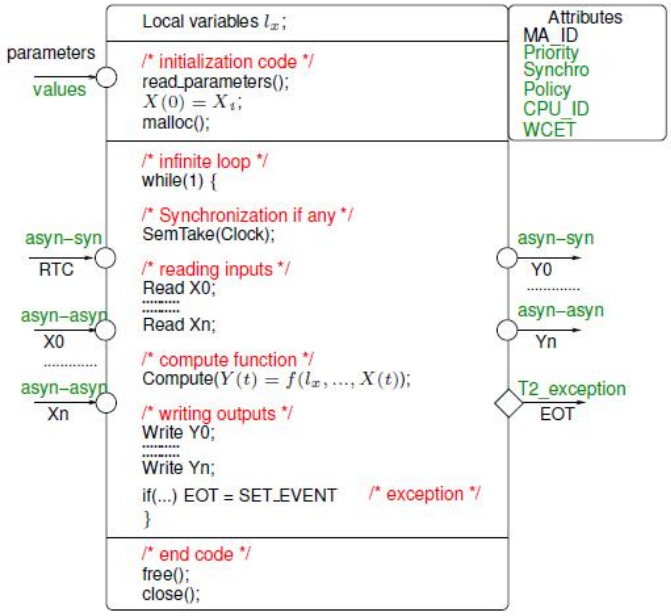

Figure 3: Algorithmic module structure.

end() is executed once at the end of the RT to cleanly close the function.

The input parameters (signature) of the init(...) and compute(...) functions correspond with the input ports of the module. The function may also receive or update parameters on its parameters ports. Events ports are used to specify the logical events(exceptions and conditions) which can be reported by the module during its execution.

A physical resource module (PhR in yellow) represents the physical process to which the controller is connected. It is in fact a gateway to the physical components of the robotic system (actuators and sensors), and its ports allow to call the drivers to connect with the robots.

3.2 Real-time features

From the real-time point of view, each module is own by a TC, i.e. by a real-time thread with a mandatory synchronization input. Each TC may have its own programmable clock.

For simple real-time schemes all the MAs of a controller may share a single TC, thus run in sequence at the same rate. In more complex run-time schemes a one-to-one mapping between MAs and TCs allows every module to be run asynchronously at its own (and possibly varying) sampling frequency. For real-time efficiency several MAs, e.g. which have a strong dependence, can be gathered in the same TC. Also, besides clock triggering, some TC execution ordering can be enforced using synchronization between an output port of a TC and the connected input port of the next TC in the control path.

The TC priorities are set according to their relative importance. Data integrity between asynchronous modules is provided by asynchronous lock-free buffers (Simpson (1997)) which are au-

tomatically inserted at code generation time.

It is well known that software execution usually show duration variations which are difficult to accurately predict. Moreover systems dimensioned according to worst-cases execution times are over-sized and over-constrained. On the other hand system's setting based on more realistic average execution times lead to occasional overruns and deadlines miss. An Execution Flag is set to every TC to control the behavior of the real-time thread in case of overrun according to a pre-defined exception policy. For example the "SKIP" policy leave the running thread to run to completion but skips its next execution.

3.3 Control path

The attitude control path starts from the drivers of the quadrotor sensors (accelerometers Acc, rate gyros Gyr and magnetometers Mag), which are the outputs of the interface module *X_PhR*. The box *X4_PhR* (where PhR stands for Physical Resource) is the interface between the controller and the device to control : sending or reading data on its ports actually calls the drivers, i.e. the functions used to interface the real-time controller with the real hardware, or with the real-time simulator.

The raw measurements are used by the Quaternion module *Quaternion* to estimate the drone attitude using a non-linear observer (Guerrero-Castellanos (2008)). The estimated attitude quaternion Q is forwarded to the *Control* module to perform the attitude control. The computed motor desired velocities are then sent to the quadrotor via the V driver port.

Provision is given for future enhancements of the sensor set, since a GPS-like position sensor and ultra-sonic sensors were expected to be integrated in an enhanced version. Therefore, a trajectory generator module *GEN_Traj* and a position estimator module *AbsPosGPS* are integrated in the control architecture to evaluate position control.

As this control path is critical for the attitude control stability, it is necessary to minimize the loop latency between the raw sensors measurements and the application of the corresponding control to the actuators. Therefore the *Quaternion* and *Control* modules are synchronized via the Q data link. In that case both threads have the same priority, and the *Control* thread has no real-time clock, it inherits the Quaternion thread execution rate thanks to the synchronization.

3.4 Diagnosis and fault tolerant control

To implement diagnosis and fault-tolerant control the drone controller uses the weak exception mechanism (T1) provided by the ORCCAD model to modify the behavior of an algorithmic module (e.g. via a conditional jump in the function code) without switching for a new RT.

The *Diag_Capteurs* module runs the diagnostic algorithm that isolates sensors failures. A failure is signaled by the *Sensor_Fail* weak (T1) exception which is forwarded (with a numerical value to precisely identify the failure) to the *Quaternion* module on a parameter port, so that the quaternion estimation algorithm can be adapted according to the reported failure.

Similarly, the *Diag_Motors* module forwards motor failures signaled by the *Motor_Fail* event to a parameter port of the *Control* module, leading to switch for a pre-defined control degraded mode.

3.5 Scheduling controller

The ORCCAD run-time API and library provides some user-level functions to observe and manage on-line the system scheduling parameters. For example, SetSampleTime() resets on-the-fly the sampling interval of the clocks used to trigger TCs, and MTgetExecTime() records the execution time (cycles) used by a given TC since the RT started[2].

Using this API, the *Scheduler* module performs on-line management of the scheduling policies used to execute the controller, for example to implement and test feedback schedulers : it monitors the controller's real-time activity and may react by setting on-the-fly the task-scheduling parameters, e.g. their firing intervals. For example, such *Feedback Schedulers* can been used to implement a (m,k)-firm dropping policy (Jia et al. (2007)), to dynamically adapt the priorities of messages on the CAN bus (Juanole et al. (2008)) or to implement varying sampling control as in (Simon et al. (2005)). Some of these features are used in some of the following sections 5 and 6.

3.6 Disturbance daemon

A *Disturbance* task allows users to generate extra loads with controlled characteristics either on the CPU or on the CAN bus, specific data corruption on the CAN or Ethernet drivers, and faulty behavior on the sensors or actuators, e.g. to assess the effectiveness of the diagnosis and fault isolation capabilities of the control system. For example it can be used to artificially corrupt data coming from a network to test the robustness of filters w.r.t. data loss (see section 4).

3.7 Socket libraries

The embedded and host computers may communicate via a CAN bus or an Ethernet connection. Two interface functions using sockets libraries have been implemented, so that the driver ports located in the *X4_PhR* interface send and receive data using either the Socket-CAN protocol[3] or UDP sockets on Ethernet.

3.8 Hardware-in-the-loop simulation setup

The utilization of Matlab/Simulink together with TRUETIME remains pure simulation. To provide more realistic results on the influence of the real-time tasks and network on the system, a hardware-in-the-loop experiment has been set up. In the particular case of the quadrotor, hardware-in-the-loop experiments provide a safe environment for the validation of all algorithms and software, prior to any experiments with a real - and very fragile - quadrotor.

3.8.1 HIL Architecture

The goal of the HIL setup consists in realistic assessment of the real control hardware and software (main controller distributed over a network) without using the real drone, i.e. mechanical

[2] the precise semantics and precision of such statement is highly dependent on the underlying RTOS capabilities

[3] `http://developer.berlios.de/projects/socketcan/`

hardware, motors and sensors. It allows for final checks of the control parameters tuning under real time/bandwidth constraints and environmental disturbances without the risk of breaking the real plant. As the real-time simulator can be run before the real process is finished, it may be used also to refine the dimensioning of some components. It can be also used to assess radically new control algorithms such as control under weakly-hard timing constraints. As far as fault tolerant control and safety are concerned, failures can be artificially injected in any parts of the system's component to evaluate the diagnosis algorithms and corresponding recovery handlers.

Therefore the HIL architecture (Figure 5) connects the main control board and real-time control system to a fake system built to mimic the dynamics and behavior of the real drone hardware and software (Figure 4). The fake process runs on a standard Linux system as a set of Posix pthreads :

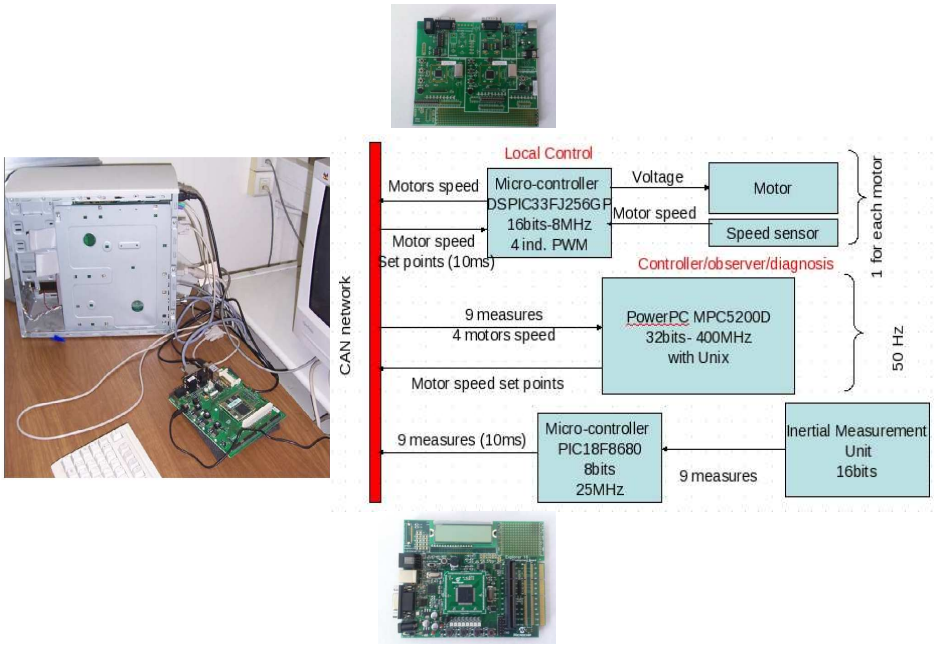

Figure 4: The quadrotor h/w bench-test.

Integration This is the core of the simulated process. The quadrotor dynamic non-linear model is handled by a numerical integrator running on top of Linux as a real-time, high priority thread. The model gathers the drone dynamics in 3D space, the motors internal dynamics and a model of the IMU sensors.

The integration must be fast enough to run the simulated model faster than real-time, and to make negligible the disturbances induced by the computing and networking delays.

Motor control This thread receives the motors velocity set-points from the main controller, the measured actual velocity and periodically computes the voltage to be applied to the motors coils (using the same control law and sampling frequency as the one used on the real dsPiC controller).

Communication This thread handles the communications between the main control board and the simulated physical process. As in the real experiments the communication link can be

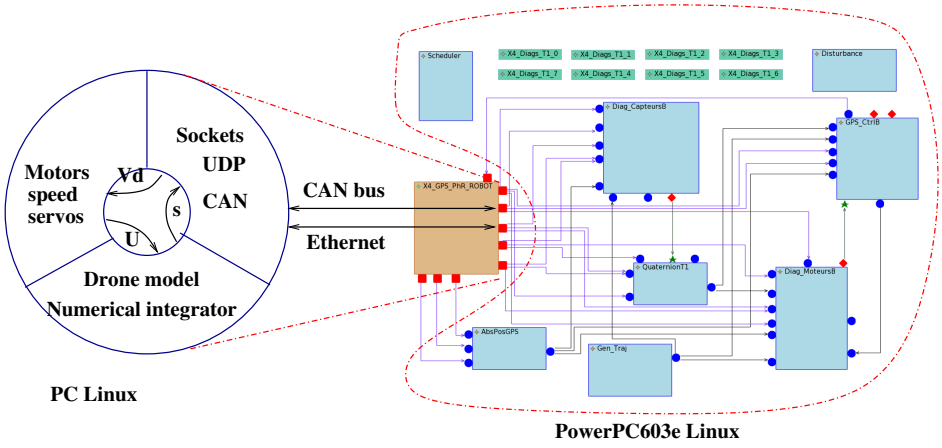

Figure 5: Hardware-in-the-loop structure.

either a CAN bus or an Ethernet link, thus two interfaces have developed based on CAN sockets and UDP sockets libraries (which thanks to the socket abstraction are very similar). This thread handles data packaging and big/little endian conversions.

3.9 Choice of the Numerical Integrator

The non-linear model of a robot (or of most mechatronic systems involving mechanical and electrical components) is usually described by a set of ordinary differential equations (ODEs) (or sometimes by a set of differential-algebraic equations (DAEs)) (Arnold et al. (2011)). There already exist a broad family of algorithms and corresponding software used to numerically solve such a set of ODEs.

To summarize the methods at hand, finding a numerical approximation of the state $X(t)$ of a system of ODEs $\frac{dX}{dt} = f(X,t)$ can be done iteratively with a time step $h_n = t_n - t_{n-1}$ using :

explicit methods X_n is computed only from the last value X_{n-1} (single step) of past values $X_{n-1}, ..., X_{n-k}$ of the solution. Explicit methods are fast but are only conditionally stable (according to the integration step) for linear systems.

implicit methods need to solve an equation $X_n = \mathcal{F}(X_n, X_{n-1}, ..., X_{n-k})$ to find the current solution at time n. They involve more computations than explicit methods (and are slower for a given step value) but they are unconditionally stable for linear systems.

For non-linear systems, no-one is unconditionally stable but implicit methods are more robust than the explicit ones, and diverge slower from the theoretic solution, especially when the system to be integrated is stiff. In both cases, the precision of the solution is mainly based on the order of the algorithm, i.e. the degree of the neglected terms of the Taylor's expansion involved in the computation : higher the order better the precision and higher the computing cost.

Another factor that influences the stability, the precision and the computing cost of the integration algorithm is the integration step. Indeed stability and precision depends both of the step size and of the time constants of the system. Systems with fast dynamics requires smaller step size

and hence higher computation costs. Adaptive step algorithms allows to dynamically adapt the step size to the current system's dynamic which may vary with time and state space location. In that case the computation cost is high only when necessary, but the number of step to perform the integration for a given horizon becomes unpredictable.

With most existing integration packages, the end-user's choice can be :

- set the value of a fixed-step integrator, leading to a predictable number of steps and computing time, but the precision cannot be controlled and may lead to false results;
- specify the precision of the integration and delegate the adaptive step management to the integrator, but in that case the computing time cannot be predicted.

A possible way to provide a predictable and safe simulation time would be using a fixed step, implicit method integrator providing unconditional stability (Ascher and Petzold (1998)). In that case the integration step should be set at a value small enough to insure the required accuracy all along the system's trajectory. Choosing a larger step may lead to poor precision and, for non-linear plants, unstability or convergence towards a false result, e.g. a local minimum far to the real solution.

Conversely, with an adaptive step integration method the controlled variable is the precision (or more exactly the estimate of the integration error). The lsoda package we have elected (Addison et al. (1991)) for the this experiment uses a variable step, variable number of steps algorithm to improve simulation speed and avoid possible numerical unstability.

For a given integration accuracy the simulation speed relies upon the fastest time constant of the model and the simulation speed cannot be guaranteed to be real-time (but the user is warned in case of deadline miss). The software uses two integration methods : an explicit Adams method is used at starting time and when the system is considered to be non-stiff. The software automatically switches to an implicit (Backward Differentiation Formula) when an on-line test estimates that the system becomes stiff and thus that the implicit integration method becomes faster than the explicit one (and switches back conversely). The variable step method speeds up the integration by increasing the step when possible, e.g when the system exhibits slow transients. Finally the explicit method is chosen when it is faster than the implicit method.

Benchmarks and comparisons with renowned methods, e.g. Runge-Kutta 4-5, have shown that despite the complexity of the integration package, the starting overhead is quite small and largely compensated by the efficiency of the method as far as the integration horizon is large enough. Therefore, we can consider that this software implements a method which is almost always faster than the potentially predictable fixed step implicit method, and that this choice consists in the best effort to run the simulation as fast as possible **given a required precision.**

3.9.1 Numerical integrator synchronization

The integration thread is triggered by data requests or arrivals from/to either the network (real system) or the other threads of the fake system. At each triggering event the integrator is first run from the last event to the current time to update the state of the drone, then it delivers the new state or accept new inputs.

Therefore even if the integration is always performed late w.r.t. real-time, the whole simulation process is driven by the events coming from the main real-time controller or by the other real-time threads of the fake system, so that it is globally synchronized with real time.

The systems works pretty well provided that the computation time needed to integrate from the last event is kept small, and in particular the integration step should be completed before the occurrence of the next request. The distortions from reality come from the added delay between a state observation request and the answer delayed by the integrator computing time. This delay depends upon the model's fastest time constant, on the integration algorithm and on the host computer speed.

In practice, for the current set-up the simulation very easily runs comfortably faster than real-time, and the bottleneck comes from the low bandwidth of the CAN bus (which is not simulated, this limitation exists both in the HIL set-up and in the real system, there is no distortion here). Note that computation times and overruns can be easily checked and reported to provide information and warnings about the timing behavior and quality of the simulation.

For some cases, it could be interesting (or even necessary) to drive the integrator from events coming from the process model itself rather than from external triggers such as real-time clocks. For example, when simulating gas engines, events of interest are the ignition instants which are linked to the crank position, i.e. the process state, rather than to time. This could be also the case for incremental encoders for robot arms links. In that case a *root finding* capability should be used to cleanly stop the integration at the point of interest. Such capability could be also be used to integrate in advance w.r.t real-time, thus reducing the risk for overruns.

Finally, if real-time simulation cannot be achieved, or if it is wanted that the simulation runs "as fast as possible" without reference to real-time, another synchronization scheme could be considered. Rather than synchronize the integrator by the real-time controller, the control software could be triggered by the "end-of-integration" events, thus minimizing the processors idle time. However care should be taken to run and synchronize all the simulation components on a coherent time scale.

4 Experiments of Extended Kalman Filters subject to inputs loss

Here is considered a Kalman Filter implementation (Verhaegen and Verdult (2007)) when measurements are transmitted through a network. Most existing works (e.g. (Sinopoli et al. (2004)) (Hespanha et al. (2007)) (Schenato et al. (2007)) (Huang and Dey (2007)) (Sun et al. (2008)) (Epstein et al. (2008))) assume that the output vector $y_k \in \Re^m$ is sent in a unique packet that may be lost. Therefore, when a packet is dropped, all the measurements at sample k are lost. This is for example the case of Ethernet where all the measurement vector of the drone (i.e. accelerometers, magnetometers, gyrometers and motor velocities)) can be send as double precision floating points data in a single UDP packet.

4.1 Kalman filter with data loss

Consider the linear discrete-time varying stochastic system

$$x_{k+1} = \Phi_k x_k + \upsilon_k \quad (1)$$
$$y_k = C_k x_k + \nu_k \quad (2)$$

where $\Phi_k \in \Re^{n\times n}$, $C_k \in \Re^{m\times n}$. υ_k and ν_k are two mutually independent sequences of independent and identically distributed (i.i.d.) Gaussian white noises with covariance matrices Q_k and R_k respectively. Moreover, the pair (Φ_k, C_k) is supposed to be observable and the pair $(\Phi_k, Q_k^{1/2})$ controllable. With our reference example the packet loss can be modeled with an i.i.d. Bernoulli binary random sequence (Sinopoli et al. (2004)) (Schenato et al. (2007)). This model is chosen for mathematical tractability. The stability of the Kalman Filter is proved via a Modified Algebraic Riccati Equation (MARE). The Kalman Filter equations become

$$\widehat{x}_{k+1/k} = \Phi_k \widehat{x}_{k/k} \quad (3)$$
$$P_{k+1/k} = \Phi_k P_{k/k} \Phi_k^T + Q_k \quad (4)$$
$$\widehat{x}_{k+1/k+1} = \widehat{x}_{k+1/k} + \gamma_{k+1} K_{k+1}(y_{k+1} - C_{k+1}\widehat{x}_{k+1/k}) \quad (5)$$
$$P_{k+1/k+1} = P_{k+1/k} - \gamma_{k+1} K_{k+1} C_{k+1} P_{k+1/k} \quad (6)$$

Φ^T denotes the transpose of matrix Φ. $\gamma_{k+1} = 1$ (0 respectively) indicates that y_{k+1} has (has not respectively) been received.

$$K_{k+1} = P_{k+1/k} C_{k+1}^T [C_{k+1} P_{k+1/k} C_{k+1}^T + R_{k+1}]^{-1} \quad (7)$$

is the gain of the Kalman Filter. Matrix P, representing the observer state covariance, is now a random variable because of the randomness of γ. This gives rise to a new Kalman Filter with partial data loss which is now presented (borrowed from (Lesecq et al. (2009)) and (Aubrun et al. (2010))).

4.2 Kalman Filter with partial data loss

However, depending on the topology of the system, on the protocol used, etc., measurements can be sent independently (or at least in several sets). In particular it is the case of a CAN bus where packets contain a preamble and at most 8 bytes of data. With such a protocol, the sensors outputs must be converted to short integers and encapsulated (with decoding flags and timestamps) in individual packets. Therefore, each measurement signal (or subset of measures) is likely to be lost separately at each sample.

Let us now assume that each measurement y_k^i is to be sent independently, and so the Kalman Filter is modified in order to deal with partial data loss (borrowed from (Lesecq et al. (2009)) and (Aubrun et al. (2010))).

In the sequel, the lost measurement is replaced with zero (the alternative replacement by the last available value of the missing input has be tested as well), and the standard deviation of the

associated noise is set to an arbitrarily large value. Consider matrix $M_k \in \Re^{m \times m}$ defined as follows

$$M_k(i,j) = 0 \quad \text{if } i \neq j \tag{8}$$
$$M_k(i,i) = 1 \quad \text{if the measurement is present} \tag{9}$$
$$M_k(i,i) = \lambda_i \gg 1 \quad \text{if the measurement is not received} \tag{10}$$

Denote $\bar{R}_k$ the covariance matrix of the noise of the measurement vector when some elements of $y_k = [y_k^1, \cdots, y_k^i, \cdots, y_k^m]^T$ may be lost. $\bar{R}_k$ can be expressed with

$$\bar{R}_k = M_k R_k M_k \tag{11}$$

Therefore, if y_k^i is not received, its associated noise variance becomes $\bar{\sigma_i}^2 = \lambda_i^2 \sigma_i^2$. The Kalman Filter formulation can now be derived. Note that there is no change in the filter structure, contrarily to the work in (Cao et al. (2009)) where the lines in matrix C corresponding to lost data are removed.

Initialization : initial state x_0, initial covariance matrix $P_0 = \lambda I$ where I is the Identity matrix and λ is large.

Prediction : the prediction step is unchanged

$$\widehat{x}_{k+1/k} = \Phi_k \widehat{x}_{k/k} \tag{12}$$
$$P_{k+1/k} = \Phi_k P_{k/k} \Phi_k^T + Q_k \tag{13}$$

Computation of the Kalman filter gain :

$$\bar{K}_{k+1} = P_{(k+1)/k} C_{k+1}^T [C_{k+1} P_{(k+1)/k} C_{k+1}^T + \bar{R}_{k+1}]^{-1} \tag{14}$$

Correction :

$$\widehat{x}_{k+1/k+1} = \widehat{x}_{k+1/k} + \bar{K}_{k+1}(y_{k+1} - C_{k+1}\widehat{x}_{k+1/k}) \tag{15}$$
$$P_{k+1/k+1} = P_{k+1/k} - \bar{K}_{k+1} C_{k+1} P_{k+1/k} \tag{16}$$

Equation (14) can be reformulated using $\bar{C}_{k+1} = T_{k+1} C_{k+1}$ where $T_{k+1} = M_{k+1}^{-1}$

$$\bar{K}_{k+1} = K_{k+1} T_{k+1} \tag{17}$$

Therefore

$$P_{k+1/k+1} = P_{k+1/k} - P_{k+1/k} \bar{C}_{k+1}^T [\bar{C}_{k+1} P_{k+1/k} \bar{C}_{k+1}^T + R_{k+1}]^{-1} \bar{C}_{k+1} P_{k+1/k} \tag{18}$$

4.3 Experiments of the Kalman filter with missing inputs

Several implementations of the Extended Kalman filter have been encoded and tested. Up to now the best one (from the stability and robustness w.r.t. initial conditions) uses U-D factorization for the prediction step (as specified in (Bierman (1976)) and detailed in (Ford (2002)) with $P_{k+1/k}$ computed as UDU^T with U is unitary upper triangular and D is diagonal, and a stabilized form for the correction step (Gleason (1996)) where the basic form of eq. 16 is rewritten as :

$$P_{k+1/k+1} = (I - K_{k+1}C_{k+1})P_{(k+1)/k}(I - K_{k+1}C_{k+1})^T + K_{k+1}R_{k+1}K_{k+1}^T \tag{19}$$

Compared with the basic implementation the end-to-end execution times are comparable, but the filter converges faster and is weakly sensitive to initial values. Figure 6 compares the output of the non-linear observer and of the Kalman filter for a given motion of the quadrotor with no data loss.

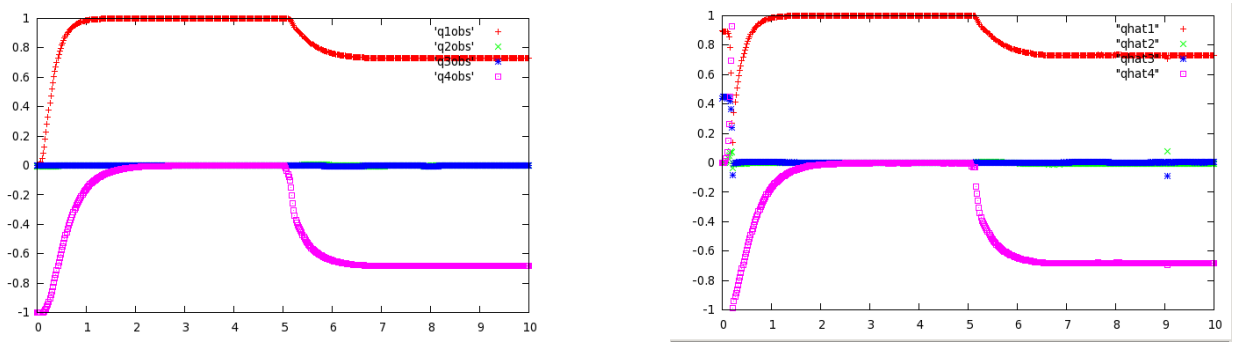

Figure 6: left: non-linear observer – right: extended Kalman filter

Several scenario and strategies for data loss robustness have been experimentally studied using the HIL simulator of the drone. To keep a stable reference, it is assumed that the non-linear observer is fed with a perfect link and do not suffer from data loss, so that the real motion and the reference attitude estimation q_{obs} is always the one given by Figure 6 left. The degradation of the attitude estimation q_{ekf} given by the Kalman filter subject to data loss is computed by the sum over the experiment duration K of the squared errors between the reference and the estimated quaternion components :

$$AEE = \sum_{k=0}^{k=K} (\sum_{i=1}^{i=4} (q_{obs}(i) - q_{ekf}(i))^2)(k)$$

For all experiments, the sampling period of the quaternion estimators and attitude control modules have been set to 50 msecs (avoiding excessive oversampling). With no data loss there is a small difference between the NL and EKF observers leading to a reference error $AEE_0 = 0.072$. Data loss are pseudo-randomly generated at the input of the Kalman filter using the random() function of the standard C library. The following results were made under several combinations of parameters, for each set series of simulations were made with probabilities of one packet loss for each measurement interval ranging from 0 to 99% :

All_data_loss It is assumed that the whole measurement vector is lost for the considered measurement (i.e. all measurements were transmitted in a single packet);

Single_data_loss It is assumed that only one randomly chosen measure is lost for the considered measurement interval; (i.e. there is one different packet for each of the 9 sensors output).

Zero_hold In case of missing measurement the corresponding input is set to 0;

Last_data_hold In case of missing measurement the last available value of the corresponding sensor is hold;

Pk_hold In case of missing input, the correction step is made according to eqs 5 and 6 (which actually means that this step is skipped and that the state covariance matrix P and estimation $\hat{x}$ are only updated during the prediction step);

Rk_updated In case of missing input, the correction step is made according to steps 8 to 18, where the corresponding element of the covariance matrix of the noise of the measurement vector is set to $\lambda_i \gg 1$.

For example the combination [All_data_loss/Zero_hold/Pk_hold] correspond with the assumptions of the reference work of (Sinopoli et al. (2004)) given in section 4.1 and the combination [Single_data_loss/Zero_hold/Rk_updated] correspond with the modified filter of (Lesecq et al. (2009)) given in section 4.2. In addition to these published modified filters, holding the last available value of missing measurements in an appealing alternative (Schenato (2009)), in particular when the values of the missing data is far from 0. Note that in the drone example neither the raw sensors values nor the attitude quaternion components can be all simultaneously null, which makes this option worth to be tested.

In both cases the modified steps in the EKF computation (w.r.t. the standard filter with no data loss) only impact the correction step and Kalman gain computation. However, the gyrometers measurements are used in the prediction step of this particular attitude estimation filter (as detailed in (Berbra et al. (2010))). In case of packet drops they can be lost as well as the others components of the measurement vector. In case of missing gyrometers data, the last available gyrometer value is hold at the filter input.

The data plotted in the following tables and figures were given by averaging the results of a handful of experiments for each parameters set. Note that the two cases, All_data_loss and Single_data_loss cannot be directly compared as, for a given loss ratio, the amount of successfully transmitted data are not at all the same.

On this particular example, we observe that the two modified filters behave similarly when the whole sensors vectors is lost. When sensory data are individually lost, the solution of section 4.1 behaves better than the one of section 4.2, i.e. it is better to just skip the correction step of the Kalman filter rather than computing the correction with a disturbed R_k covariance matrix (but the gyrometers models are supposed almost noiseless and unbiased in this experiment).

Surprisingly using 0 or the last available transmitted value as input for missing channels gives similar results. In all cases these modified Kalman filter appear to be remarkably robust w.r.t.

loss ratio	Pk_hold Zero_hold	Rk_up zero_hold	Pk_hold Last_hold	Rk_up last_hold
20%	0.08	0.077	0.08	0.078
40%	0.09	0.088	0.09	0.09
60%	0.11	0.115	0.115	0.11
70%	0.16	0.17	0.17	0.17
80%	1.37	1.44	1.4	1.37
90%	3.9	4.2	3.7	3.8
95%	13.4	14.2	13.4	13.9

Table 1: Case 1 : AEE for the All_data_loss case

loss ratio	Pk_hold Zero_hold	Rk_up zero_hold	Pk_hold Last_hold	Rk_up last_hold
20%	0.08	0.17	0.086	0.17
40%	0.083	0.26	0.08	0.27
60%	0.11	0.36	0.10	0.35
80%	0.082	0.55	0.085	0.55
90%	0.096	0.48	0.105	0.63
95%	0.15	0.70	0.14	0.7
99%	0.25	0.56	0.25	0.82
99.9%	1.05	0.55	1.10	0.56

Table 2: Case 2 : AEE for the Single_data_loss case

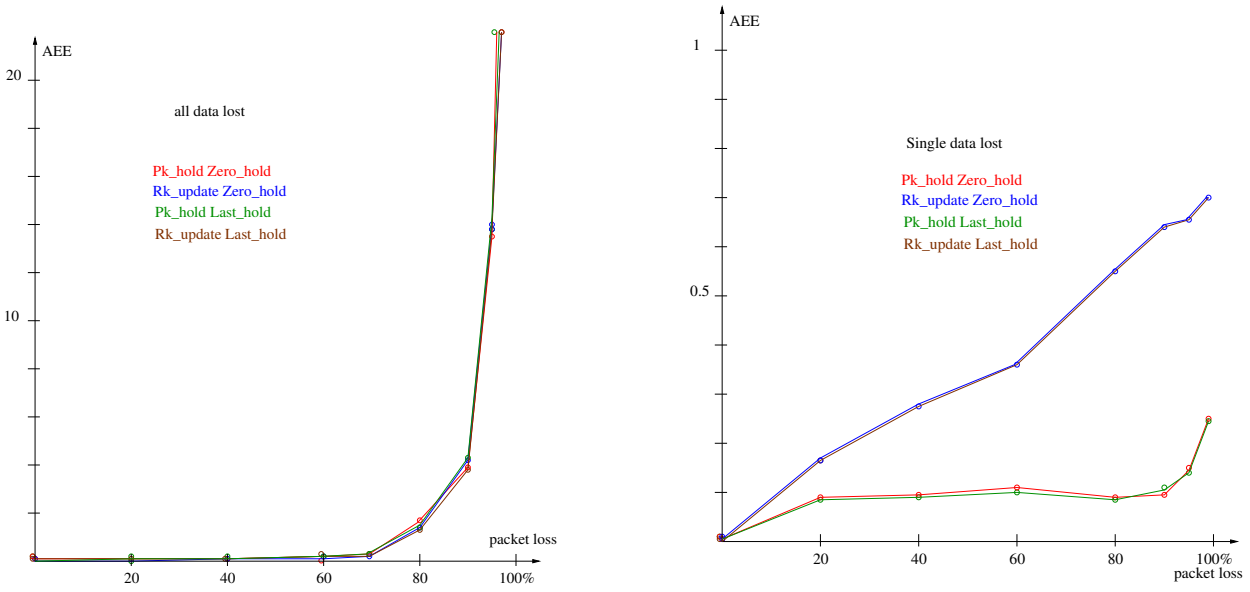

Figure 7: left: all measures lost – right: one random measure lost

input data loss. Anyway it is not possible to provide any general conclusion from this small set of experiments run on a particular example, in particular more realistic assumptions for sensors related noise and bias need to be further investigated.

5 Implementation of control under (m,k)-firm constraints

During system and network overload periods, excessive delay or even data loss may occur. To maintain the quality of control of an NCS, the implementation system (including both computer and network) overload must be correctly handled. As we can see in the previous chapters, a common approach to dealing with this overload problem is to dynamically change the sampling period of the control loops. In this chapter, as an alternative to the explicit sampling period adjustment, we present an indirect sampling period adjustment approach which is based on selective sampling data dropping according to the (m, k)-firm model (Hamdaoui and Ramanathan (1994)). The interest of this alternative is its easy implementation despite having less adjustment quality, since only the multiples of the basic sampling period can be exploited. Upon overload detection, the basic idea is to selectively drop some samples according to the (m, k)-firm model to avoid long consecutive data drops. The consequence is that the shared network and processor will be less loaded. However, the control stability and performance must still be maintained to an acceptable level. This can be achieved by keeping either the total control tasks on a same processor or the messages sharing a same network bandwidth schedulable under the (m, k)-firm constraint.

We consider a global control architecture that integrates a set of plants to control, each one being controlled by a discrete controller. We are interested in the deployment of the control application onto limited resources; for example, all the numerical controllers share the same processors or all the plant states sampled by sensors are transmitted to the controller through the same communication architecture. Moreover, we suppose that according to the global state of the plant, a supervisor chooses the current working mode of the global system. In particular, it can stop the control of a plant, start the control of a plant or modify the control strategy of a plant (control law, sampling period, etc.) The consequence of the transition between working modes is the modification of the set of active tasks/messages that can bring about :

- some of them are stopped,
- new tasks are activated or new messages are transmitted,
- the characteristics of tasks may be modified, for example, changing their WCET (Worst Case Execution Time); the characteristics of messages can be modified, for example, their size or the sampling period can be transformed by using a new control strategy.

This means that the scheduling of the messages on a network or the scheduling of the tasks on a processor have to be redefined each time the supervisor modifies the global control mode. The schedulability analysis of a set of tasks or messages subject to hard real-time constraints (all the instances have to meet their deadline) can lead to over provisioning of the resources and this oversizing can be worse in the case of an architecture that implements several working modes as already mentioned before. Moreover, the relationship between the performance of the control and

the scheduling of the activities is not well-known quantitatively; therefore, the identification of the scheduling parameters relies generally on experiments and/or simulations (Simon and Benattar (2005)) that are not exhaustive and so, can not be generalized. So, a feasible scheduling provided by applying the relaxation of timed constraints as proposed in the classical solution does not lead systematically to the optimal performance of the control application.

A new scheduling architecture for handling such configurations, as well as an adaptive technique that makes adjustments on-line for, on the one hand, the (m, k)-firm constraints of activities (data transmission and / or task implementing the control law) and, on the other hand, the parameters of the control laws, has been proposed. By doing so, the global performance of the application is fixed at an optimized level and the schedulability under (m, k)-firm constraints is guaranteed.

Extensive presentations of this approach, i.e. co-design of control loops and computing resources management subject to (m,k)-firm scheduling can be found in (Li et al. (2006)), (Felicioni et al. (2008)) and (Felicioni, Jia, Simonot-Lion and Song (2010)).

5.1 LQG control of the drone

The application of co-design between LQ control and (m,k)-firm scheduling for the drone control and diagnosis under networked induced data loss can be found respectively in (Guerrero-Castellanos et al. (2009)), (Felicioni, Berbra, Gentil and Lesecq (2010)), (Felicioni and Junco (1998)) and (Felicioni (2011)) form which the following models and control algorithms are borrowed.

A linear control law that locally stabilizes the system described by the nonlinear model around zero Euler angles (a hover condition) has been established. Note that non-linearities are second order, therefore it is reasonable to consider a linear approximation.

For hover condition ($\Phi \simeq 0, \Theta \simeq 0, \Psi \simeq 0$), the following dynamical linearized model is obtained, in terms of roll, pitch and yaw angles:

$$\begin{aligned}
\ddot{\Phi} &= \dot{\Theta}\dot{\Psi}\left(\frac{I_{fy} - I_{fz}}{I_{fx}}\right) + \frac{1}{I_{fx}}\tau_{\Phi} \\
\ddot{\Theta} &= \dot{\Phi}\dot{\Psi}\left(\frac{I_{fz} - I_{fx}}{I_{fy}}\right) + \frac{1}{I_{fy}}\tau_{\Theta} \\
\ddot{\Psi} &= \dot{\Phi}\dot{\Theta}\left(\frac{I_{fx} - I_{fy}}{I_{fz}}\right) + \frac{1}{I_{fz}}\tau_{\Psi}
\end{aligned}$$

where $I_{f.}$ and $\tau_{.}$ denotes respectively the inertia and torque around the rotations axis.

Considering the following state vector

$$x^T = (\Phi, \dot{\Phi}, \Theta, \dot{\Theta}, \Psi, \dot{\Psi})^T \tag{20}$$

the linearized system for hover condition yields:

$$\dot{x} = Ax + Bu$$

with

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_z} \end{pmatrix}$$

This model can be decomposed in three independent second-order subsystems, each one has one control (torque) input and the following matrices:

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 \\ b_0 \end{pmatrix}$$

where b_0 can be $\frac{1}{I_x}$, $\frac{1}{I_y}$ or $\frac{1}{I_z}$.

The control objective is to reach $x = (\Phi, \Theta, \Psi)^T = 0$ starting from a close enough initial attitude. It is proposed to use a discrete linear controller such as

$$u_i = u(ih) = Lx_i$$

where h is he sampling interval. It can be decomposed in three decoupled control laws as

$$u_i = [\tau_\Phi \tau_\Theta \tau_\Psi]^T = [L_\Phi L_\Theta L_\Psi] x_i^T$$

In order to guarantee that each instance of the real-time task satisfies its deadline, it is proposed to use a scheduling policy based on a (m,k)-pattern. It means that if only m out of any k consecutive task instances can be schedulable, then only m instances are executed and the other (k-m) instances are rejected. As the discarded instances can be thought of as a sampling period variation, the controlled plant can be thought of as a concatenation of systems in time, and it can be modeled as a Discrete-Time Switched System (DTSS) Felicioni and Junco (1998). Therefore, to avoid the performance degradation or even instability of each controlled system, the parameters of the control laws are on-line adapted to these variations.

The discretization with a sampling time $h_n = t_{i+1} - t_i$ of the open-loop continuous-time plant described by (20), yields the discrete-time linear system (obtained by usual discretization methods)

$$x_{i+1} = \Phi(h_n)x_i + \Gamma(h_n)u_i \quad y_i = C_d(h_n)x_i + D_d(h_n)u_i \tag{21}$$

The closed loop system using a static state feedback controller $u_i = L_n x_i$ is

$$x_{i+1} = \Phi_n x_i + \Gamma_n u_i = (\Phi_n + \Gamma_n L_n)x_i = \Phi_n^{CL} x_i. \tag{22}$$

Considering there is a finite number of sampling periods $h_n, n = 1, \ldots, k$, then there is a family describing the open-loop systems 20, and the problem to be solved is to find a family of controllers $(L_n; n = 1, \ldots, k)$ such that the DTSS closed-loop system 22 is asymptotically stable.

Felicioni and Junco (1998) proposes to choose the controller parameters $(L_n; n = 1, \ldots, k)$ to make solvable the Lie algebra generated by the family of closed-loop DTSS matrices

$M = \Phi_n^{CL}, n = 1, \ldots, k$, which guarantees the existence of a Common Quadratic Lyapunov Function CQLF for the DTSS family. An explicit solution results: let $h_n = n.h$, the adaptive control law parameters are $L_n = [l_{n,1}, L_{n,2}]$ with

$$l_{n,1} = \frac{-4l_{1,1}}{b_0 n^2 h^2 l_{1,1} + 2b_0 nhl_{1,2} - b_0 nh^2 l_{1,1} - 4 - 2b_0 hl_{1,2}} \tag{23}$$

$$l_{n,2} = \frac{-2(nhl_{1,1} + 2l_{1,2} - hl_{1,1})}{b_0 n^2 h^2 l_{1,1} + 2b_0 nhl_{1,2} - b_0 nh^2 l_{1,1} - 4 - 2b_0 hl_{1,2}} \tag{24}$$

5.2 Implementation of the (m,k)-firm scheduled LQ controller

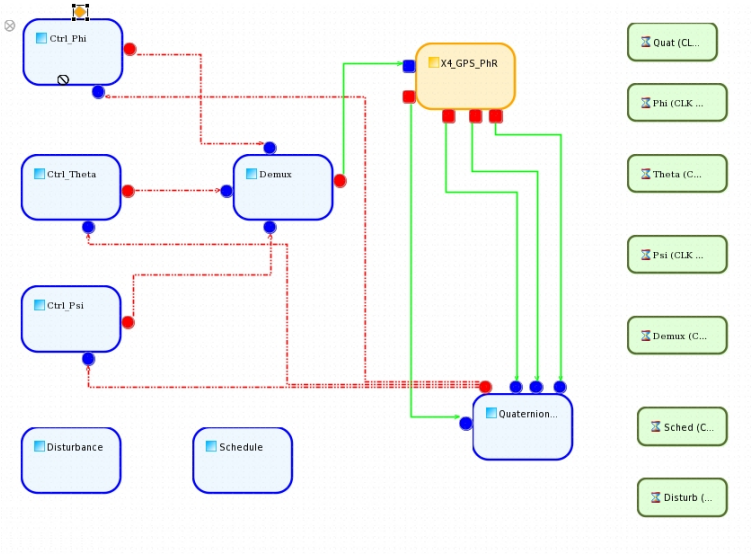

Figure 8: Decoupled LQ drone controller

Figure 9 depicts the new control system which is now split into three modules Ctrl_Phi, Ctrl_Theta and Ctrl_Psi for the Φ (roll), Θ (pitch) and Ψ (yaw) angles control. In order to obtain a suitable transient response we consider a basic sampling time for each subsystem $h_\Phi = h_\Theta = 0.1$ sec. and $h_\Psi = 0.12$ sec. The control law gains for the base period are given by:

$$L_\Phi(h_\Phi) = L_\Theta(h_\Theta) = [-0.02959 - 0.02524]; L_\Psi(h_\Psi) = [-0.03055 - 0.03282]$$

These initial values are set during the init() function of the Ctrl_X modules and they can be evaluated at every activation thanks to eq. 24 according to the current value of the module's sampling interval returned by the GETSAMPLETIME() statement.

The value of k = 5 for each subsystem is chosen according to (Felicioni et al. (2008)). Among others, the easiest way to schedule control-tasks under a (m,k)-firm constraint consists in simply reset their activation interval using the MTSETSAFESAMPLETIME() statement of the Orccad API. The control intervals are managed from the SCHEDULE control task. To implement a feedback scheduler based on CPU time consumption, this same task may evaluated the cost (in time or CPU

cycles) induced by the control tasks execution (using the ORCGETEXECTIME(TASKID)) statement and compute new tasks execution intervals with any suitable algorithm (e.g. period rescaling as in Cervin (2003) or H_∞ design as in Simon et al. (2005)).

Figure 9 show some sample results comparing the drone observed position with all axis controllers running at their nominal period (on the left) and for the Φ and Θ controllers restricted to 1 allowed execution out of 5 with equidistant sampling (on the right), one of the $(m_\Phi, m_\Theta, m_\Psi)$ scheduling configuration known to be stable (even if seriously degraded) Guerrero-Castellanos et al. (2009).

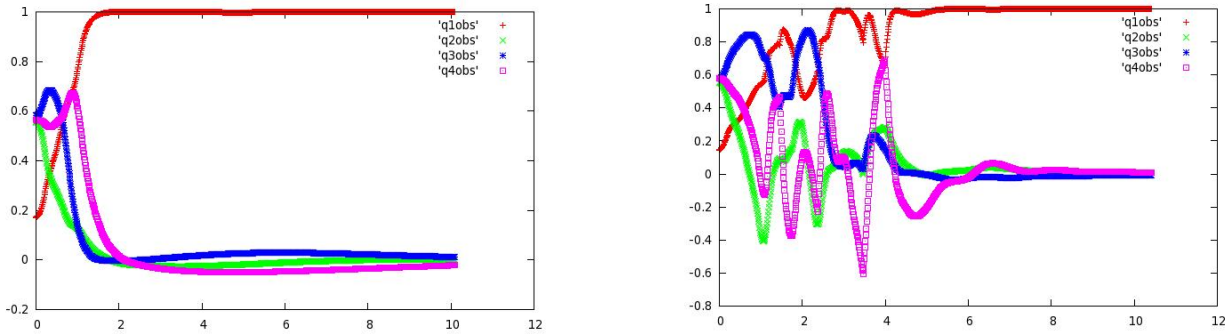

Figure 9: Drone control under (m,k)-firm scheduling : left $m_\Phi = m_\Theta = m_\Psi = 5$ – right $m_\Phi = m_\Theta = 1, m_\Psi = 5$

6 LPV varying sampling controllers

As already stated in a previous report (Roche, Sename, Seuret, Simon and Varrier (2010)), the processing load induced by a real-time control task is defined by $U = \frac{c}{h}$ where c and h are the execution time and period of the task. The impact of control messages over a network occupied bandwidth can be evaluated similarly by $B = \frac{d}{h}$ where d is the message size. Reducing the computing cost of a control task can be done in two ways:

- Increasing the sampling interval h can be easily done in real-time, providing that the control interval remains high enough to preserve the system's stability and that interval switching is also handled from the stability viewpoint;
- Reducing its execution time c, e.g. by using a simplified version of the control algorithm: this requires a control tasks switching, which may be complex to be safely performed in real-time.

Therefore, as stated in (Simon et al. (2009)), the key actuator to be used for CPU utilization or network bandwidth control is the control interval. A first idea is to design a bank of controllers, each of them being designed and tuned for a specific sampling frequency, and to switch between them according to the decisions of the feedback scheduler. However, it has been observed that switching without caution between such controllers may lead to instability although each controller in isolation is stable (Schinkel et al. (2002)).

In (Robert et al. (2010)) the LPV polytopic approach is used to design a control law with adaptation of the sampling period to account for the available computing resources for an inverted pendulum. Indeed the main drawback of the polytopic method could be the large number of LMIs to solve as the number of varying parameters increases. This is not the case in the Linear Fractional Transform (LFT) method and, as emphasized in (Apkarian and Gahinet (1995)), it leads to a LMI problem whose solution can directly be implemented. Moreover, this approach allows to consider in the same way varying parameters and uncertainties (adaptation to the parameters and robustness with respect to uncertainties).

6.1 A general LPV model

The equations defining an LPV system with respect to vector of parameters ρ are:

$$\Sigma(\rho(.)) : \left[\begin{array}{c} \dot{x} \\ \hline z \\ y \end{array}\right] = \left[\begin{array}{c|cc} A(\rho(.)) & B_1(\rho(.)) & B_2(\rho(.)) \\ \hline C_1(\rho(.)) & D_{11}(\rho(.)) & D_{12}(\rho(.)) \\ C_2(\rho(.)) & D_{21}(\rho(.)) & D_{22}(\rho(.)) \end{array}\right] \left[\begin{array}{c} x \\ \hline w \\ u \end{array}\right] \tag{25}$$

where $x(t)$ is the state vector which takes values in a state space $X \in \mathcal{R}^n$, $u(t)$ is the control input $u \in \mathcal{R}^{n_u}$, $w(t)$ is the external input $w \in \mathcal{R}^{n_w}$, $z(t)$ is the controlled output $z \in \mathcal{R}^{n_z}$ and $y(t)$ is the measured output $y \in \mathcal{R}^{n_y}$.Then, $\rho(.)$ is a varying parameter vector that takes values in the parameter space $\mathcal{P}_\rho$ such that:

$$\mathcal{P}_\rho := \Big\{ \rho(.) := \left[\begin{array}{ccc} \rho_1(.) & \dots & \rho_n(.) \end{array}\right]^T \in \mathcal{R}^n$$

$$\rho_i(.) \in \left[\begin{array}{cc} \underline{\rho}_i & \overline{\rho}_i \end{array}\right] \forall i = 1, \dots, n \Big\}$$

where n is the number of varying parameters. $\mathcal{P}_\rho$ is a convex set.

6.2 LFR approach

The LFR formulation is widely used in robust analysis to study the influence of the uncertainties on the stability and performances of a closed-loop system. It can also be used to build a parameter dependent model of a dynamical system, depending on a known set of parameters. In particular, this approach can be used to keep some system non linearities in a LFR model, and then linear control tools can be used to compute a controller scheduled by the parameters (as in(Gauthier et al. (2007))). In this section, the LFR formulation is presented and applied to the case of discrete-time sampling varying modeling and control of LPV systems.

The approach comes from the robust control theory and consists in separating the LTI part P (not depending on the set of parameters) from the varying part Δ (parameters or uncertainties), as shown on figure 10.

The matrix Δ represents the influence of the set of parameters $\rho(.)$ of equation 25 on the plant.

From this model, a gain scheduled controller can be computed, depending on the same set of parameters Δ, or a subset of Δ, as presented in (Apkarian and Gahinet (1995)). Here the

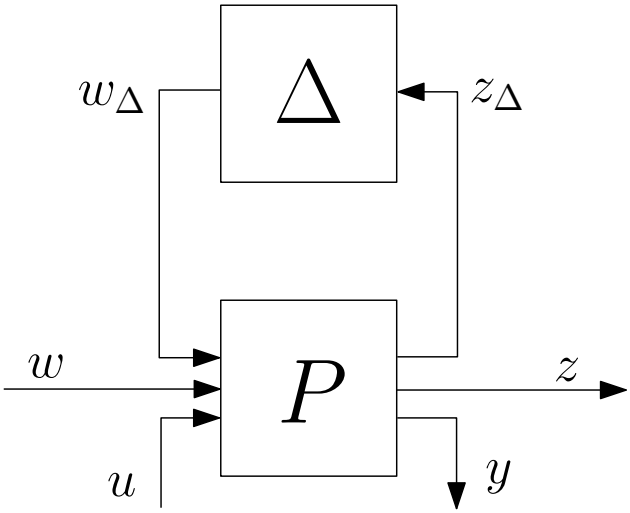

Figure 10: System under LFT form

LFT approach proposed in (Roche, Sename and Simon (2010)) will be extended to set a LFR model that accounts for system and sampling parameters. The steps bellow describe the proposed methodology. During the synthesis of a discrete-time varying sampling controller, depending on system parameters and on the sampling interval, the discretization step is straightforward, since only LTI system (P) needs to be discretized. Moreover, while the controller synthesis relies also on the resolution of LMIs, in this case the number of LMIs to be solved is not dependent on the number of parameters, but only the sizes of the matrices in the LMIs are increasing.

6.3 LFR model depending on system parameters

For a LPV system represented by equation 25 depending on a vector of parameters $\rho(.)$ an equivalent Linear Fractional Representation can be found of the form presented in figure 10, with P a continuous-time LTI plant ($P(s)$) and Δ a block diagonal matrix.

The equation of the system under LFR representation is as follows:

$$\left[\begin{array}{c} \dot{x} \\ \hline z_\Delta \\ z \\ y \end{array}\right] = \left[\begin{array}{c|ccc} A & B_\Delta & B_1 & B_2 \\ \hline C_\Delta & D_{\Delta\Delta} & D_{\Delta 1} & D_{\Delta 2} \\ C_1 & D_{1\Delta} & D_{11} & D_{12} \\ C_2 & D_{2\Delta} & D_{21} & D_{22} \end{array}\right] \left[\begin{array}{c} x \\ \hline w_\Delta \\ w \\ u \end{array}\right] \tag{26}$$

Here the vector of parameters $\rho(.)$ (of equation 25) represents some system parameters to be kept in the model (e.g. some non-linearities, see (Pfifer and Hecker (2008), Lohning (2010), Biannic et al. (2006))).

6.4 Parametrized discretization

As described in (Roche, Sename and Simon (2010)) the LFR formulation can be used to obtain a discrete-time gain scheduled model of a LTI system, with the sampling interval as varying pa-

rameter. The interest of the LFR formulation is that the varying part is separated from the system dynamics. Then the LFR model in figure 10 can be discretized, assuming that the uncertain input/outputs (w_Δ and z_Δ) are sampled and hold at the sampling frequency. The methodology proposed in (Roche, Sename and Simon (2010)) can then be applied in that context to get a sampling dependent LFR discrete-time model.

From the LFR model obtained in previous section (equation 26), with some non-linearities of the model as parameters, a new LFR will be computed, adding the sampling interval to the existing set of varying parameters.

Let us consider the continuous LTI part of the LFR given in equation 26, and define:

$$\begin{aligned} \tilde{A} &= A & \tilde{B} &= [B_\Delta\ B_1\ B_2] \\ \tilde{C} &= \begin{bmatrix} C_\Delta \\ C_1 \\ C_2 \end{bmatrix} & \tilde{D} &= \begin{bmatrix} D_{\Delta\Delta} & D_{\Delta 1} & D_{\Delta 2} \\ D_{1\Delta} & D_{11} & D_{12} \\ D_{2\Delta} & D_{21} & D_{22} \end{bmatrix} \end{aligned} \tag{27}$$

The inputs and outputs are gathered in a single vector as:

$$\tilde{u} = \begin{bmatrix} w_\Delta \\ w \\ u \end{bmatrix} \ and\ \tilde{y} = \begin{bmatrix} z_\Delta \\ z \\ y \end{bmatrix} \tag{28}$$

The sampling period is assumed to belong to the interval $[h_{min}, h_{max}]$ with $h_{min} > 0$, and is approximated around the nominal value h_0 as:

$$h = h_0 + \delta \qquad \text{with} \qquad h_{min} - h_0 \leqslant \delta \leqslant h_{max} - h_0 \tag{29}$$

The exact discretization of the LTI system ($\tilde{A}$,$\tilde{B}$,$\tilde{C}$,$\tilde{D}$) is described in equation (30)

$$\begin{aligned} x_{k+1} &= A_d x_k + B_d \tilde{u}_k \\ \tilde{y}_k &= C_d x_k + D_d \tilde{u}_k \end{aligned} \tag{30}$$

with:

$$\begin{aligned} A_d &= e^{\tilde{A}h} & B_d &= \int_0^h e^{\tilde{A}\tau} d\tau \tilde{B} \\ C_d &= \tilde{C} & D_d &= \tilde{D} \end{aligned} \tag{31}$$

Matrices A_d and B_d depend both on the constant part h_0 and on the varying part δ of the sampling period:

$$A_d = A_{h_0} A_\delta \tag{32}$$

$$B_d = B_{h_0} + A_{h_0} B_\delta \tag{33}$$

Only the matrices A_δ and B_δ depend on the varying part of the sampling interval and shall appear in the varying matrix of the LFR model. As detailed in the appendix, an approximate form of equation (31) w.r.t δ can be obtained by a Taylor series expansion at order k. This Taylor series

expansion leads to an LFR representation depending on the variation of the sampling interval (δ) and the previous parameters (Δ) (see Figure 11) as:

$$P_d \; : \; \begin{cases} x_{k+1} = \mathcal{A}x_k + \mathcal{B}\begin{bmatrix} w_\Delta \\ u_\delta \\ u \end{bmatrix} \\ \begin{bmatrix} z_\Delta \\ y_\delta \\ y \end{bmatrix} = \mathcal{C}x_k + \mathcal{D}\begin{bmatrix} u_\Delta \\ u_\delta \\ u \end{bmatrix} \end{cases} \tag{34}$$

with:

$$\Theta = \begin{bmatrix} \Delta & 0 \\ 0 & \delta I_{2\times k\times n_s} \end{bmatrix} \tag{35}$$

$$\mathcal{A} = A_{h_0} \quad \mathcal{B} = \begin{pmatrix} \mathcal{B}_1 & \mathcal{B}_2 & B_{h_0} \end{pmatrix} \tag{36}$$

$$\mathcal{C} = \begin{pmatrix} \mathcal{C}_1 \\ \mathcal{C}_2 \\ C_d \end{pmatrix} \quad \mathcal{D} = \begin{pmatrix} \bar{\mathcal{D}} & 0 & 0 \\ 0 & \bar{\mathcal{D}} & \mathcal{D}_{2u} \\ 0 & 0 & D_d \end{pmatrix} \tag{37}$$

where k is the order of the Taylor series expansion in δ and n_s is the number of states.

For more details on the derivation of this model, please refer to (Roche, Sename and Simon (2010)) and to the appendix.

6.5 LFR controller

Following the works by (Apkarian and Gahinet (1995)) an H_∞ gain scheduled LFR controller can be computed from this LFR model. This controller depends on the same set of varying parameters Θ (or on a sub-set of this one). The controller will be computed using tools derived from the bounded Real Lemma. Some performance specification are therefore fixed using some weighting functions ($\mathcal{H}_\infty$ design).

The dependence between the discrete-time LTI plant P_d, the controller K, the parameter block (including the system and sampling parameters) and the weighting functions W_i and W_o is presented in figure 11.

In the considered problem, only a single weighting function, W_e, is placed on the tracking error and is defined in continuous time, depending on the sampling interval. This allows the adaptation of the performances with respect to the current sampling period, as explained in (Robert et al. (2010)). So W_e is defined with the following state space system, using the frequency $f = 1/h$:

$$\begin{cases} \dot{x} = (a \times f)x + (a \times f - b \times f)u \\ y = x + u \end{cases} \tag{38}$$

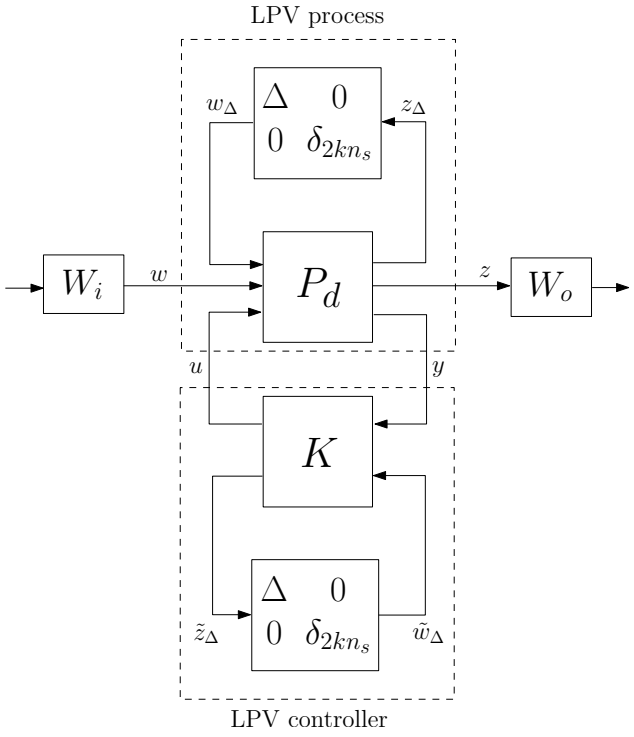

Figure 11: LFR control structure

This weight is then discretized to obtain a discrete-time representation:

$$W_{e_d}(z) : \begin{cases} x_{k+1} = A_d\, x_k + B_d\, u_k \\ y_k = x_k + u_k \end{cases} \tag{39}$$

$$\begin{cases} A_d = e^{af\,h} = e^a \\ B_d = (af)^{-1}(A_d - I)bf = a^{-1}(A_d - I)b \end{cases} \tag{40}$$

The simplification between h and f leads to a discrete-time LTI representation of the weight. This allows the interconnection of the weighting function to the LTI part of the LFR model.

Other weights could be added on the control input to account for actuator saturation, or on the measured output to limit its evolution.

Finally, the LFR controller is computed using the methodology presented in (Apkarian and Gahinet (1995)).

The authors wish to stress that the controller synthesis relies also on the resolution of LMIs (as for the Polytopic case). But in the LFR case the number of LMIs to be solved is not dependent on the number of parameters, but only the sizes of the matrices in the LMIs are increasing (with the increase of the Θ matrix).

6.6 Implementation

The AUV controller is implemented as a set of modules and associated TCs (i.e. real-time threads) as depicted in Figure 12.

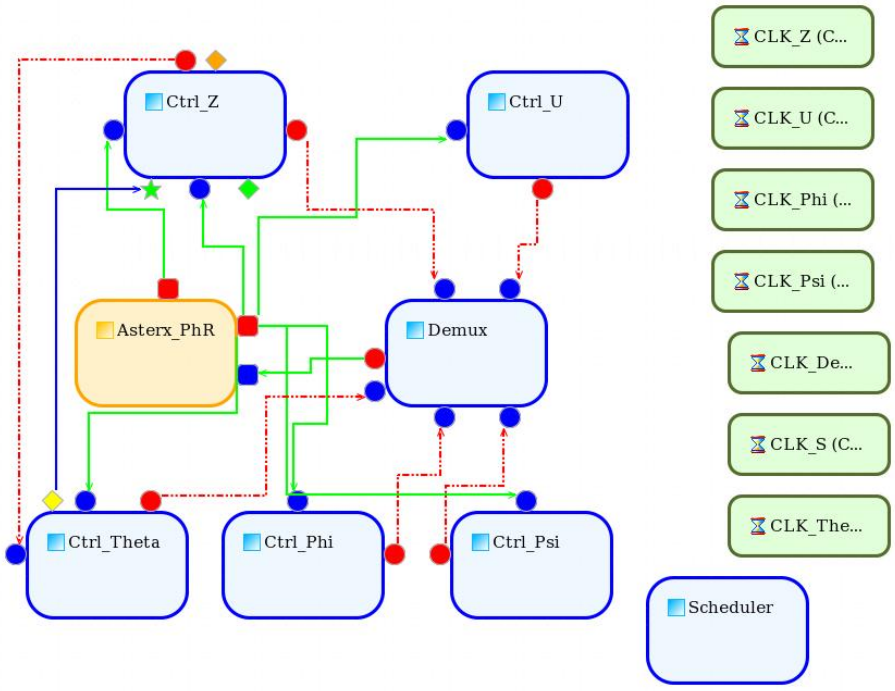

Figure 12: LPV varying control for the Asv.

- CTRL_U implements a classical (i.e. fixed sample rate) $\mathcal{H}_\infty$ controller to ensure a constant velocity despite the disturbances coming from drag forces induced by the control fins (Roche et al. (2011));
- CTRL_PHI and CTRL_PSI implement simple fixed gains P.D. controllers to stabilize the vehicle around the roll and yaw axis;
- CTRL_THETA and CTRL_Z run in cascade to implement the LPV/LFT controller described in section 6.5, coding the structure of Figure 11 and the gain matrices provided when synthesizing the $\mathcal{H}_\infty$ controller with the Matlab Robust Control Toolbox;
- DEMUX gathers the outputs of the previous control tasks and manages the actuators saturations;
- SCHEDULER manages the control interval of the varying sampling controllers (Theta and Z control), using the MTSETSAFESAMPETIME() statement to smoothly change the values of the clocks associated with the CTRL_THETA and CTRL_Z modules;
- ASTERX_PHR gives gateways with a real-time simulator of the AUV, thus providing a HIL architecture similar to the one designed for the quadrotor of Figure 5.

Preliminary tests show that the control system's behavior is compliant with the simulation results previously obtained using Matlab/Simulink (Roche (2011)). However, the simulator is now real-time (while using about 5% of a pentiumM CPU), and varying sampling control can be very easily designed using Orccad's features.

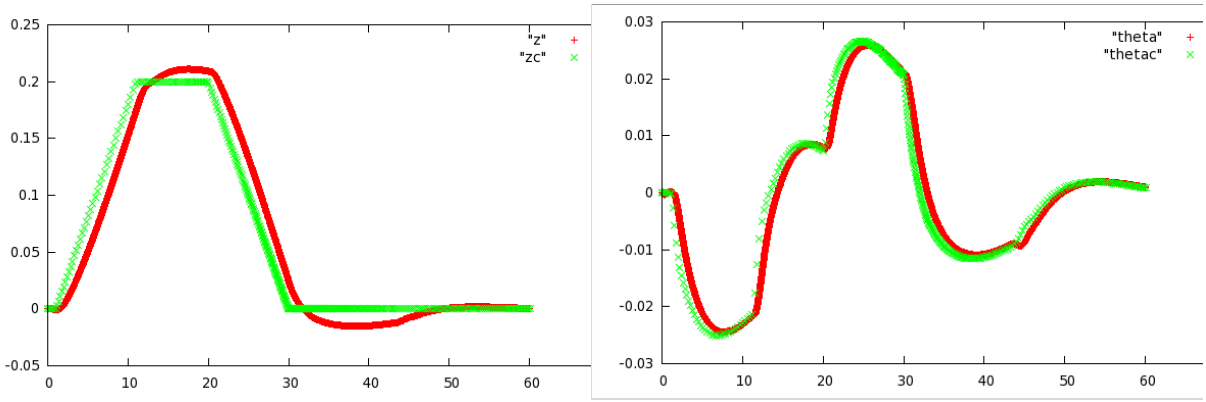

Figure 13: LPV control: altitude and pitch.

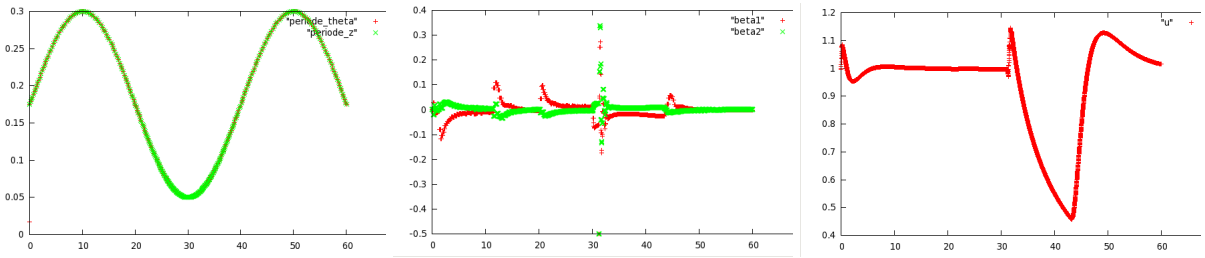

Figure 14: LPV control: control intervals, control fins angles, forward velocity.

7 Combining LPV and MPC two-tier scheme for the AUV control

In this section, we introduce a two-tier model predictive control (MPC) architecture for the an autonomous underwater vehicle application presented in the previous sections. Originally, the two-tier MPC architecture was proposed for systems with both continuous and asynchronous sensing and actuation, and in particular for process control systems based on hybrid communication networks (i.e. point-to-point wired links integrated with networked wired or wireless communication) and using multiple heterogeneous measurements (e.g. temperature and concentration). Nevertheless, it is relevant for the design of controllers of autonomous vehicles for which the communication or the use of certain sensors may be expensive or not possible for all times, for example, in the aforementioned AUV application.

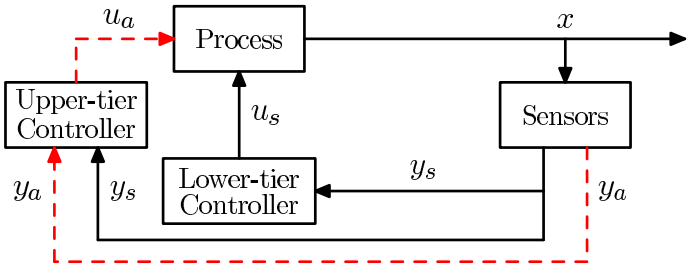

Figure 15: Two-tier control strategy (solid lines denote dedicated point-to-point, wired communication links and continuous sensing and actuation; dashed lines denote networked (wired or wireless) communication or asynchronous sampling and actuation).

Assuming that there exists a lower-tier control system which relies on point-to-point communication and continuous measurements to stabilize the closed-loop system, we propose to use Lyapunov-based model predictive control to design an upper-tier networked control system to profit from both the continuous and the asynchronous measurements as well as from additional networked control actuators. The proposed two-tier control system architecture preserves the stability properties of the lower-tier controller while improving the closed-loop performance. Figure 15 shows a schematic representation of the proposed strategy.

In the control problem considered, the lower-tier controller is the proposed varying sampling control for LPV systems, in particular and without loss of generality, we will use as an example the controller based on the LFR approach. The main idea of the application of a two-tier MPC to the UAV application is twofold, first, the MPC controller can take explicitly into account the actuator limits, and second, it can profit from the use of the model in case there are lost measurements due to interferences or simply because energy want to be saved.

We review next the original two-tier MPC formulation. More details can be obtained in (Muñoz de la Peña Sequedo and Raimondo (2010)).

7.1 Two-tier MPC formulation

We consider nonlinear systems described by the following state-space model

$$\begin{aligned} \dot{x}(t) &= f(x(t), u_s(t), u_a(t), w(t)) \\ y_s(t) &= h_s(x(t)) \\ y_a(t) &= h_a(x(t)) \end{aligned} \tag{41}$$

where $x(t) \in R^{n_x}$ denotes the vector of state variables, $y_s(t) \in R^{n_{y_s}}$ denotes continuous and synchronous measurements, $y_a(t) \in R^{n_{y_a}}$ are asynchronous and sampled measurements, $u_s(t) \in R^{n_{u_s}}$ and $u_a(t) \in R^{n_{u_a}}$ are two different sets of possible control inputs and $w(t) \in R^{n_w}$ denotes the vector of disturbance variables.

System (41) is controlled using both continuous synchronous and sampled asynchronous measurements. We assume that $y_s(t)$ is available for all t, while $y_a(t)$ is sampled and only available at some time instants t_k where $\{t_{k\geqslant 0}\}$ is a random increasing sequence of times. We assume that the measurement of the full state $x(t_k)$ can be obtained by combining measurements $y_s(t_k)$ and $y_a(t_k)$.

The continuous measurement $y_s(t)$ can be used to design a continuous output-feedback controller to stabilize the system. We term the control system based only on the continuous measurements $y_s(t)$ as lower-tier controller. This control scheme does not use the asynchronous measurements $y_a(t)$. Following this idea, we assume that there exist an output feedback controller $u_s(t) = k_s(y_s)$ (where $k_s(y_s)$ is assumed to be a sufficiently smooth function of y_s) that renders the origin of the nominal closed-loop system (i.e., $w(t) \equiv 0$) asymptotically stable with $u_a(t) \equiv 0$.

The main objective of the two-tier control architecture is to improve the performance of the closed-loop system using the information provided by $y_a(t)$ while guaranteeing that the stability properties of the lower-tier controller are maintained. This is done by defining a controller (upper-tier controller) based on the full state measurements obtained from both the synchronous and asynchronous measurements at time steps t_k. In the two-tier control architecture, the upper-tier controller decides the trajectory of $u_a(t)$ between successive samples, i.e., for $t \in [t_k, t_{k+1})$ and the lower-tier controller decides $u_s(t)$ using the continuously available measurements.

The proposed upper-tier LMPC optimization problem is defined as follows:

$$\begin{aligned}
\min_{u_a \in S(\Delta)} \quad & \int_0^{\tau_f} L(\tilde{x}(\tau), k_s(h_s(\tilde{x}(\tau))), u_a(\tau), 0) d\tau && \text{(42a)}\\
& \dot{\tilde{x}}(\tau) = f(\tilde{x}(\tau), k_s(h_s(\tilde{x}(\tau))), u_a(\tau), 0) && \text{(42b)}\\
& \dot{\hat{x}}(\tau) = f(\hat{x}(\tau), k_s(h_s(\hat{x}(\tau))), 0, 0) && \text{(42c)}\\
& \hat{x}(0) = \tilde{x}(0) = x(t_k) && \text{(42d)}\\
& V(\tilde{x}(t)) \leqslant V(\hat{x}(t)) \; \forall t \in [0, \tau_f] && \text{(42e)}
\end{aligned}$$

where $x(t_k)$ is the state obtained from $y_s(t_k)$ and $y_a(t_k)$, $\tilde{x}(\tau)$ is the predicted trajectory of the two-tier nominal system for the input trajectory computed by the LMPC (42), $\hat{x}(\tau)$ is the predicted trajectory of the two-tier nominal system for the input trajectory $u_a(\tau) \equiv 0$ for all $\tau \in [0, \tau_f)$. The optimal solution to this optimization problem is denoted $u^*_{u,a}(\tau|t_k)$. This signal is defined for all $\tau > 0$ with $u^*_{u,a}(\tau|t_k) = 0$ for all $\tau \geqslant \tau_f$.

The control inputs of the proposed two-tier control architecture based on above LMPC are defined as follows:

$$\begin{aligned}
u_s(t) &= k_s(h_s(x(t))), \; \forall t \\
u_a(t) &= u^*_{u,a}(t - t_k|t_k), \; \forall t \in [t_k, t_{k+1})
\end{aligned} \tag{43}$$

where $u^*_{u,a}(t - t_k|t_k)$ is the optimal solution of the LMPC optimization problem (42) at time step t_k.

7.2 AUV two-tier control

In this particular application, the measurements for both the lower-tier and the upper-tier controller are sampled, moreover, the sampling time is variable. Nevertheless, the two-tier architecture, will provide robustness for cases in which the measurements delay, are lost or simply are not made. Another consideration, is that the model of the system and the LFR controller is linear. This will allow us to define a QP problem to define the MPC control input.

The MPC controller needs a state space representation of the closed-loop system. In the first draft we assume that the following model is available.

$$\begin{bmatrix} x_p(k+1) \\ z_\Delta(k) \\ z(k) \\ y(k) \end{bmatrix} = P \begin{bmatrix} x_p(k) \\ w_\Delta(k) \\ w(k) \\ \tilde{u}(k) \end{bmatrix}$$

$$\begin{bmatrix} x_c(k+1) \\ \tilde{z}_\Delta(k) \\ u(k) \end{bmatrix} = K \begin{bmatrix} x_c(k) \\ \tilde{w}_\Delta(k) \\ y(k) \end{bmatrix}$$

with

$$P = \begin{bmatrix} A^p & B^p_\Delta & B^p_w & B^p_u \\ C^p_\Delta & D^p_{\Delta\Delta} & D^p_{\Delta w} & D^p_{\Delta u} \\ C^p_w & D^p_{w\Delta} & D^p_{ww} & D^p_{wu} \\ C^p_y & D^p_{y\Delta} & D^p_{yw} & D^p_{yu} \end{bmatrix}$$

$$K = \begin{bmatrix} A^c & B^c_\Delta & B^c_y \\ C^c_\Delta & D^c_{\Delta\Delta} & D^c_{\Delta y} \\ C^c_u & D^c_{u\Delta} & D^c_{uy} \end{bmatrix}$$

$$w_\Delta(k) = \Delta z_\Delta(k)$$

$$\tilde{w}_\Delta(k) = \Delta \tilde{z}_\Delta(k)$$

The objective of the upper-tier MPC controller is to modify the control input given by the LPV controller in order to take into account the predictions of the sample time and model parameters in order to satisfy the state and input constraints. To this end, the input $u(k)$ is perturbed with the signal $v(k)$ decided by the MPC controller. The input applied to the system is denoted $\tilde{u}(k)$.

$$\tilde{u}(k) = u(k) + v(k)$$

The optimization problem which defines the two-tier MPC is the following:

$$\min_{v(k),v(k+1),\ldots,v(k+N-1)} \sum_{0}^{N-1} (z(k+j) - w(k+j))^2 + \lambda \tilde{u}(k+j)^2$$

subject to

$$\begin{bmatrix} x_p(k+1) \\ z_\Delta(k) \\ z(k) \\ y(k) \end{bmatrix} = \begin{bmatrix} A^p & B^p_\Delta & B^p_w & B^p_u \\ C^p_\Delta & D^p_{\Delta\Delta} & D^p_{\Delta w} & D^p_{\Delta u} \\ C^p_w & D^p_{w\Delta} & D^p_{ww} & D^p_{wu} \\ C^p_y & D^p_{y\Delta} & D^p_{yw} & D^p_{yu} \end{bmatrix} \begin{bmatrix} x_p(k) \\ w_\Delta(k) \\ w(k) \\ \tilde{u}(k) \end{bmatrix}$$

$$\begin{bmatrix} x_c(k+1) \\ \tilde{z}_\Delta(k) \\ u(k) \end{bmatrix} = \begin{bmatrix} A^c & B^c_\Delta & B^c_y \\ C^c_\Delta & D^c_{\Delta\Delta} & D^c_{\Delta y} \\ C^c_u & D^c_{u\Delta} & D^c_{uy} \end{bmatrix} \begin{bmatrix} x_c(k) \\ \tilde{w}_\Delta(k) \\ y(k) \end{bmatrix}$$

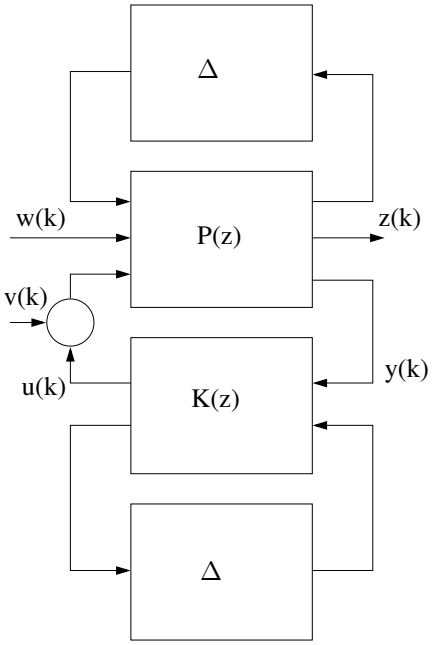

Figure 16: Two-tier MPC over an LFR varying sampling time controller

$$w_\Delta(k) = \Delta z_\Delta(k)$$
$$\tilde{w}_\Delta(k) = \Delta \tilde{z}_\Delta(k)$$
$$\tilde{u}(k) = u(k) + v(k) \in U$$

The MPC controller needs the following information to define the optimization problem:

- $x_p(k)$. Initial state of the system.
- $x_c(k)$. Initial state of the LPV controller.
- $w(k+j)$ with $j = 0, \ldots, N-1$. A prediction N steps ahead of the value of the signal w.
- Δ. Matrix corresponding to the model and sampling parameters. We assume that this matrix will remain constant along the whole prediction horizon. It is also possible to use a time-varying matrix using a prediction, that is, using $\Delta(k+j)$ with $j = 0, \ldots, N-1$.

Given the inputs at time k, the MPC controller provides $v(k)$. The optimization procedure is repeated at each sampling time.

The cost function is:

$$\sum_{0}^{N-1} (z(k+j) - w(k+j))^2 + \lambda \tilde{u}(k+j)^2$$

This implies that the MPC controller will try that the predicted output of the system $z(k)$ follows the input reference signal $w(k)$ while penalizing the value of the input $\tilde{u}(k+j)$.

This cost function is not well defined for reference tracking and will yield tracking errors in steady state. This is another reason to try to obtain an error based model in which the state is the tracking error and in which it holds that $xc = 0$ and $u = 0$ is an equilibrium point of the system.

It is important top remark that state and input constrains can be included in the optimization problem. It is also possible to take into account a contractive constraint based on the Lyapunov function of the LFR controller. This constraint would guarantee closed-loop practical stability even in the presence of asynchronous measurements. See (Muñoz de la Peña Sequedo and Raimondo (2010)) for more details.

The resulting optimization problem is a quadratic programming problem. This class of optimization problems can be solved efficiently with off-the-self optimization tools.

7.3 MPC based Feedback Scheduling

In order to follow a convex optimization approach for the feedback scheduling of the LPV controller we need to obtain a function which estimates the future performance of a given sample time (parameter Δ).

To this end we will use the tracking error model of the closed-loop system. We assume that the input $u(k)$ to the system is also available as an output of the closed-loop system:

$$\begin{bmatrix} x_p(k+1) \\ x_c(k+1) \end{bmatrix} = A(\Delta) \begin{bmatrix} x_p(k) \\ x_c(k) \end{bmatrix}$$

$$\begin{bmatrix} y(k) \\ u(k) \end{bmatrix} = C(\Delta) \begin{bmatrix} x_p(k) \\ x_c(k) \end{bmatrix}$$

The performance is given by an upper bound on the future cost to go assuming that Δ is constant.

$$J(\Delta) = \sum_{j=0}^{\infty} y(k+j)^2 + \gamma u(k+j)^2 = \sum_{j=0}^{\infty} \begin{bmatrix} x_p(k+j) \\ x_c(k+j) \end{bmatrix}^T C(\Delta)^T Q C(\Delta) \begin{bmatrix} x_p(k+j) \\ x_c(k+j) \end{bmatrix}$$

Next we define:

$$X(k) = \begin{bmatrix} x_p(k+1) \\ x_c(k+1) \end{bmatrix}, Q(\Delta) = C(\Delta)^T Q C(\Delta)$$

In order to obtain a bound on $J(\Delta)$, we need to find a matrix $P(\Delta)$ such that:

$$X^T P(\Delta) X - X^T A^T(\Delta) P(\Delta) A(\Delta) X \geqslant X^T Q(\Delta) X$$

This inequality implies that:

$$X^T(k) P(\Delta) X(k) - X^T(k+1) P(\Delta) X(k) \geqslant X^T(k) Q(\Delta) X(k)$$

for all k, and hence:

$$\begin{array}{l} X^T(0) P(\Delta) X(0) - X^T(1) P(\Delta) X(1) \geqslant X^T(0) Q(\Delta) X(0) \\ X^T(1) P(\Delta) X(1) - X^T(2) P(\Delta) X(2) \geqslant X^T(1) Q(\Delta) X(1) \\ X^T(2) P(\Delta) X(2) - X^T(3) P(\Delta) X(3) \geqslant X^T(2) Q(\Delta) X(2) \\ \ldots X^T(\infty - 1) P(\Delta) X(\infty - 1) - X^T(\infty) P(\Delta) X(\infty) \geqslant X^T(\infty - 1) Q(\Delta) X(\infty - 1) \end{array}$$

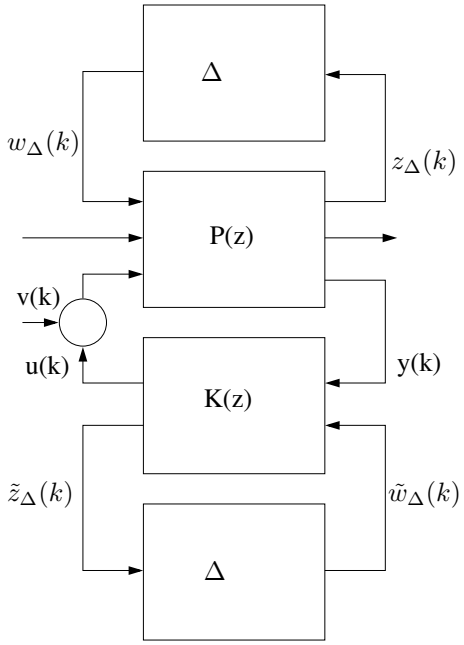

Figure 17: Feedback scheduler model

Summing all these inequalities and taking into account that the closed-loop system is stable and hence that $X(\infty)$ goes to 0, it holds that,

$$X^T(0)P(\Delta)X(0) \geqslant \sum_{k=0}^{\infty} X^T(k)Q(\Delta)X(k)$$

and hence is an upper bound on the performance index for a given Δ. Matrix P can be evaluated from the tracking error system matrices.

The idea is to approximate each of the components of $\delta P(\Delta) = P(\Delta) - P(\Delta_0)$ where Δ_0 corresponds to the minimum sampling time that can be used with a quadratic function of the sampling:

$$\delta P_{i,j}(\Delta) = \sigma_{i,j}h^2 + \mu_{i,j}h + \chi_{i,j}$$

where h is the deviation from the minimum sampling time h_0.

The constants $\sigma_{i,j}, \mu_{i,j}, \chi_{i,j}$ would be evaluated off-line from simulations using least squared.

Lack of time and man-power did not allow to implement this later solution before the end of the FeedNetBack project, however this work is going on to assess the real-time feasibility of the approach.

References

Addison, C. A., Enright, W. H., Gaffney, P. W., Gladwell, I. and Hanson, P. M.: 1991, Algorithm 687: a decision tree for the numerical solution of initial value ordinary differential equations, *ACM Transactions on Mathematical Software* **17**(1).

Apkarian, P. and Gahinet, P.: 1995, A convex characterisation of gain-scheduled $\mathcal{H}_\infty$ controllers, *IEEE Transaction on Automatic Control*, Vol. 40, pp. 853 – 864.

Arnold, M., Burgermeister, B., Führer, C., Hippmann, G. and Rill, G.: 2011, Numerical methods in vehicle system dynamics : State of the art and current developments, *Vehicle System Dynamics* .

Ascher, U. and Petzold, L.: 1998, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, ISBN 0-89871-412-5.

Aubrun, C., Berbra, C., Gentil, S., Lesecq, S. and Sauter, D.: 2010, Fault detection and isolation, fault tolerant control, *Co-design approaches for dependable networked control systems*, ISTE-Wiley.

Berbra, C., Gentil, S., Lesecq, S. and Simon, D.: 2010, Control and diagnosis for an unmanned aerial vehicle, *Co-design approaches for dependable networked control systems*, ISTE Wiley.

Berbra, C., Simon, D., Gentil, S. and Lesecq, S.: 2009, Hardware in the loop networked diagnosis of a quadrotor drone, *7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes SafeProcess'09*.

Biannic, J.-M., Marcos, A., Jeanneau, M. and Roos, C.: 2006, Nonlinear simplified lft modelling of an aircraft on ground, *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pp. 2213–2218.

Bierman, G. J.: 1976, Measurement updating using the U-D factorization, *Automatica* **12**(4), 375–382.

Borrelly, J.-J., Coste-Manière, E., Espiau, B., Kapellos, K., Pissard-Gibollet, R., Simon, D. and Turro, N.: 1998, The ORCCAD architecture, *International Journal of Robotics Research* **17**(4), 338–359.

Cao, X., Chen, J., Gao, C. and Sun, Y.: 2009, An optimal control method for applications using wireless sensor/actuators networks, *Computer and Electrical Engineering* **35**(5).

Cervin, A.: 2003, *Integrated Control and Real-Time Scheduling*, PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden.

Epstein, M., Shi, L., Tiwari, A. and Murray, R.: 2008, Probabilistic performances of state estimation across a lossy network, *Automatica* **44**, 3046–3053.

Felicioni, F., Berbra, C., Gentil, S. and Lesecq, S.: 2010, On-line adaptive control of a quadrotor under (m,k)-firm constraint, *XXII Congreso Argentino de Control Automático - AADECA*, Buenos Aires, Argentina.

Felicioni, F. E.: 2011, *Estabilidad y Performance de Sistemas Distribuidos de Control*, PhD thesis, Facultad de Ciencas Exactas, Ingenería y Agrimensura, Universidad Nacional de Rosario, Argentina.

Felicioni, F., Jia, N., Simonot-Lion, F. and Song, Y.-Q.: 2010, Overload management through selective data dropping, *Co-design approaches for dependable networked control systems*, ISTE-Wiley.

Felicioni, F., Jia, N., Song, Y.-Q. and Simonot Lion, F.: 2008, Optimal on-line (m,k)-firm constraint assignment for real-time control tasks based on plant state information, *12th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, Hamburg, Germany.

Felicioni, F. and Junco, S.: 1998, A Lie algebraic approach to design of stable feedback control systems with varying sampling rate, *17th IFAC World Congress*, Seul, Korea.

Ford, J. J.: 2002, Non-linear and robust filtering : From the kalman filter to the particle filter, *Technical Report DSTO-TR-1301*, Weapons Systems Division, Aeronautical and Maritime Research Laboratory.

Gauthier, C., Sename, O., Dugard, L. and Meissonnier, G.: 2007, An $\mathcal{H}_\infty$ linear parameter-varying (LPV) controller for a diesel engine common rail injection system, *Proceedings of the European Control Conference*, Budapest, Hungary.

Gleason, D. M.: 1996, Avoiding numerical stability problems of long duration DGPS/INS kalman filters, *Journal of Geodesy* **70**(5), 263–275.

Guerrero-Castellanos, J.-F.: 2008, *Estimation de l'attitude et commande bornée en attitude d'un corps rigide: Application à un mini helicopter à quatre rotors*, PhD thesis, Université Joseph Fourier-Grenoble I, France. (in French).

Guerrero-Castellanos, J.-F., Berbra, C., Gentil, S. and Lesecq, S.: 2009, Quadrotor attitude control through a network with (m-k)-firm policy, *European Control Conference ECC09*, Budapest, Hungary.

Hamdaoui, M. and Ramanathan, P.: 1994, A service policy for real-time customers with (m,k)-firm deadlines, *Fault-Tolerant Computing Symposium*, Austin, USA, pp. 196–205.

Hespanha, J., Naghshtabrizi, P. and Xu, Y.: 2007, A survey of recent results in networked control systems, *Proceedings of the IEEE* **95**(1), 138–162.

Huang, M. and Dey, S.: 2007, Stability of kalman filtering with markovian packet losses, *Automatica* **43**(4), 598–607.

Jia, N., Song, Y.-Q. and Simonot-Lion, F.: 2007, Graceful degradation of the quality of control through data drop policy, *European Control Conference, ECC'07*, Kos, Greece.

Jimenez, J., Lopez, M., Smyth, P., Gennari, G., Raimondo, D., Donaggio, P., Gasparella, S., Lain, B., Opderbecke, J., Schenato, L., Simon, D., Oechtering, T. and Fischione, C.: 2010, FeedNetBack-D01.04 - Tool Specifications, *Deliverable*, FeedNetBack IST project 223866.

Juanole, G., Mouney, G. and Calmettes, C.: 2008, On different priority schemes for the message scheduling in networked control systems, *16th Mediterranean Conference on Control and Automation*, Ajaccio, France.

Lesecq, S., Gentil, S. and Daraoui, N.: 2009, Quadrotor attitude estimation with data losses, *European Control Conference ECC09*, Budapest, Hungary.

Li, J., Song, Y.-Q. and Simonot Lion, F.: 2006, Providing Real-time Applications with Graceful Degradation of QoS and Fault Tolerance According to (m,k)-firm Model, *IEEE Transactions on Industrial Informatics* **2**, 112–119.

Lohning, M.: 2010, Lpv and lfr modelling of elastic robots for controller synthesis, *Control and Automation (ICCA), 2010 8th IEEE International Conference on*, pp. 522–527.

Muñoz de la Peña Sequedo, D. and Raimondo, D. M.: 2010, FeedNetBack-D04.02-design of model predictive controllers over networks, *Technical report*, FeedNetBack IST project.

Ohlin, M., Henriksson, D. and Cervin, A.: 2007, *TrueTime 1.5—Reference Manual*.

Opnet Technologies Inc: 2011, OPNET, http://www.opnet.com/.

Pfifer, H. and Hecker, S.: 2008, Generation of optimal linear parametric models for lft-based robust stability analysis and control design, *47th IEEE Conference on Decision and Control, CDC'08*, pp. 3866 –3871.

Robert, D., Sename, O. and Simon, D.: 2010, An H_∞ LPV design for sampling varying controllers : experimentation with a t inverted pendulum, *IEEE Transactions on Control Systems Technology* **18**(3), 741–749.

Roche, E.: 2011, *Commande à échantillonnage variable pour les systèmes LPV : application à un sous-marin autonome*, PhD thesis, Gipsa-lab, EEATS doctoral school, Grenoble University, France.

Roche, E., Sename, O., Seuret, A., Simon, D. and Varrier, S.: 2010, Feednetback-d04.03 - design of robust variable rate controllers, *Contrat*, FeedNetBack IST project.

Roche, E., Sename, O. and Simon, D.: 2010, LPV / $\mathcal{H}_\infty$ varying sampling control for autonomous underwater vehicles, *Proceedings of the IFAC SSSC*, Ancona, Italie.

Roche, E., Sename, O., Simon, D. and Varrier, S.: 2011, A hierarchical varying sampling $\mathcal{H}_\infty$ control of an auv, *18th IFAC World Congress (IFAC WC 2011)*, Milano, Italy.

Schenato, L.: 2009, To zero or to hold control inputs with lossy links?, *IEEE Transactions on Automatic Control* **54**(5), 1093–1099.

Schenato, L., Sinopoli, B., Franceschetti, M., Poolla, K. and Sastri, S.: 2007, Fundations of control and estimation over lossy networks, *Proceeedings of the IEEE* **95**(1), 163–187.

Schinkel, M., Chen, W.-H. and Rantzer, A.: 2002, Optimal control for systems with varying sampling rate, *American Control Conference ACC'02*, Anchorage, USA.

Simon, D.: 2011, Hardware-in-the-loop test-bed of an Unmanned Aerial Vehicle using Orccad, *6th National Conference on Control Architectures of Robots*, INRIA Grenoble Rhône-Alpes, Grenoble, France.

Simon, D. and Benattar, F.: 2005, Design of real-time periodic control systems through synchronisation and fixed priorities, *International Journal of Systems Science*, Vol. 36, pp. 57–76.

Simon, D., Robert, D. and Sename, O.: 2005, Robust control/scheduling co-design: application to robot control, *RTAS'05 IEEE Real-Time and Embedded Technology and Applications Symposium*, San Francisco.

Simon, D., Seuret, A., Hokayem, P., Lygeros, J. and Camacho, E.: 2009, FeedNetBack-D04.01-State of the art in control/computing co-design, *Technical report*, FeedNetBack IST project.

Simpson, H.-R.: 1997, Multireader and multiwriter asynchronous communication mechanisms, *IEE Proceedings-Computer and Digital Techniques* **144**(4), 241–244.

Sinopoli, B., Schenato, L., Franceschetti, M., Poolla, K., Jordan, M. and Sastry, S.: 2004, Kalman filtering with intermittent observations, *IEEE Transactions on Automatic Control* **49**(9), 1453–1464.

Sun, S., Xie, L., Xiao, W. and Soh, Y.: 2008, Optimal linear estimation for systems with multiple packet dropouts, *Automatica* **44**, 1333–1342.

Törngren, M., Henriksson, D., Redell, O., Kirsch, C., El-Khoury, J., Simon, D., Sorel, Y., Zdenek, H. and Årzén, K.-E.: 2006, Co-design of control systems and their real-time implementation - a tool survey, *Technical Report TRITA - MMK 2006:11*, Royal Institute of Technology, KTH, Stockolm, Sweden.

Verhaegen, M. and Verdult, V.: 2007, *Filtering and System Identification*, Cambridge University Press.