Real-time Simulation for a Functional Electrical Stimulation system validation

Daniel Simon, David Andreu, Samy Lafnoune

Real-time Simulation for a Functional Electrical Stimulation system validation

Daniel Simon, David Andreu and Samy Lafnoune
DEMAR team, INRIA Sophia Antipolis - Méditerranée
LIRMM, CNRS UMR 5506, Montpellier
http://www.lirmm.fr/demar

Abstract

Functional Electrical Stimulation (FES) is used in therapy for rehabilitation or substitution for disabled people. They are control systems using electrodes to interface a digital control system with livings. Hence the whole system gathers continuous-time (muscles and nerves) and discrete-time (controllers and links) components. During the design process, realistic simulation remains a precious tool ahead of real experiments to check without danger that the implementation matches the functional and safety requirements. The paper presents a real-time open software simulation system, dedicated to the analysis of FES systems deployed over distributed execution resources and wireless links. The simulation tool is especially devoted to the joint design and analysis of control loops and real-time features.

1 The FES framework

Functional electrical stimulation (FES) is one of existing rehabilitation techniques to restore lost motor functions for motor-impaired people. For example, in case of spinal cord injuries, the natural pathways between the central nervous system (CNS) and peripheral nerves are open under the lesion level. Therefore such lesions forbid both the activation of movements through motor nerves and the collection of sensory information by the CNS.

In FES systems, electrodes are connected to nerves or muscles and generate electrical pulses to induce controlled contractions of muscles. Similar electrodes can be used to collect activity signals from muscles (ElectroMyoGrams EMG) or from nerves (ElectroNeuroGrams ENG). FES is primarily used for rehabilitation of functions for people with disabilities. It means people with spinal cord injuries, but it is also used for others neurological disorders, such as those following head injuries, stroke and Parkinson's disease. Until now FES is mainly used in open-loop mode, where the therapist selects predefined currents patterns (e.g., frequency, intensity and envelope of the current) according to the desired effect on the patient. However open-loop stimulation only allows for quite simple movements generation.

It is argued, e.g. [17], that closed-loop FES is necessary to safely control and coordinate the numerous electrodes and muscles needed to generate complex movements such as walking.

In that case (Figure 1), the stimulation controller is fed back by various sensors, such as limbs joints angles, IMUs providing accelerations, electrophysiological signals such as EMG... These signals are then used by feedback controllers to accurately control the artificially actuated limbs.

Indeed it is well known that feedback control provides adaptability w.r.t. varying operation conditions and robustness w.r.t. the uncertainties spoiling the control loops components.

However, the design of effective and safe distributed control loops, especially using wireless links, still needs basic research to put together the control and computing sciences. The system safety, including human beings in the loops, firstly relies on *robustness*. Robustness deals with the ability of a control system to behave accordingly to its specification despite modeling errors and disturbances. It is an effective way to cope with the unavoidable uncertainties of real systems, coming both from the imperfect knowledge of the physical controlled system and from the numerical implementation complexity of the controllers.

Compared with mechanical devices, the large uncertainties and variability of livings make them especially difficult to be accurately modeled and safely controlled. Conversely, considering humans in the loop, the safety of the technology must be validated, e.g. formally assessed, to allow for its certification and usage in everyday life on a large scale.

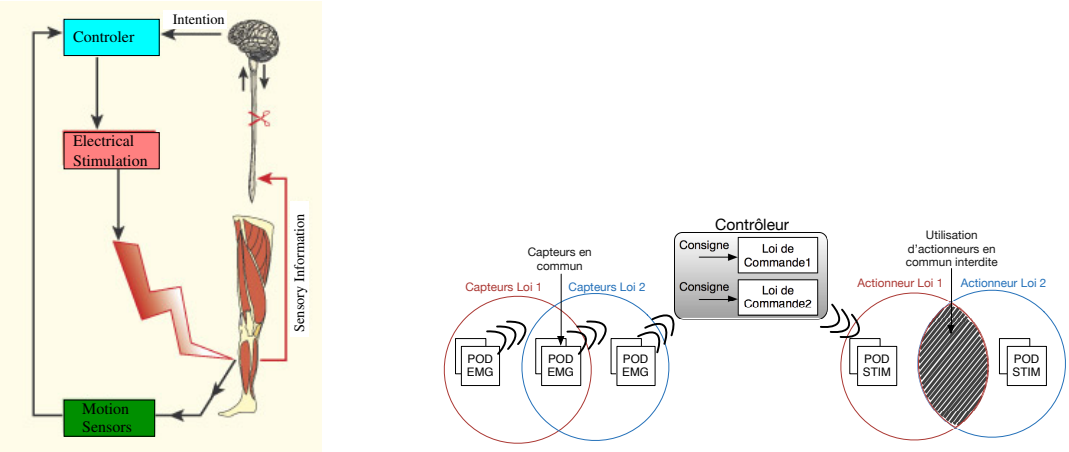


Figure 1: Feedback FES

1.1 The Phenix Liberty FES system

The Phenix Liberty FES system has been developed along a partnership between the Demar research team and the Vivaltis company [1]. It is devoted to external stimulation, i.e. the electrodes are glue on the skin, facing the target muscle or nerve. Compared with its competitors, a distinctive feature of the system is its wireless implementation. Considering the multiple sensors and actuators needed to control complex motion, distributed systems over wireless links distinctively comply with mobility constraints, leading to acceptance from the human users. The marketed system is used everyday for electro-therapy, using open-loop stimulation patterns. The finite state executes a preset stimulation sequence in an open-loop fashion when a specific condition is met. Although it is fluently used to correct, e.g., foot drop, finite-state controllers typically are not adequate to adapt stimulation patterns for frequent changes of walking speed. FES also fastly involves muscular fatique, thus needed an on-line detection and adaptation of the stimulation profiles. Current research aims to design and implement closed-loop FES using this system.

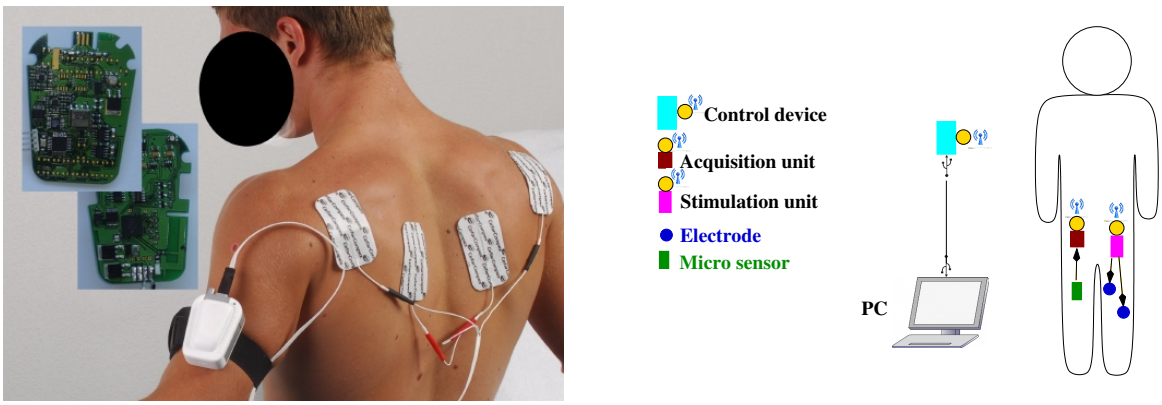


Figure 2: The Phenix FES system

The system is deployed over a wireless architecture where the main components are :

- A main PC is used as a end-users (i.e. a therapist) interface, to select the stimulation pattern out of a library of pre-defined profiles, and to provide the gateway with the controller using an USB link;
- The *control device* manages the communications with the distributed stimulation units through the wireless network running the protocol stack. For future feedback controlled stimulation it will also run the real-time control loops. The controller board is equipped with a Freescale MC1322X micro-controller and an integrated IEEE802.15.4 2.4 GHz Radio Frequency (RF) emitting/receiving chip.

[1] http://www.vivaltis.com/

There are several kind of distributed units (DSUs, also named PODs) :

- The "Stim/Bio" DSUs allows for both the stimulation of muscles for movement generation and for the acquisition of EMG signals for biofeedback purpose. They are made of two electronic boards :
  - A generic board embeds the housekeeping hardware and software to provide the electrical supply (battery), switches and LEDs, and a physical network transceiver. The protocol stack and the requests (stimulation and acquisition) processing are handled by a Freescale MC1322X micro-controller.
  - A daughter board executes the stimulation patterns or the acquisition requests through two input/output analogue channels.
- The "universal" DSUs use the same generic board, but embeds in its daughter board specific sensors. For example, IMUs or goniometers are planned to be embedded in a near future.

The communication architecture used a specifically designed 3-layer protocol stack, in compliance with the reduced OSI model. These layers are the Application layer, the MAC layer and the Physical layer. The physical layer ensures the telecommunication over a wireless medium (used in considered hardware design). The MAC layer of this protocol stack ensures a deterministic medium sharing, meaning that only one unit can speak over the network at a time (no collision), so that the response time to requests (including acknowledgment) is known and bounded. It allows the controller to work with either a single (unicast) or a group of target DSUs (multicast) by using an individual or group identifier [11].

The various control, communication and supervision tasks run on the micro-controllers under control of a CMX real-time multi-tasking executive.

1.2 From control design to real-time experiments

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 control algorithms. 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.

Sampling jitters and the network-induced delays have a strong impact on control performances. Thus, the schedulability of a control system should be assessed not only with respect to deadlines but also by considering other criteria such as time-varying delay and jitters. Furthermore when actuators, sensors and controllers are distributed over a network, the use of network and processor resources is a key point to handle.

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 over-sizing 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 control algorithm, its implementation on the available execution resources and the controlled process 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. [6], to study the influence of the 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.

In this design process, simulation is an indisputable step between concept design and prototype validation. Realistic simulations allow for the preliminary evaluation, tuning and possibly redesign of proposed solutions ahead of implementation, thus lowering the risks. To be confident in the result, building such simulations needs high fidelity models both for the components and for their interaction.

2 Simulation setup

2.1 Numerical simulation of control process

The design and validation of complex systems, like cyber-physical systems which include both physical and computational devices, is costly, time consuming and needs knowledge and cooperation of different disciplines [15]. For example, medical robotics belong to such category and require the coordinated design of both bio-mechanical, electrical, and chemical models (from a physical point of view) and many-sided controllers (from a computational point of view). A global simulation is needed at an early stage to speed-up the design, development and validation phases, especially when real experiments are costly or dangerous for human beings.

The main purpose of the numerical simulation is, when an analytic solution cannot be derived, to approximate as faithfully as possible the behavior of the complex dynamic system. In other words, bounding and minimizing the simulation errors is an important goal of numerical simulations so that the designers can be confident with the prediction.

Both modeling and numerical integration deal with approximations, hence it is first needed to find a satisfactory trade-off between the simulation speed and precision. Ultimately, the simulation of the physical models will take into account some real-time constraints introduced by the interaction with the real components. Indeed, these models are intended to validate controllers, e.g., to combine efficiency stimulation with safety concerns. Using real-time simulation means that the interaction between the simulated world and the real world must be consistent, i.e. that the simulated time and real-time must be carefully meshed to meet at some precise points [9].

Real-time simulation implies matching two concepts of time:

- Real-time : it is the physical time or the time reference of the real physical system that is modeled and simulated, e.g. as measured by an absolute wall clock;
- Simulated time : it is the time elapsed during the execution of the simulation that can be measured by the numerical integrator clock (i.e., the accumulation of integration steps).

Indeed, in a real-time simulation the simulation time needs to be meshed to the real-time. Several simulation steps of growing accuracy and fidelity to the real implementation can be processed, from basic functional investigation in continuous time until hardware-in-the-loop setups including parts of the final hardware and software, just prior to real tests [3].

Model-in-the-loop 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 Scilab/Scicos or Matlab/Simulink. 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 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. Note that there exist Rapid Control Prototyping tools able to directly generate run-time code and I/O interface from the models tools.

Software-in-the-loop As for MIL, the SIL phase considers only simulated (i.e. software) elements, but it takes into account the actual production code, including implementation related constraints such as fixed-point computations and memory size limits.

Additionally, in a *real-time* SIL simulation, the data flows between the plant and the controller models must respect the computation and communication rates of the real components. Therefore the execution of different components (computation tasks and clocks) of the system must be carefully synchronized to avoid mixing data flows with inconsistent time scale and labels [13]. Such simulation may help to check that the specified timing parameters of the implemented software are consistent with the controller and plant dynamics, and help to dimension the hardware architecture.

Hardware-in-the-loop After validating the real software in a real-time SIL simulation, the real hardware can be checked in a real-time Hardware-In-the-Loop (HIL) simulation. A HIL simulation consists of a combination of simulated and real components, which means that a real component can be replaced by an artificial one. Typically, in a HIL simulation the plant is still simulated using a numerical integration of the plant model, under control of the implementation code running (at least partly) on the real target execution system (CPUs, OS, I/O devices, fieldbus,...). The simulated plant moves consistently with the real system dynamics, and the computation time seen by the remaining simulated parts of the control architecture, if any, must be equal to the continuous real-time time passing by the real world.

HIL is often used in industry to test embedded software on its final execution platform due to the unavailability of some parts of the real system. It offers many advantages : reducing costs by requiring only some equipment, rapid prototyping, good representation of the system, flexibility by allowing repetitive tests and trying destructive tests without impacting on the hardware [7].

2.2 Simulation architecture

Objectives Considering that a long term objective is the safe and efficient implementation of motion control using FES feedback loops, and that the analysis of such complex hybrid system is out of scope of pure theoretic analysis, we need a software tool to help the system designer to design, dimension, analyze and tune the components of the system and their interaction, w.r.t. specified performance and safety levels.

More precisely, we need that the system must use an open source framework (as far as possible) and run on stock PCs. It must easily evolve from functional simulation to HIL to cover the different phases of software development, in other words it must be open and easily customizable to integrate new features, or refined components versions, when needed along the design process. Ideally real components and their artificial software model should be easily (plug-and-play) exchangeable with no need to reconfigure the other modules.

It is needed to avoid useless models complexity leading to excessive computation times and difficult analysis. According to the designer current focus, some simulated components might use quite simple models while others should be more detailed. For example, a preliminary analysis of the control loops sampling rate do not need a very detailed physical model of the communication medium (which could be added later on for refined tuning).

Processes and threads Based on previous experience the simulation software is designed as a set of Unix processes and Posix threads running under Linux on stock multi-core PCs. The first version of the simulator is a pure software which does not integrate any piece of real hardware, and calculations are performed with the usual floating point capabilities of the PC.

However, to mimic the real distributed architecture, it is split in 3 processes so that it can be run on a multi-core PC (see Figure 3), for example one core simulates the controller and each DSU is simulated on another core :

- the "Controller" process executes the FES controllers running on the control chip under supervision of a multitasks O.S.; it also simulates the network transceiver through a simplified model of the communication stack;
- the "Medium" process is used to simulate the physical features of the wireless link;
- the "DSU" process handles models of the Stim and Acquisition DSU fonctions in dedicated threads. It also handles the computation thread running the numerical integrator.

Using the (non portable) *pthread_attr_setaffinity_np* CPU allocation statement, each of these process can be run on a different core. In the future these processes might be easily allocated on different chips connected by the real transceivers to enhance the simulation framework towards an HIL simulation.

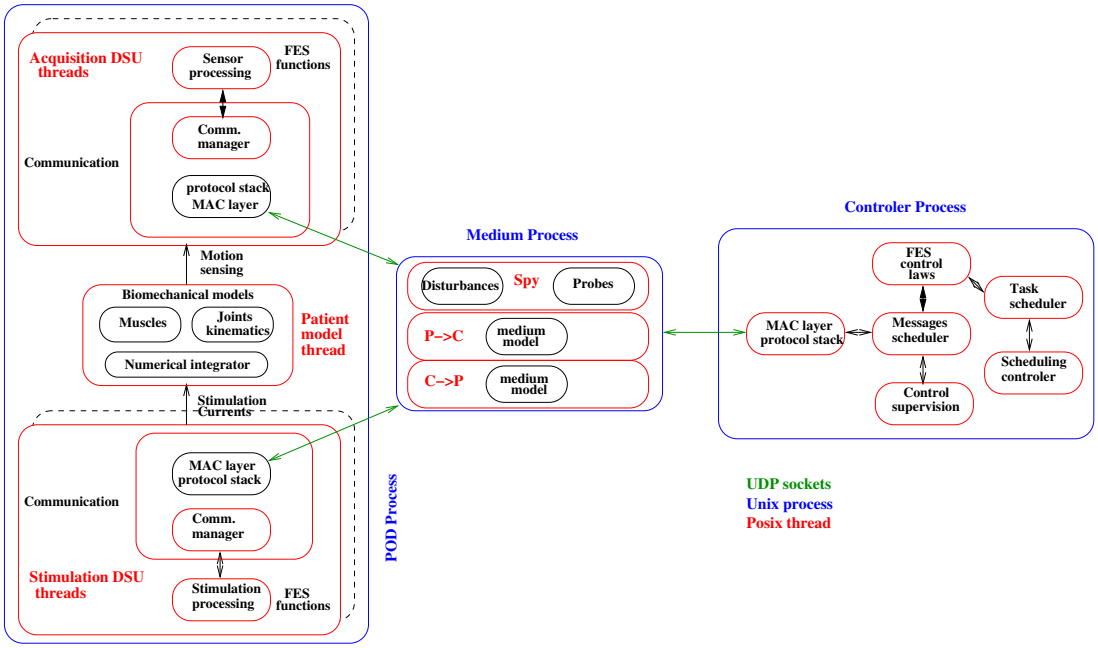


Figure 3: Simulation architecture

Communication and synchronization A simplified version of the communication protocol stack is implemented in dedicated communication threads. The data exchanged between the controller and the DSUs are encapsulated in packets whose structure is similar to those handled by the real network. In the simulation, the packets are handled in two steps by UDP Unix sockets :

- the packet is first transmitted using a point-to-point UDP socket from the comm. manager of the controller (or of the Pod) to the medium process receiving thread;
- in the current version, the model of the medium just add a constant delay to the packet transmission, and data loss are handled by a simple random process.
- after being delayed, the packet is sent to all receivers using a diffusion UDP socket (using the "so_broadcast" parameter of UDP sockets). This feature allows for sending data with a unique time stamp to groups of Pods.

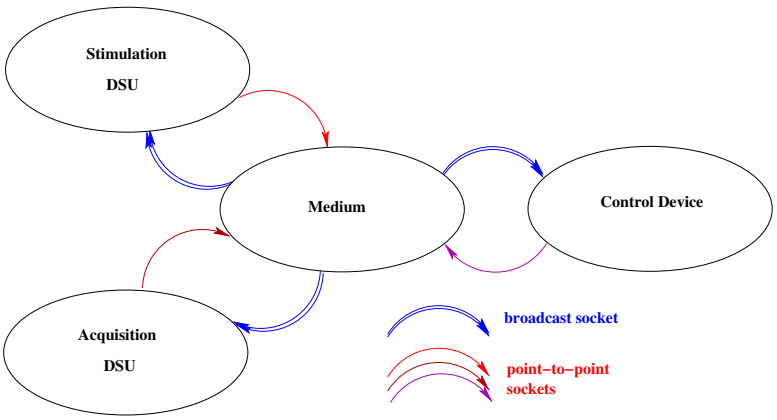


Figure 4: Communication sockets

Timing values All threads in the simulation can be unlocked either by signals from clocks (e.g. for periodic control tasks), by blocking waiting messages on an incoming socket or by semaphores signaled by communication requests sent by another thread. The threads are scheduled by the Linux kernel, set with the RT_FIFO real-time mode, according to their priorities and locks. However stock Linux kernels sometimes provides inconsistent behaviors, although the simulation works coherently with kernel compiled with real-time options, i.e. with high resolution timers and low latency preemption model under superuser privilege.

The transmission delays on the wireless links are simulated by pure delays in the medium component. When the simulation is run on a modern laptop, all the calculation times are by far smaller than the real ones measured on the production micro-controller. To recover realistic activity charts and keep the right interleaving of events in the simulation, active loops with the right duration can be added to the simulated threads. They burn cycles, keeping the thread active with the right priority for a time equal to the value estimated for the real micro-controller. These timing values are set according to metrology sessions made with the production system [8]. Probes using functions of the Posix real-time library can be inserted anywhere in the threads to measure, e.g., communication delays between two tags or to record missed deadlines.

Solver triggering The numerical integration thread is a special case. Indeed this thread is not the ghost of a real time task existing in the real implementation.

In real life the state of the physical process naturally evolves in parallel with the numerical execution of the control and communication systems. The time scales of the continuous and numerical entities are meshed at some meeting instants at the occurence of different kind of events :

- when a control system requests an updated sensor value;
- when a control system sends a new control value;
- when the simulation system needs a snapshot of the controlled process and of its environment;
- when the continuous process raises an event (e.g., via an asynchronous sensor or crossing a model discontinuity).

The integration thread is triggered by any of these events. At each triggering event the integrator is first run from the last interruption until the current time to update the state of the continuous process, then it delivers the new state or accept new control inputs.

Therefore even if the integration is always performed late w.r.t. real-time, the numerical integration process is driven by the events coming from the real-time control system, or by the other real-time threads of the fake system, so that the simulated continuous time is meshed with real time at the synchronization instants.

The system works pretty well provided that the computation time needed to update the continuous state from the last event is kept small, in particular the integration step must be finished before the occurrence of the next synchronization 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 complexity and fastest time constant, on the integration algorithm and on the host computer speed.

In practice, for the current set-up the simulation runs very comfortably faster than real-time on a modern laptop. Computation times and overruns can be easily checked by software probes and reported to provide information and warnings about the timing behavior and quality of the simulation.

Note that, if the numerical integration cost becomes prohibitive to run the simulation on a single CPU w.r.t. real-time, the model can be split over several CPUs. Slackened synchronization between decoupled sub-models allows for significant speed-ups while keeping the integration errors under control [3].

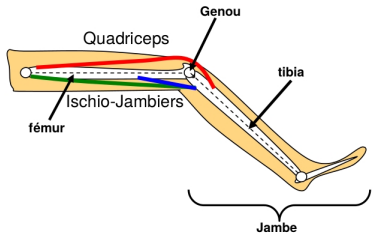
2.3 Continuous plant and numerical integration

Continuous model The model of the continuous plant is usually given as a set of ordinary differential equations (ODEs), differential algebraic equations (DAEs) or partial derivative equations (PDEs), where the continuous (physical) time is an independent variable.

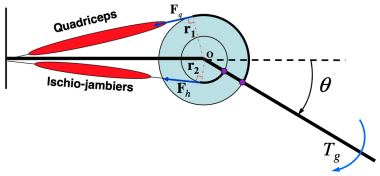
Building high fidelity system-level models of complex systems, like mechatronic systems and even more systems involving livings, is a challenging duty. One problem is the diversity of modeling and simulation environments used by the various involved multi-disciplinary teams. Particular environments are preferred for a specific use due to distinctive strengths (modeling language, libraries, solvers, cost, etc.), for example the automatic derivation of the kinematic and dynamic equations of a multi-body system can be a distinctive feature.

Perfect modeling must be forgot. A model is always an approximation of reality, both in its structure and in its parameters values. A problem for simulation, due to both model complexity and fast system dynamics, is the prohibitive CPU times which can be observed when high-fidelity models are run, therefore preventing any real-time simulation. It is needed to find a satisfactory trade-off between the simulation speed and precision. A good model captures the essential behavior of the physical plant. Therefore, according to its user and objectives (e.g. an control scientist dealing with gains tuning, or a therapist willing to choose the stimulation frequency), different models of a given physical plant can be chosen.

Building a high-fidelity model needs precise knowledge about the plant itself, here coming from bio-mechanics and neuroscience. The simulation setup has been tested using the model of a human knee controlled by electrostimulation, a topic for which the Demar team developed a strong experience over years. The main structure and parameters of the model were mainly taken from [5] (page 47) for artificially excited muscles and from [16] for the joint non-linear behavior.



(a) Agonist/antagonist muscles pair for knee flexion/extension



(b) Simplified kinematics

Figure 5: 2D bio-mechanical model of the knee

It is given as a small set of non-linear ODEs. A contractile element (CE) contraction dynamics is given by [5] :

$$\dot{K}_c = -K_c|u| + \alpha K_{cm} Fl_c(\epsilon_c)|u|_+ - K_c|\dot{\epsilon}_c| \quad (1)$$

$$\dot{F}_c = -F_c|u| + \alpha F_{cm} Fl_c(\epsilon_c)|u|_+ - F_c|\dot{\epsilon}_c| + L_{c0} K_c \dot{\epsilon_c} \quad (2)$$

where K_c and F_c are the stiffness and force developed by the CE, K_{cm} and F_{cm} their maximum values, u and $|u|_+$ chemical control variables coming from an activation model, $Fl_c(\epsilon_c)$ the force/length relation of the CE, α the recruitment rate, and $\dot{\epsilon}_c$ the contraction speed of the CE.

The numerical integration of the later variable, given by

$$\dot{\epsilon_c} = \frac{k_s L_0 \dot{\epsilon} + F_c|u| - \alpha F_{cm} Fl_c(\epsilon_c)|u|_+}{k_s L_{c0} + K_c L_{c0} - s_v F_c} \quad (3)$$

needs a special attention as it involves the *sign* function :

$$s_v = \text{sign}(k_s L_0 \dot{\epsilon} + F_c|u| - \alpha F_{cm} Fl_c(\epsilon_c)|u|_+) \quad (4)$$

Correctly handling this non-linearity needs an integrator enabled with a *root-finding* capability to provide a fast and stable integration (see next section).

The two muscles involved in the knee flexion/extension, i.e. the quadriceps and hamstring, have similar models with different parameters and are actuated by their own electrode. For our purpose, the knee joint is considered to be a perfect cylindrical d.o.f. leading the usual equation of motion (Figure 5b) :

$$\ddot{\Theta} = \frac{1}{J}(F_2 r_2 - F_1 r_1 - MgL_{og} \sin(\Theta - \Theta_0) - B\dot{\Theta} + T_e) \quad (5)$$

where F_i is the force induced by muscle i, B a non-linear damping and T_e a passive elasticity torque [16].

Finally the model is encoded for integration as a set of 8 first order non-linear ODEs, 3 for each muscle and two for the joint dynamics.

Numerical integration 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)) [1]. 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 depend 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 [2]. 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, instability 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 open-source *LSODAR* package we have elected [12] for this experiment uses a variable step, variable number of steps integration algorithm. It is suitable for both stiff and non-stiff systems, and automatically selects between non-stiff and stiff methods.

It is associated with a root-finding stopping criteria : when it detects a sign change in a particular function of the states (root detection), it runs a root-finding algorithm to find the exact time point at which the sign change occurred. The later feature is essential to correctly integrate continuous plants with discontinuities, i.e. where the values of some states or parameters, or even the model structure itself, suddenly change where a given threshold is crossed (zero-crossing). It is for example the case for models of impacts, dry friction, and here for muscles contraction.

2.4 Control design

To generate movement for a joint d.o.f., two muscles (an agonist and antagonist pair) must be generally stimulated. For the knee these two muscles are the quadriceps on the anterior face (extension actuator) and the hamstring on the posterior face (flexion actuator). In a first simple control approach, the two muscles will be actuated in mutual exclusion according to the flexion and extension phases of the movements (isometric contractions are not considered up to now), see Figure 6.

As usual, the first control algorithm to be considered is a PID. Indeed PID (Proportional, Integral, Derivative) control algorithms are widely used, as they are able to control many single input/single output (SISO) systems through a very

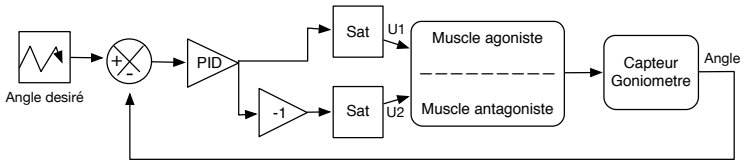


Figure 6: Control scheme

simple modeling and tuning effort. In that case the control input u is written as a function of the error signal -that is the difference between a desired output and its measure on the real system- $e(t) = r(t) - y(t)$ as:

$$u(t) = Ke(t) + \frac{K}{T_i}\int_0^t e(\tau)d\tau + KT_d\frac{d}{dt}e(t) \tag{6}$$

Here the proportional term $K \cdot e(t)$ controls the bandwidth and rising time of the control loop, the derivative term $KT_d\frac{d}{dt}e(t)$ damps the oscillations and overshoots and the integral term $\frac{K}{T_i}\int_0^t e(\tau)d\tau$ nullifies the static errors, that is the value of the error $e(t)$ when t goes to the infinite.

Indeed, the ideal continuous PID must be discretized for implementation purpose. For example, using a backward difference method (with T_s the sampling period) yields

$$u_k = u_{k-1} + K[e_k - e_{k-1}] + \frac{K \cdot T_s}{T_i}e_k + \frac{K \cdot T_d}{T_s}[e_k - 2e_{k-1} + e_{k-2}] \tag{7}$$

Note that for safety reasons the output currents sent to the electrodes must be limited to avoid damages to the muscles and joints. Due to the integral term, the control saturation must be carefully handled by the control algorithm to avoid large position overshoots. The particular numerical implementation given by (7) naturally provides the desired anti-windup action.

3 Preliminary experiments

In this preliminary version, the simulation system is a "Model-in-the-loop" simulation, where all the objects are modeled, the physical plant state is evaluated by a numerical integrator and the control functions are encoded in plain C and executed using the floating point features of a modern laptop. Anyway, it goes beyond usual simulations with Scicos or Simulink as it handles quite precisely some implementation and scheduling related features such as tasks and events interleaving, or computation and communication times.

Hence it can be already used to explore the possible performance of FES feedback loops according to various scheduling parameters.

3.1 PID tuning and sampling rate

A simple PID controller has been setup for preliminary tests of the simulation platform. Indeed a PID with fixed gains would perform uniformly for a linear system, which is not the case for the knee joint, where even a simple model shows several sources of non-linearities. According to [16] the damping and passive elasticity of the joint are non-linear. The stimulation currents are limited, as for any physical systems, but for FES saturation occurs early due to safety constraints rather than technology limits. The joint is actuated by two muscles with different characteristics for flexion and extension. Finally, the muscle model itself is also unusual, with different gains for contraction and elongation.

Hence it is not possible to find fixed gains for the PID allowing a constant response for all movements amplitudes and directions (Figure 7). Moreover, it is known that fatique occurs faster during FES than for natural muscle control (which was not tested here). Therefore fixed gains controllers cannot ensure a constant performance of FES control loops, and adaptive control, e.g. predictive control as in [4], should be used to cope with human beings variability.

Initially simulations were made with a sampling rate equal to 40 msec, this value is the one used for experiments using the marketed version of the Phenix system. Indeed, it was observed that the bottleneck is due to the USB link between the main PC (where the control laws run) and the controller which is currently used as a gateway between the PC and the DSUs. In fact, the end-to-end communication time (including control tasks execution and messages transmission and

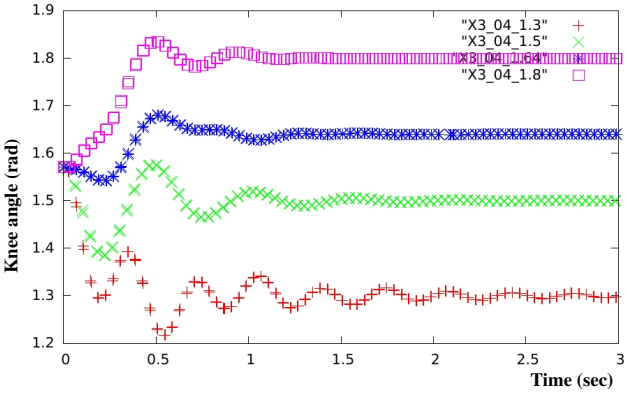


Figure 7: Knee position for various set-points

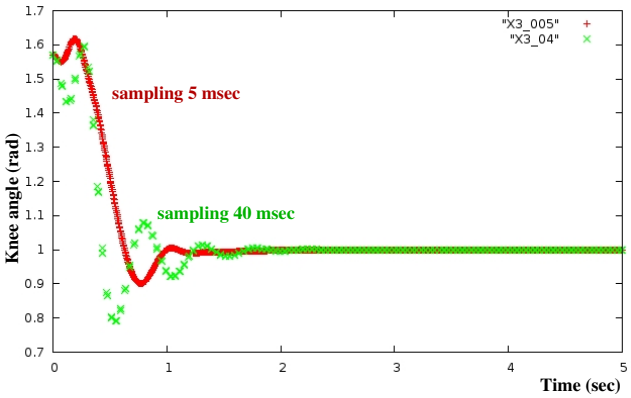


Figure 8: Knee position and sampling rates

acknowledgment) between the controller and the DSUs is lower than 5 msecs (Figure 10). Lowering the sampling rate to 5 msecs shows an improved performance of the joint controller (Figure 8). Note that sampling rates smaller than 5 msec induce many message loss and poor control performance.

Therefore the real-time simulator allows for a fine tuning of control gains according to the execution resources at hand, and to explore the benefits of architectural modifications for the control performance.

3.2 Communication protocol enhancements

The current communication threads do not implement the production code of the full protocol stacks, but only its characteristic features. The simulated transceiver encapsulates data in frames according to the STIMAP protocol [10] used to connect the controller and DSUs. These frames are further sent/received over UDP sockets. Several versions of the communication frames have been implemented, and their impact on the feedback control performance and on the communication load has been evaluated.

Single receiver (Production version) A data frame issued by the controller contains the address of a single receiving/answering DSU, and the payload only contains the stimulation parameters for this particular DSU;

Single receiver with optimization The optimization uses the mutual exclusion between the agonist/antagonist muscle and send only frames to the active muscle in the current motion phase. However some safety messages are sent (at a slower rate compared with control messages) to keep alive the DSU of the passive muscle, leading to irregular timing patterns;

Multiple receivers Messages are addressed to a group of DSUs, with a payload made of the concatenation of the individual stimulation parameters: the frame is longer than a single receiver frame but shorter than the concatenation of

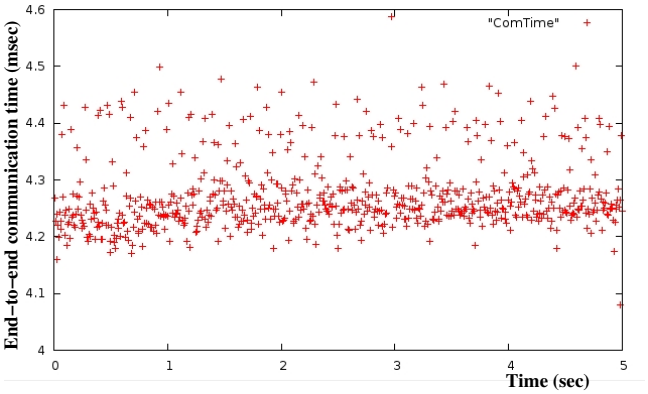


Figure 9: End-to-end communication time

the single receiver frames, thus saving communication bandwidth;

Multiple receivers with sliding windows Here the acknowledgments from the DSUs are sent as early as possible without waiting for full fixed timing slots.

The observation of the simulations traces (Figure 10) show that using optimized versions of the protocol may significantly save bandwidth and decrease the end-to-end communication delay between sensing and actuation [14], leading to improvements in the performance and robustness of the FES control law. Such results are useful to motivate (even costly) modifications of the production hardware and software.

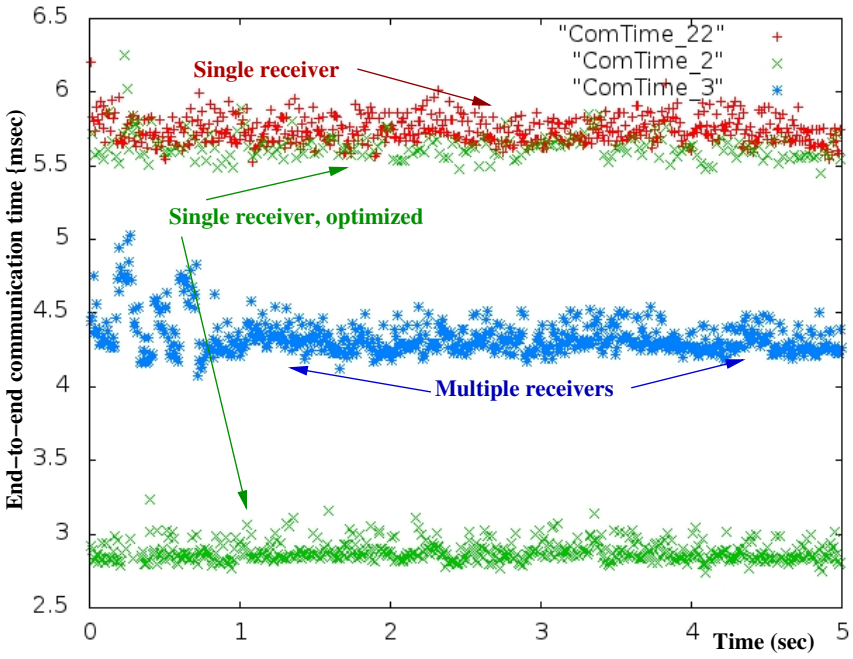


Figure 10: End-to-end communication time

4 Summary and future developments

Realistic simulations are effective tools to design and tune complex systems whose analysis cannot be provided only by theory. Several simulation steps can be explored, from simple functional analysis to HIL, to design, test, tune and validate

both the single components of the system and their interactions in a distributed architecture. Simulations are precious, as they allow for non-destructive trials, which must be considered in any domain but this is of particular interest for bio-engineering.

It is expected that this particular simulator may provide inputs in two main directions. Firstly it allows for preliminary testing and tuning new FES protocols without needing for real experiments with patients, and may help for writing the ethical protocols needed for any experiments involving livings. Secondly it can be used to preliminary evaluation of new technologies or implementations, without costly reworking of existing electronic chips or certified components.

The simulation software is open, so that enhancements w.r.t. to the original release can be added upon request of various designers and to fulfill various objectives. Some ideas to be implemented are :

- Refinement of the physical communication medium model, so that the wireless link fading due to of obstacles, e.g. a human body, can be evaluated leading to the design of a feedback control of the emission power of the transceivers;
- Development and test of feedback schedulers for automatic adaptation of the CPU and communication loads along a long range FES application.
- Integration of the full code of the protocol stack in the controller and DSUs, further allowing for a HIL simulation by changing the simulated medium for the real wireless one and going to a HIL simulation;

As for any simulation, a serious problem is the simulation system and models calibration and validation. This process needs, at some points, comparisons with observations of the reality and can be costly. Anyway this is needed to gain the confidence of the various users of the system.

References

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

[2] U.M. Ascher and L.R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, ISBN 0-89871-412-5, 1998.

[3] Abir Ben Khaled-El Feki. *Simulation temps-réel distribuée de modèles numériques : application au groupe motopropulseur*. PhD thesis, EEATS, Automatique Productique, Université de Grenoble, May 2014. `http://www.theses.fr/2014GRENT033`.

[4] M. Benoussaad, K. Mombaur, and C. Azevedo-Coste. Nonlinear model predictive control of joint ankle by electrical stimulation for drop foot correction. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 983–989, Nov 2013.

[5] Mourad Benoussaad. *Protocole d'identification sous FES et synthèse des séquences de stimulation chez le blessé médullaire*. PhD thesis, University Montpellier II, 2009. `http://tel.archives-ouvertes.fr/tel-00452009`.

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

[7] H. K. Fathy, Z. S. Filipi, J. Hagena, and J. L. Stein. Review of hardware-in-the-loop simulation and its prospects in the automotive area. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conf. Series*, volume 6228, 2006.

[8] Thomas Fattal. Etude et métrologie d'un dispositif de stimulation électro-fonctionnelle sans fil. Rapport de stage IUT, Université Montpellier 2, 2010.

[9] C. Faure, M. Ben Gaïd, N. Pernet, M. Fremovici, G. Font, and G. Corde. Methods for real-time simulation of cyber-physical systems : Application to automotive domain. In *7th Int. Wireless Communications and Mobile Computing Conf. IWCMC*, pages 1105 – 1110, Istanbul, Turkey, 2011.

[10] Karen Godary-Dejean, David Andreu, and Richard Romain. Temporal bounds verification of the STIMAP protocol. In *RTNS 2011 : 19th International Conference on Real-Time and Network Systems*, page 10, Nantes, France, September 2011.

[11] Karen Godary-Dejean, David Andreu, and Guillaume Souquet. Sliding Time Interval based MAC Protocol and its Temporal Validation. In IFAC, editor, *FET'07: 7th IFAC International Conference on Fieldbuses & Networks in Industrial & Embedded Systems*, pages 119–126, Toulouse, France, November 2007.

[12] A. C. Hindmarsh and L. R. Petzold. Algorithms and software for Ordinary Differential Equations and Differential-Algebraic Equations, part II: Higher-order methods and software packages. *Comput. Phys.*, 9:148–155, 1995.

[13] W. H. Kwon and S.-G. Choi. Real-time distributed software-in-the-loop simulation for distributed control systems. In *IEEE Int. Symp. on Computer Aided Control System Design CACSD*, pages 115–119, 1999.

[14] Samy Lafnoune. Contrôle d'ordonnancement dans un système de stimulation électrique distribué. Master's thesis, Master2 Robotique, Université Montpellier 2,, August 2014. http://hal-lirmm.ccsd.cnrs.fr/lirmm-01096604.

[15] E.A. Lee. Computing foundations and practice for cyber-physical systems: A preliminary report. Technical Report UCB/EECS-2007-72, University of California, Berkeley, May 2007.

[16] Steven R McFaull and Mario Lamontagne. In vivo measurement of the passive viscoelastic properties of the human knee joint. *Human Movement Science*, 17(2):139 – 165, 1998.

[17] Qin Zhang, Mitsuhiro Hayashibe, and Christine Azevedo Coste. Evoked Electromyography-Based Closed-Loop Torque Control in Functional Electrical Stimulation. *IEEE Transactions on Biomedical Engineering*, 60(8):2299–2307, March 2013.