A Passivity Approach for Model-Based Compositional Design of Networked Control Systems

XENOFON KOUTSOUKOS, NICHOLAS KOTTENSTETTE, JOSEPH HALL, EMEKA EYISI, HEATH LEBLANC, JOSEPH PORTER, and JANOS SZTIPANOVITS,
Institute for Software Integrated Systems, Vanderbilt University

The integration of physical systems through computing and networking has become pervasive, a trend now known as cyber-physical systems (CPS). Functionality in CPS emerges from the interaction of networked computational and physical objects. System design and integration are particularly challenging because fundamentally different physical and computational design concerns intersect. The impact of these interactions is the loss of compositionality which creates tremendous challenges. The key idea in this article is to use passivity for decoupling the control design of networked systems from uncertainties such as time delays and packet loss, thus providing a fundamental simplification strategy that limits the complexity of interactions. The main contribution is the application of the approach to an experimental case study of a networked multi-robot system. We present a networked control architecture that ensures the overall system remains stable in spite of implementation uncertainties such as network delays and data dropouts, focusing on the technical details required for the implementation. We describe a prototype domain-specific modeling language and automated code generation tools for the design of networked control systems on top of passivity that facilitate effective system configuration, deployment, and testing. Finally, we present experimental evaluation results that show decoupling of interlayer interactions.

Categories and Subject Descriptors: J.7 [**Computers in Other Systems**]

General Terms: Design

Additional Key Words and Phrases: Networked control systems, model-based design, passivity

ACM Reference Format:
Koutsoukos, X., Kottenstette, N., Hall, J., Syisi, E., LeBlanc, H., Porter, J., and Sztipanovits, J. 2012. A passivity approach for model-based compositional desingn of networked control systems. ACM Trans. Embedd. Comput. Syst. 11, 4, Article 75 (December 2012), 31 pages.
DOI = 10.1145/2362336.2362342 http://doi.acm.org/10.1145/2362336.2362342

1. INTRODUCTION

The integration of physical systems through computing and networking has become the most pervasive application of Networking and Information Technology (NIT), a trend now known as Cyber Physical Systems (CPS). While this pervasive use of NIT for integrating systems offers an exceptional opportunity for the way we build complex engineering systems, it is an additional source of heterogeneity and complexity. There are unique challenges that emanate from this integration role that are vital for the

This work is supported in part by the National Science Foundation under Grants CNS-1035655 and CCF-0820088, the US Army Research Office under Grant ARO W911NF-10-1-005, the US Air Force Office of Scientific Research under Grant MURI FA9550-06-0312, and Lockheed-Martin.
Authors' address: X. Koutsoukos, N. Kottenstette, J. Hall, E. Syisi, H. LeBlanc, J. Porter, and J. Sztipanovits, Institute for Software Systems, Vanderbilt University, Box 1829, Station B., Nashville, TN 37235; X. X. Koutsoukos (correspondings author) email: xenofon.koutsoukos@vanderbilt.edu.

DOI 10.1145/2362336.2362342 http://doi.acm.org/10.1145/2362336.2362342

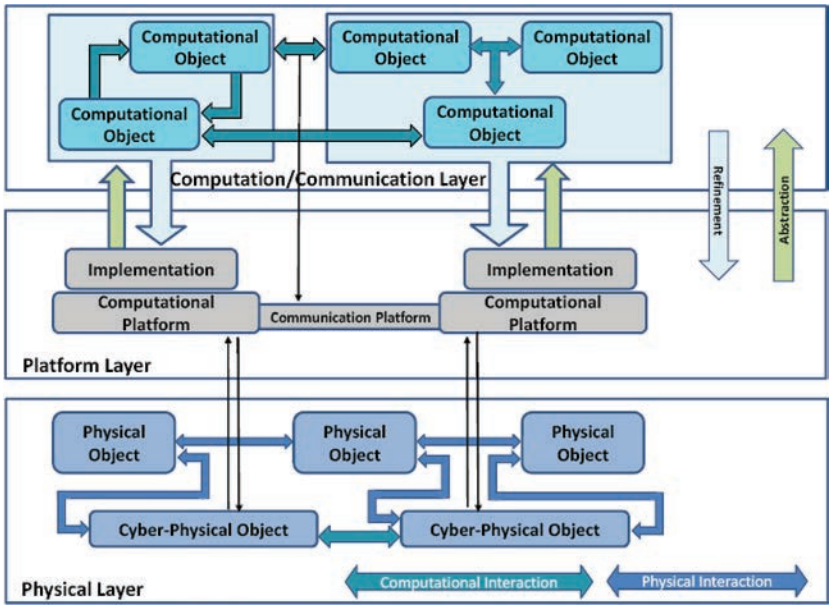


Fig. 1. System layers.

development of CPS. In CPS, functionality emerges from the interaction of networked computational and physical objects. System design and integration are particularly challenging because fundamentally different physical and computational design concerns intersect. Moreover, as the complexity of engineered systems continues to increase, our lack of a systematic theory creates more and more problems. Finding a solution is difficult because essential design concerns—usually separated into hardware, software, and systems and control engineering—come together and the hidden, poorly understood interactions and conflicts suddenly surface.

Building systems from components is a basic method in all engineering disciplines to manage complexity, decrease time-to-market and contain cost. The feasibility of component-based system design depends on two key conditions: *compositionality*, meaning that system-level properties can be computed from local properties of components, and *composability*, meaning that essential component properties do not change as a result of interactions with other components. Component-based design is widely used in both computer science and engineering, but has been most successfully applied in areas where small system size and homogeneity limit complexity. Unfortunately, CPS are inherently heterogeneous in structure, component behaviors, and types of interactions among components. CPS have three principal design layers, the Computation/Communication Layer, Platform Layer, and Physical Layer as shown in Figure 1. The Physical Layer embodies physical components such as plants as well as their interactions expressed in physical time. The design requirements are described using dynamical system models and key properties are stability and performance. The Platform Layer comprises the hardware side of CPS and includes the computation and communication platforms that interact with the Physical Components through sensors and actuators. The Computation/Communication Layer comprises the software components that concentrate much of the complexity in modern systems.

Existing design frameworks typically neglect the interaction across the layers. Establishing stability, for example, can be nontrivial, and much of the focus in systems

theory has been on standardized system architectures such as feedback loops. However, effects of the computational implementation such as network delays and quantization are often neglected. The impact of these neglected interactions is the loss of compositionality which creates tremendous difficulties. Existing approaches are usually unable to deal with the uncertainty introduced by the interaction of heterogeneous components. Compositionality on the controller design layer depends on assumptions that the effects of the implementation uncertainties can be neglected and the models are accurate. Since these assumptions are not satisfied by the implementation layer, the overall design needs to be verified after implementation—even worse—changes in any layer may require re-verification of the full system.

The key idea in this article is to use passivity for decoupling the control design from network uncertainties such as time delays and packet loss, thus providing a fundamental simplification strategy that limits the complexity of interactions. Our objective is to demonstrate a model-based framework for the compositional design of NCS based on the principle of decoupling, a fundamental simplification strategy that limits the complexity of interactions. The main contribution is the application of the approach to an experimental case study of a networked multi-robot system. In such systems, the Physical Layer is concerned with the design of controller dynamics to synchronize multiple systems to perform a global task. Platform and software related timing uncertainties, for example, delays caused by the communication network and jitter caused by the schedulers, will affect the system behavior. To decouple such interactions, we use passivity, a very significant concept from system theory that is fundamental in building systems that are insensitive to implementation uncertainties [Fettweis 1986]. Although the focus is narrowed on the experimental case study, the framework is based on foundational principles that can be used in other CPS domains.

Our team has investigated the use of passivity for the design of networked control systems in Kottenstette et al. [2008, 2011]. This article presents a significant extension of our previous work that is different in multiple aspects. Specifically, in contrast to the work presented in Kottenstette et al. [2008] that considered a single robotic arm controlled via a wireless network, this article is concerned with the control and synchronization of networked multi-robot systems. The objective is to design the network so that the robots follow a reference trajectory provided by the network controller in a synchronized manner in order to achieve some global task. Instead of simply relaying an identical reference to each robot, we couple the robots' positions in a feedback manner to ensure synchronization of the global task. This coupling between the physical plants is essential for achieving synchronization and makes the design completely different and much more challenging than in the case of a single robot controlled via a network. The theoretical foundations of our approach for digital control of multiple passive plants over networks including proofs of stability are presented in Kottenstette et al. [2011]. In order to apply the approach to the experimental networked multi-robot system, we describe how to design local digital controllers that make the robotic agents passive, and then we present the network control architecture including the network controller for the multi-robot system focusing on the control-related technical details required for the implementation. Finally, we address the challenges of compositional design and system integration by developing a model-based framework. In particular, we develop a prototype domain-specific modeling language and automated code generation tools for design of networked control systems on top of passivity, which facilitate effective system configuration, deployment, and testing. A preliminary version of the prototype language for the case of linear plants that includes only a code generator for simulation in Matlab/Simulink using the TrueTime toolbox is presented in Eyisi et al. [2009]. In contrast, this article presents the tools required to design and implement the experimental networked multi-robot system.

In summary, the contribution of this article is the application of a passivity approach for model-based compositional design of NCS to an experimental multi-robot networked system consisting of two robotic manipulators controlled by a human operator using a haptic paddle. In order to explain our methodology, we first present the networked control architecture for control and synchronization of multiple robotic systems based on passivity that ensures the overall system remains stable in spite of implementation uncertainties such as network delays and data dropouts. We describe a prototype domain-specific modeling language and automated code generation tools for model-based compositional design of networked control systems on top of passivity. Finally, we present experimental evaluation results that demonstrate robustness of the approach in the presence of persistent and intermittent network interruptions as well as a comparison study with the case when the robots follow the reference trajectory independently.

The rest of the article is organized as follows. Section 2 compares the proposed approach with related work. Section 3 presents the networked control architecture. Section 4 describes the prototype modeling language. The experimental networked multi-robot system is presented in Section 5. Experimental results are presented in Section 6. Section 7 presents conclusions and future work.

2. RELATED WORK

The last decade has seen the emergence of numerous methods for design of NCS. Passivity has been exploited as a major tool because it provides an inherent safety for controlling complex systems. This section compares the proposed approach with related work in NCS focusing in particular on passivity-related methods. Related work in model-based design of embedded control systems is also presented.

2.1. Networked Control Systems

Cyber-physical systems such as automotive vehicles, building and industrial automation systems, and groups of unmanned vehicles consist of plants, controllers, sensors, and actuators connected over communication networks. Modeling, analysis, and design of NCS has emerged as an important topic in the intersection of control systems, communication networks, and computer science [Antsaklis and Baillieul 2007]. Previous research in NCS has successfully used abstractions for physical processes interacting with computation and communication components for investigating stability, control design, and estimation [Walsh et al. 2002; Lian et al. 2002; Montestruque and Antsaklis 2004; Seiler and Sengupta 2005; Baillieul and Antsaklis 2007]. However, we still lack the ability to systematically integrate heterogeneous components and design NCS in an efficient, systematic, and scalable manner. The focus so far has been on control-theoretic aspects [Hespanha et al. 2007] but the study of the implementation aspects has been limited.

Limitations and uncertainties in the communication channel introduce three main phenomena that need to be addressed: network-induced delays, data dropouts, and quantization error. Network delays and data dropouts present significant challenges in NCS and require the development of novel analysis and synthesis methods. While there have been several methods for characterizing stability and performance (see Hespanha et al. [2007] and the references therein), time-varying delays cause significant challenges. The effects of quantization in feedback control systems have been investigated also at length focusing on control and stabilization, see, for example Brockett and Liberzon [2000] and Nešic and Liberzon [2009]. A method to address delays and quantization using a unified framework has been presented in Nešic and Liberzon [2009]. A recent related approach to our work which aims at decoupling the control design from the implementation layers has been proposed in Skaf and Boyd [2007]. The approach

allows the design of state-feedback controllers that minimize a quadratic performance bound for a given level of timing jitter using linear matrix inequality methods and also allows truncating the coefficients of the controller while guaranteeing that a given set of relaxed performance constraints is met. This approach has been adopted for studying performance degradation caused by time-varying delays in Bhave and Krogh [2008].

The approaches described above typically consider classical control design methods, and in addition, design the quantizers and the communication protocols. Theoretically, the problem is formulated as the construction of appropriate Lyapunov functions which can serve as certificates ensuring stability in the presence of network uncertainties. Although such analysis provides valuable guidelines for designing new protocols, it imposes assumptions that do not necessarily hold in existing communication protocols. For example, a sufficiently small upper bound on the inter-transmission intervals is essential for guaranteeing stability [Nešic and Liberzon 2009]. On the other hand, the main idea in our work is that by imposing passivity on the component dynamics, the design becomes insensitive to network effects, thus decoupling the controller design and implementation design layers with respect to network effects. This separation of concerns empowers the model based design process to be used for networked control systems without imposing additional constraints on the communication protocols. It should be noted that passive structures have additional advantages with regard to robustness in the presence of finite length representations and saturation [Fettweis 1986] but are not considered in this article.

2.2. Passivity

Stability is, of course, a primary concern when designing complex systems such as NCS. In physical processes, stability can be analyzed using energy conservation laws. The theory of passive dynamical systems provides a strong foundation for a compositional framework for stability [Haddad and Chellaboina 2008; van der Schaft 1999]. A passive system is one from which the net scaled energy that can be extracted is finite, i.e., a passive system only stores and dissipates energy but cannot generate energy of its own. Further, passive systems have a unique property that when connected in either a parallel or negative feedback manner the overall system remains passive [Bao and Lee 2007]. Passive control theory is very general and broad and applies to a large class of controllers for linear, nonlinear, continuous and discrete-time control systems. The inherent safety of passive systems has been widely exploited by researchers in human interacting machines like smart exercise machines and teleoperated devices (e.g., Li and Horowitz [1997] and Niemeyer and Slotine [2004]).

Passivity provides significant advantages dealing with network delays, data dropouts, and quantization and has been used already for NCS design [Gao et al. 2008]. In particular, previous work performed mainly in telerobotics has helped guide our research. The problem of bilateral teleoperation of a single controller/plant pair for both continuous and discrete time communication channels is studied in Chopra et al. [2008]. This work demonstrates the design of a master-slave robotic telemanipulation system and shows that passivity in the communication channel can be maintained with both fixed and time-varying (increasing) delays provided dropped data and reordering of data are properly handled. The approach is limited to one plant and one controller and does not address the synchronization of multiple robots. The approach has been extended in Chopra and Spong [2006] to interconnect continuous-time robotic systems which communicate over a balanced-directed graph and assumes continuous dynamics and fixed-time delays in order to achieve tracking. The approach presented in Hirche et al. [2009] demonstrates how to interconnect a continuous-time controller to a continuous-time plant while maintaining stability for arbitrary fixed time delays. Although there are conjectures that the results can be extended to transmitting

discrete-time wave variables over a network using passivity-based compression-decompression schemes [Hirche and Buss 2007], a discrete-time implementation of the controller is not specified. An approach which allows one to connect discrete-time controllers to continuous-time plants is presented in [Stramigioli et al. 2005]. The interconnection between discrete and continuous port-Hamiltonians systems is considered and a method which preserves passivity even with dropped data and time-varying delays is presented. The method is applied to telemanipulation and haptic interfaces, and simulations are used to show the resilient behavior to dropped data and time varying delays.

Passivity has been used also for group coordination of multi-agent systems. A passivity-based design framework for controlling nonlinear systems which have been rendered passive through an internal feedback configuration is presented in Arcak [2007]. The design procedure is applied to group coordination problems and group agreement (consensus). The framework has been applied to synchronize a group of rigid bodies in Bai et al. [2008] and to combine formation control with path following in Ihle et al. [2007]. This work does not consider time-varying delays and is illustrated using simulations, however, it shows that passivity is a major tool that can be used for group coordination of networked control systems. We have extended our passivity-based approach for networked multi-agent systems and have shown that a formation around a target can be established in a stable manner in the presence of network delays and packet loss [LeBlanc et al. 2010].

In addition to the differences from previous work on NCS design based on passivity described as just, the proposed approach exhibits two salient features that are novel and facilitate the application of the model-based design and the implementation to the experimental networked multi-robot system. First, we use the power junction to interconnect digital controllers to multiple discrete time plants over a network in a negative feedback manner such that the total power input is always greater than or equal to the total power output and passivity is ensured. The power junction allows stability analysis in the presence of time-varying delays and data loss, is a pivotal component in the model-based design framework because it captures the interactions between plants and controllers, and is configured as a group of servers in the experimental testbed allowing automated code generation, system configuration, and deployment. Second, we introduce a passive up-sampler and a passive down-sampler in order to reduce the amount of network traffic while implementing a systematic method for interconnecting the controller with multiple plants using wave-variables. The passive up-sampler and down-sampler are used to match the bandwidth constraints of the network while reconciling different rates at local and network controllers.

2.3. Model-Based Design

Model-based design for embedded control systems involves creating models and checking correctness at different stages in the development process [Karsai et al. 2003]. Model-based design flow progresses along precisely defined abstraction layers, typically starting with control design followed by system-level design for the specification of platform details, code organization, and deployment details, and the final stage of integration and testing on the deployed system. This design approach cannot be applied directly to NCS because domain heterogeneity and tight coupling between design concerns create a number of challenges. Ensuring controller stability and performance for physical systems in the presence of network uncertainties (e.g. time delay, packet loss) couples the control and system-level design layers. In addition, downstream code modifications during testing and debugging invalidate results from earlier design-time analysis and any component change often results in "restarting" the design process.

A number of research projects seek to address the problems of model-based design for NCS. The ESMoL modeling language for designing and deploying time-triggered

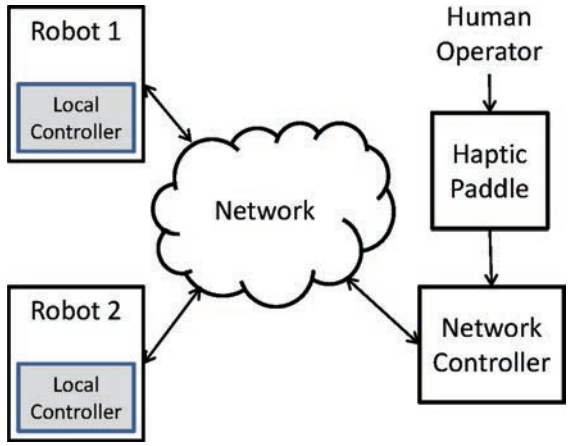


Fig. 2. Networked control system.

control systems explicitly captures in model structure many of the essential relationships in an embedded design [Porter et al. 2008]. The ESMoL tools include schedule determination for time-triggered communications, code generation, and a portable time-triggered virtual machine. AADL [AS-2 Embedded Computing Systems Committee 2004] is a textual language and standard for specifying deployments of control system designs in data networks [Hudak and Feiler 2007]. AADL projects also include integration with verification and scheduling analysis tools. The Metropolis modeling framework [Balarin et al. 2003] aims to give designers tools to create verifiable system models. Metropolis integrates with SystemC, the SPIN model-checking tool, and other tools for scheduling and timing analysis.

The main contribution of the article compared with existing model-based design approaches is a domain specific modeling language which exploits the advantages stemming from passivity-based design. Our approach supports forward generation of platform-specific simulation models and executable models while ensuring global stability (in a robust way) by a combination of component analysis and specific rules for composition of passive components. By imposing passivity constraints on the component dynamics, the design becomes insensitive to network effects decoupling the controller design and implementation design layers, and thus facilitating efficient deployment and testing.

3. NETWORKED CONTROL ARCHITECTURE

This section presents the networked control architecture for a system consisting of multiple robotic agents serving as physical plants controlled by a network controller. The objective is to design the NCS so that the robots follow a reference trajectory provided by the network controller in a synchronized manner in order to achieve some global task. The system is shown in Figure 2. Each robotic agent consists of a robotic arm and a local controller. The robots communicate with the network controller over a communication channel. The network controller receives a desired reference trajectory from the operator, for example, using a haptic paddle, and is responsible for making sure each robotic agent tracks the desired trajectory while it remains synchronized with the other agent. Instead of simply relaying an identical reference to each robot, we couple the robots' positions in a feedback manner to ensure synchronization of the global task. Each robot takes into consideration not only the desired command by the network controller but also the positions of the other robots in order to move

in a synchronized manner. This coupling between the physical plants is essential for achieving synchronization and makes the passivity-based design much more challenging than in the case of a single robot controlled via a network [Kottenstette et al. 2008]. The main challenge is how to feed back possibly delayed and incomplete information in order to modify the desired trajectories at the local controllers and perform the global task in an uncertain environment while the overall system remains stable. After stability is ensured, the controllers can be tuned to optimize performance.

In the following, after a brief background on passivity, we present the control architecture focusing on the technical details required for the implementation of the experimental networked multi-robot system. The theoretical foundations including proofs of stability can be found in Kottenstette et al. [2011]. In this article, we first describe how to design local digital controllers that make the robotic agents passive, and then we briefly describe the network control architecture concluding with the network controller for the multi-robot system.

3.1. Background on Passivity

There are various precise mathematical definitions for passive systems [Kottenstette and Antsaklis 2010]. Essentially all the definitions state that the output energy must be bounded so that the system does not produce more energy than was initially stored [Haddad and Chellaboina 2008; van der Schaft 1999]. Strictly output passive systems and strictly input passive systems with finite gain have a special property in that they are ℓ^2-stable. Also, passive systems are Lyapunov-stable in terms of all observable states. Passive systems have a unique property that when connected in either a parallel or negative feedback manner the overall system remains passive. By simply closing the loop with any positive definite matrix, any discrete time passive plant can be rendered strictly output passive. This is an important result because it makes it possible to directly design low-sensitivity strictly-output passive controllers using the wave digital filters described in Fettweis [1986].

When delays are introduced in negative feedback configurations, the network is no longer passive. One way to recover passivity is to interconnect the two systems with wave variables. Wave variables define a bilinear transformation under which a stable minimum phase continuous system is mapped to a stable minimum phase discrete-time system, and thus, the transformation preserves passivity. They were introduced in order to circumvent the problem of delay-free loops and guarantee that the implementation of wave digital filters is realizable [Fettweis 1986]. Wave variables allow interconnecting a plant and a controller over a network while preserving passivity subject to arbitrary fixed time delays and data dropouts. If additional information is transmitted along with the continuous wave variables, the communication channel will also remain passive in the presence of time-varying delays [Niemeyer and Slotine 2004]. We have developed a method for controlling multiple plants over a network while maintaining passivity in the presence of delays and data dropouts in Kottenstette et al. [2011], which is used in our control architecture.

3.2. Robotic Control

The first step is the design of a local digital controller so that each robotic agent possesses a passive input-output interface. The robot dynamics (denoted by the passive mapping $H_{\dot{\theta}} : \tau \rightarrow \dot{\theta}$ in Figure 3) are defined by

$$\tau = M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + g(\theta).$$

The state variables θ denote the robot joint angles, τ is the input torque vector, $M(\theta)$ is the mass matrix, $C(\theta, \dot{\theta})$ is the matrix of centrifugal and Coriolis effects, and $g(\theta)$ is the gravity vector [Ortega and Spong 1988].

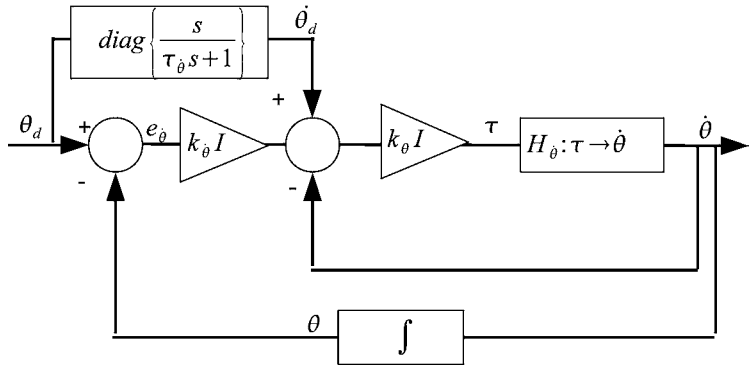


Fig. 3. Continuous robotic control structure.

An idealized continuous control structure is shown in Figure 3. The control structure enforces stability and the robotic-joint angular position θ tracks the desired angular position θ_d at steady-state. The filter, denoted by

$$\text{diag}\left\{\frac{s}{\tau_{\dot{\theta}}s+1}\right\} = \text{diag}\left\{\hat{H}_{\dot{\theta}}(s)\right\},$$

is used to estimate the desired velocity set-point for each joint. Since $H_{\dot{\theta}}$ is considered passive and an integrator is also a passive mapping, then the control structure is stable for any $0 < k_\theta,\ k_{\dot{\theta}}, \tau_{\dot{\theta}} < \infty$ [Kottenstette and Porter 2009]. Practically, however, additional dynamics due to the motors which apply torque to the robot, discrete-time sampling, and quantization effects will effectively limit how large k_θ and $k_{\dot{\theta}}$ can be before instabilities arise. The control gains can be tuned in a simple manner to assure that the overall system is stable.

In order to implement this idealized control system in the networked control architecture, we need to develop a digital control structure that is still passive. Each robotic joint is driven by a servo controller which reports both angular position and velocity. Each servo controller uses a finite-impulse response filter to estimate velocity. However, during operation the velocity-estimator of servo-controllers in the feedback loop introduce significant (and destabilizing) group delay [Oppenheim et al. 1997]. To account for this delay, we synthesize an infinite-impulse response filter whose magnitude and phase response closely matches the continuous-time filter it is emulating (see Figure 3) at a sample rate T_s using the inner-product equivalent sample and hold (IPESH) transform [Kottenstette et al. 2011], thus ensuring passivity of the digital filter. The resulting digital filter $\hat{H}_{\dot{\theta}}(z)$ is defined by

$$\hat{H}_{\dot{\theta}}(z) = \frac{1-p_{\dot{\theta}}}{T_s}\frac{z-1}{z-p_{\dot{\theta}}},\ p_{\dot{\theta}} = \exp\left(-\frac{T_s}{\tau_{\dot{\theta}}}\right).$$

The digital control structure for each robot is shown in Figure 4. Although $H_\theta : \theta_d(i) \rightarrow \theta(i)$ is a stable system, it is not passive. In order to interconnect stable systems over a network in a feedback manner and not be subject to potentially destabilizing delays and data dropouts we need to modify them in a manner such that their input-output mapping is passive. We add a high-pass filter in order to create a passive system as shown in Figure 5 (denoted $H_{\theta_p} : \theta_d(i) \rightarrow \theta_p(i)$). We only pass the higher frequencies through $H_\gamma(z)$ (with near zero phase shift) in order to compensate for the non-passive behavior of H_θ at those frequencies. The filter is designed first in the continuous domain

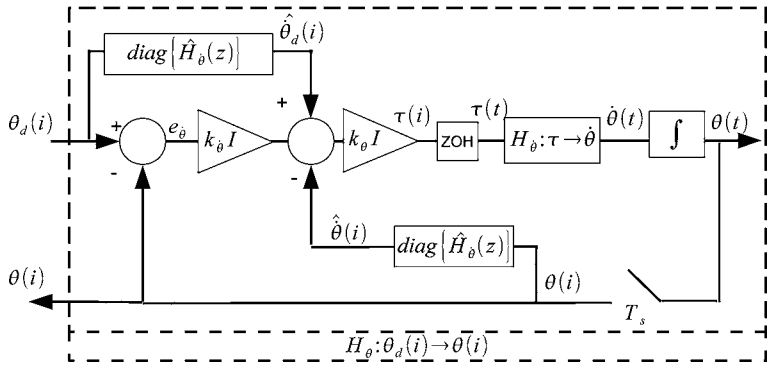


Fig. 4. Digital robotic control structure.

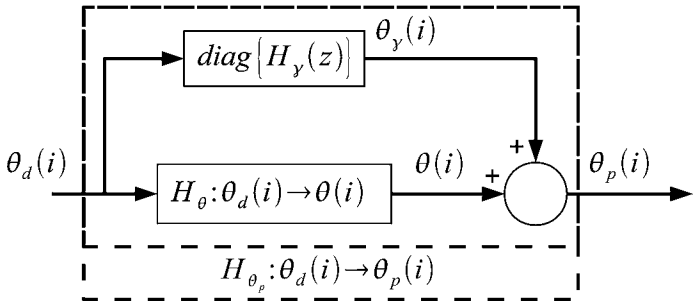


Fig. 5. Passive digital robotic map.

and transformed using the IPESH to obtain

$$H_{\gamma}(z)=\frac{\gamma(1-p_{\gamma})}{\omega_{\gamma}T_s}\frac{z-1}{z-p_{\gamma}},\ p_{\gamma}=\exp(-T_s\omega_{\gamma}).$$

The resulting control structure achieves $\theta(i)\approx\theta_p(i)$ during lower-frequency transients, $\theta(i)=\theta_p(i)$ at steady-state, and preserves passivity for higher-frequency transients.

3.3. Multi-Rate Digital Control Network

The network control structure is shown in Figure 6. A network controller denoted H_{c1} and two robotic systems denoted $H_{\theta_{p2}}$ and $H_{\theta_{p3}}$ respectively are connected to an averaging power junction denoted PJ.[1] The robots use the local digital control structure described in the previous subsection so that they are passive and the controller is designed to be passive. The network controller runs at a slower rate than the robot controllers to account for the communication bandwidth constraints. A passive down sampler (denoted $\mathsf{PDS_k}$:M) and a passive up-sampler (denoted $\mathsf{PUS_k}$:M) are used to reconcile the rates, where $\mathsf{k}\in\{2,3\}$ is the robot index and M is a positive integer describing the ratio of the two rates. Information is transmitted in the form of wave variables that are computed using a bilinear transformation (denoted b in the figure). Finally, the network delays at the various links are denoted as z^{-c_k} and z^{-p_k}.

[1]The common index in the controller and the robotic systems is used to simplify the notation of the various variables transmitted in the network.

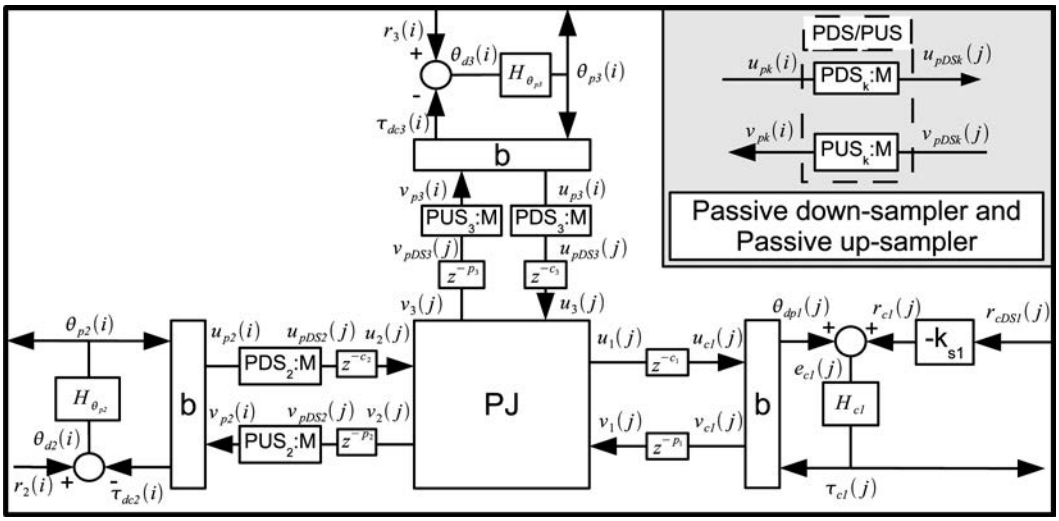


Fig. 6. Passive multi-rate digital control network.

3.3.1. Wave Variables. Wave variables are used to transmit torque and angular position over the network. These variables are required to maintain passivity of the feedback loop when subject to arbitrary fixed time delays and data dropouts [Niemeyer and Slotine 2004; Anderson and Spong 1992]. The primary advantage of using wave variables is that they tolerate most time-varying delays, such as those that occur when using the TCP/IP transmission protocol.

The sensor output in a wave variable form for each passive robotic system $H_{\theta_{\mathsf{pk}}}$ is

$$u_{\mathsf{pk}}(i) = \frac{1}{\sqrt{2b}}(b\theta_{\mathsf{pk}}(i) + \tau_{\mathsf{dck}}(i)),\ \mathsf{k} \in \{2, 3\},$$

and the command output of the controller H_{c1} is

$$v_{c1}(j) = \frac{1}{\sqrt{2b}}(b\theta_{\mathsf{dp1}}(j) - \tau_{c1}(j)).$$

Based on these equations, the wave variable transformation (denoted as b in Figure 6) is implemented as

$$\begin{bmatrix} u_{\mathsf{pk}}(i) \\ \tau_{\mathsf{dck}}(i) \end{bmatrix} = \begin{bmatrix} -I & \sqrt{2b}I \\ -\sqrt{2b}I & bI \end{bmatrix} \begin{bmatrix} v_{\mathsf{pk}}(i) \\ \theta_{\mathsf{pk}}(i) \end{bmatrix}$$

and

$$\begin{bmatrix} v_{c1}(j) \\ \theta_{\mathsf{dp1}}(j) \end{bmatrix} = \begin{bmatrix} I & -\sqrt{\frac{2}{b}}I \\ \sqrt{\frac{2}{b}}I & -\frac{1}{b}I \end{bmatrix} \begin{bmatrix} u_{c1}(j) \\ \tau_{c1}(j) \end{bmatrix},$$

where b is a positive constant of the impedance of the scattering transformation.[2]

3.3.2. Passive Up-Sampler and Down-Sampler. Because of bandwidth constraints, the local digital controllers for each robot run at a faster rate than the network controller. We

[2]By abuse of notation, we use the same symbol for the scattering transformation and the impedance as customary.

denote T_s the sampling period of the local controllers and MT_s the sampling period of the network controller where M is a positive integer. The corresponding time indexes are $i = \lfloor \frac{t}{T_s} \rfloor$ and $j = \lfloor \frac{t}{MT_s} \rfloor$ for the robots and controller respectively, and are related by $j = \lfloor \frac{i}{M} \rfloor$.

In order to preserve passivity in the multi-rate digital control network, the passive upsampler (PUS) and passive downsampler (PDS) are implemented as follows.

—PDS

$$u_{\mathsf{pDSk}_k}(j) = \sqrt{\sum_{i=M(j-1)}^{Mj-1} u^2_{\mathsf{pk}_k}(i)}\,\mathrm{sgn}\left(\sum_{i=M(j-1)}^{Mj-1} u_{\mathsf{pk}_k}(i)\right),$$

—PUS

$$v_{\mathsf{pk}_k}(i) = \sqrt{\frac{1}{M}}\, v_{\mathsf{pDSk}_k}(j-1),\ i = Mj, \ldots, M(j+1)-1,$$

where $u_{\mathsf{pk}}, u_{\mathsf{pDSk}} \in \mathbb{R}^{m_s}$, and each k^{th} element within the respective vector $u_{\mathsf{pk}}, u_{\mathsf{pDSk}}$ is denoted $u_{\mathsf{pk}_k}, u_{\mathsf{pDSk}_k},\ k \in \{1, \ldots, m_s\}$.

3.3.3. The Averaging Power Junction. The power junction is used to interconnect wave variables from multiple controllers and plants such that the total power input is always greater than or equal to the total power output. The power constraints ensures that passivity is preserved when composing a network in which multiple passive plants can be interconnected to multiple passive controllers. In our architecture, the controller and the two robotic systems which are interconnected to the averaging power junction have corresponding wave variable pairs $(u_1, v_1), (u_2, v_2), (u_3, v_3)$. The *power-output* pairs are denoted (u_1, v_1) in which $u_1 \in \mathbb{R}^{m_s}$ is an outgoing wave and $v_1 \in \mathbb{R}^{m_s}$ is an incoming wave. The *power-input* pairs are denoted $(u_{\mathsf{k}}, v_{\mathsf{k}}),\ \mathsf{k} \in \{2, 3\}$ in which $u_{\mathsf{k}} \in \mathbb{R}^{m_s}$ is an incoming wave and $v_{\mathsf{k}} \in \mathbb{R}^{m_s}$ is an outgoing wave from the averaging power junction. Each l^{th} component ($l \in \{1, \ldots, m_s\}$) of the outgoing waves v_{k} (denoted v_{k_l}) are computed from the respective l^{th} component of the incoming waves $v_{\mathsf{j}}.\ \mathsf{j} \in \{m\}$ (denoted v_{j_l}) as follows:

$$\begin{aligned}
\mathsf{sf}_v &= \frac{|\sum_{\mathsf{j}=1}^{m} v_{\mathsf{j}_l}|}{\sum_{\mathsf{j}=1}^{m} |v_{\mathsf{j}_l}|},\\
v_{\mathsf{k}_l} &= \mathsf{sf}_v \mathrm{sgn}\left(\sum_{\mathsf{j}=1}^{m} v_{\mathsf{j}_l}\right) \frac{\sqrt{\sum_{\mathsf{j}=1}^{m} v^2_{\mathsf{j}_l}}}{\sqrt{n-m}} = \frac{v_{1_l}}{\sqrt{3-1}},\ \mathsf{k} \in \{2, 3\}.
\end{aligned}$$

Similarly, each l^{th} component ($l \in \{1, \ldots, m_s\}$) of the outgoing waves u_{j} (denoted u_{j_l}) are computed from the respective l^{th} component of the incoming waves u_{k} (denoted u_{k_l}) as follows:

$$\begin{aligned}
\mathsf{sf}_u &= \frac{|\sum_{\mathsf{k}=m+1}^{n} u_{\mathsf{k}_l}|}{\sum_{\mathsf{k}=m+1}^{n} |u_{\mathsf{k}_l}|},\\
u_{\mathsf{j}_l} &= \mathsf{sf}_u \mathrm{sgn}\left(\sum_{\mathsf{k}=m+1}^{n} u_{\mathsf{k}_l}\right) \frac{\sqrt{\sum_{\mathsf{k}=m+1}^{n} u^2_{\mathsf{k}_l}}}{\sqrt{m}},\\
u_{1_l} &= \mathsf{sf}_u \mathrm{sgn}\left(\sum_{\mathsf{k}=2}^{3} u_{\mathsf{k}_l}\right) \sqrt{\sum_{\mathsf{k}=2}^{3} u^2_{\mathsf{k}_l}}.
\end{aligned}$$

The $k_{s1} > 0$ parameter, shown in Figure 6, is an appropriate scaling term to account for the effects of the averaging power junction computed as $k_s = \sqrt{2M}$.

3.3.4. Network Controller. The network controller contains a digital lag-compensator $\text{diag}\{H_{c1}(z)\}$ which minimizes the steady state error between the reference position and the actual position of every joint of each robotic agent. In order to ensure passivity, the controller is synthesized by applying the IPESH-transform to a continuous-time lag-compensator model and is given by

$$H_{c1}(z) = k\left(1 + \omega\frac{T_s}{2}\frac{z+1}{z-1}\right).$$

The passivity-based approach ensures that the multi-rate control network shown in Figure 6 is passive if the plants and the controller are passive and l_2^m*-stable* if the plants and the controller are strictly-output passive (details can be found in [Kottenstette et al. 2011]).

4. MODEL-BASED DESIGN

Model-based design flow progresses along precisely defined abstraction layers, starting with control design [Karsai et al. 2003]. Control design models are passed on to the system-level design stage for the specification of platform details, code organization, and deployment details. The final stage involves integration and testing on the deployed system. To address the challenges of compositional design of NCS, we present a model-based framework based on the proposed control architecture. We use the Generic Modeling Environment (GME) from the Model Integrated Computing (MIC) tool suite [Karsai et al. 2003; Ledeczi et al. 2001a] to create a domain-specific modeling language (DSML) called the Passivity-based Networked Control Systems (PaNeCS) language. GME provides a metamodeling environment similar to UML. The class stereotypes are defined as follows. Models are entities which may contain other objects while Atoms are indivisible entities which cannot contain other objects; Connections are association classes used to describe the relationship between two entities and they represent a line that connects two entities of a model. Connectors signified by "." specify a visualization for a connection in the model. Each of the entities associated with the connector have well defined roles (src and dst) with respect to the connector. These roles define the direction of the connection between the entities.

4.1. Overview of PaNeCS

PaNeCS raises the level of abstraction of NCS design and allows automated code generation, system configuration, deployment, and testing. More importantly, it facilitates effective engineering processes and methods for designing, building, and analyzing NCS by utilizing the compositionality across design views stemming from the underlying passivity principles. The prototype language defines the allowable connections between components to ensure that any networked control system model built using passive components will be passive. PaNeCS provides the flexibility to easily model networked control systems and configure the system parameters. It also allows testing by running different experiments under various network conditions by simply adjusting parameters to generate appropriate software for each configuration. Although PaNeCS can be used to define general NCS [Eyisi et al. 2009], here we present the model components for the networked multi-robot system.

4.1.1. Components. The language top level consists of eight main components: the *PhysicalPlant*, *PhysicalReference*, *PlantSystem*, *ControllerSystem*, *PowerJunction*, *ReferenceSystem*, *Processor*, and *Network*.

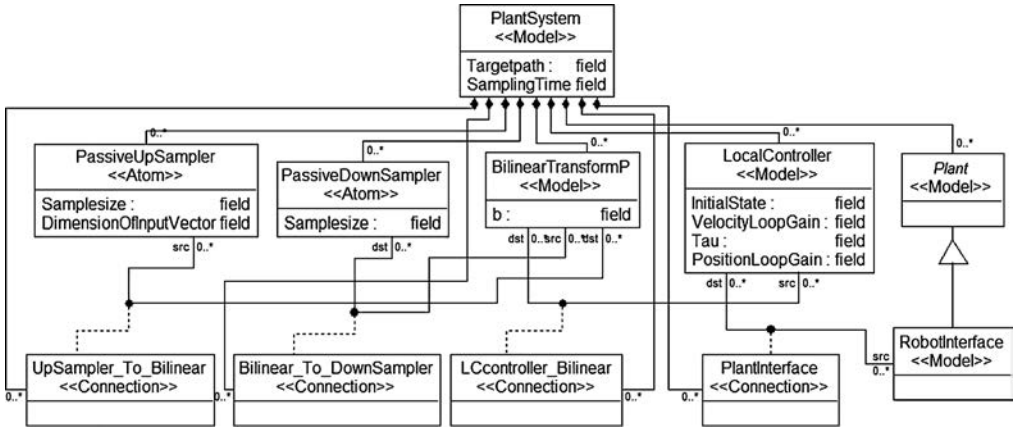


Fig. 7. PlantSystem components language.

PhysicalPlant represents the robotic system controlled over a network. PhysicalReference represents the Haptic Paddle which is used to generate the desired reference trajectory. Figure 7 shows the parts of the metamodel that describe the PlantSystem. *RobotInterface* depicts the software interface for connecting to the PhysicalPlant. *Local Controller* represents the software component for implementing the local control commands that are sent to the robot and also for processing the sensor data received from the robot. The *Local Controller* is parameterized by the initial state of the robot, q_0, the position loop gain, K_x, the velocity loop gain, K_v and the time constant, τ. These attributes are used for configuring the passive controller for the robot. *BilinearTransformP* models the wave variable transformations presented in Section 3.3.1 and it is parameterized by the impedance parameter b. *PassiveUpSampler* and *PassiveDownSampler* pair represent the components for implementing the multi-rate digital control network presented in Section 3.3.2. The *PassiveUpsampler* is parameterized by the sample size attribute denoted as *Sample* and the dimension of the input vector denoted as *DimensionOfInputVector*. The *PassiveDownSampler* is parameterized by the sample size attribute *Sample*. Figure 8 shows the language of the *Ports* of the *PlantSystem*. The *Send* and *Receive* blocks indicate the ports for sending and receiving wave variables from the power junction respectively. The two blocks are both parameterized by the port number parameter denoted as *PortNo*. Additionally, the *Receive* block has a parameter, *VectorLength*, which specifies the length of the vector of the data that is received.

Figure 9 shows the parts of the metamodel which describe the ControllerSystem. *Digital Controller* represents the digital control structure for the robots and the network controller described in Section 3.2 and 3.3.4 respectively. The *Digital Controller* is parameterized by the control gain parameter, K and the control design frequency, w, the cutoff frequency, w_n and the damping coefficient, ζ for the filter used in smoothing out the reference trajectory. *BilinearTransformC* is similar to the *BilinearTransformP* in the PlantSystem. Likewise, the ControllerSystem contains *Send* and *Receive* similar to the PlantSystem for sending and receiving wave variables. Additionally, in the ControllerSystem a *Receive* block also indicates a port for receiving reference signals.

The PowerJunction models the components for implementing the interconnections of the PlantSystem and the ControllerSystem following the description in Section 3.3.3. The PowerJunction has two types of entities representing the two possible interconnections to the power junction. *PowerInputPowerOutput* entity models the software component for interconnecting the plants and the power junction while *PowerOutputPowerInput* entity models the software component for interconnecting the controller

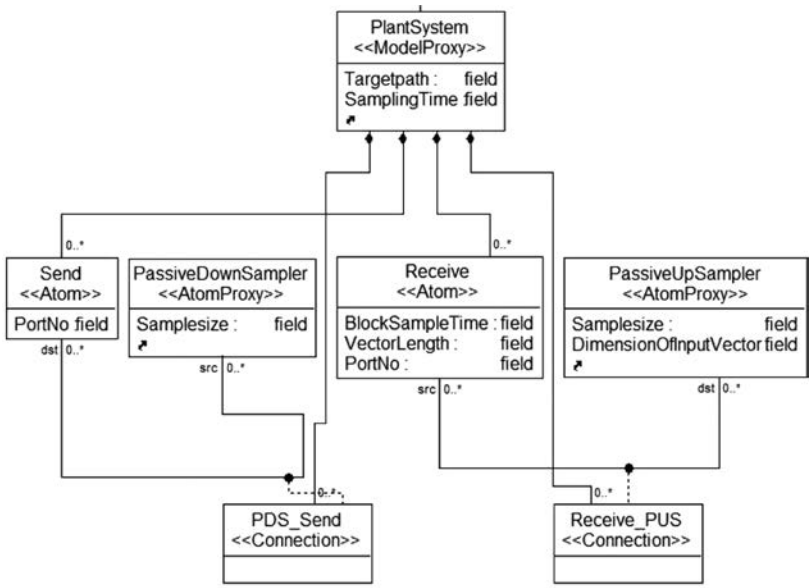


Fig. 8. PlantSystem ports language.

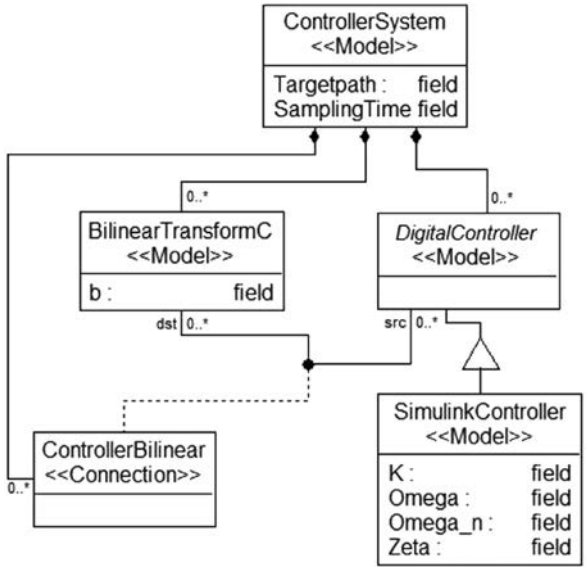


Fig. 9. ControllerSystem components language.

and the power junction. *PowerInputPowerOutput* and *PowerOutputPowerInput* are both parameterized by the send and receive port numbers denoted as *SndPortNo* and *RcvPortNo*, respectively.

Figure 10 shows the part of the language which describes the ReferenceSystem. *HapticInterface* models the software interface for connecting to the haptic paddle. *InverseKinematics* models the software component for computing the inverse kinematics of the trajectory received from the Haptic paddle. This software component translates

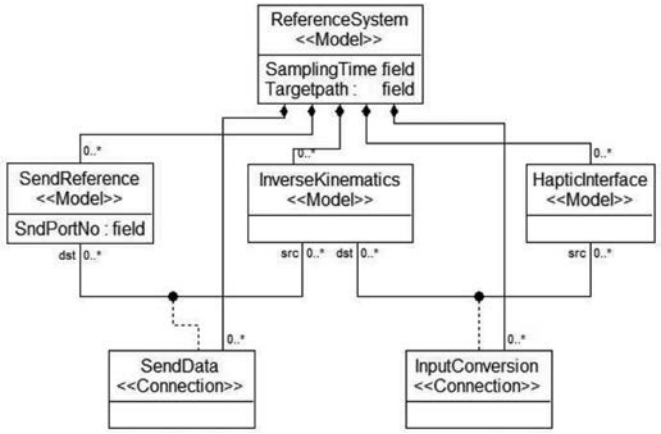


Fig. 10. ReferenceSystem language.

the *x-y-z* coordinates from the Haptic paddle to the required joint angles for the robotic arms [Craig 1989]. *SendReference* models the ports for sending the reference signal to the controller. It is parameterized by the port number denoted as *SndPortNo*.

The Processor models the computer on which the computations and software components are executed. It is parameterized by the IP address of the computer denoted as *IPAddress*. The Network entity models the type of network used for the control system. It is parameterized by *NetworkType* which provides an option for choosing either to use a wired Ethernet or a wireless network.

4.1.2. Aspects. PaNeCS has three main design views or aspects: *Control Design Aspect*, *Platform Aspect*, and *Processor Assignment Aspect*.

The Control Design Aspect visualizes the control design modeling layer. The entities in this view include the PhysicalPlant (Robotic arms), PhysicalReference (Haptic paddle), PlantSystem, ControllerSystem, PowerJunction, and ReferenceSystem as well as their interconnections indicating the flow of control, sensor, and reference signals.

The Platform Aspect visualizes the physical platform layer. This model view shows the hardware components which are used to implement the the NCS. The entities in this view include the Processors, the PhysicalPlant, PhysicalReference, and the Network as well as their interconnections indicating the flow of data packets over the network.

The Processor Assignment Aspect depicts the mapping of the software components to processors on which the computations and implementations are to be performed. The entities in this view include the Processors, PlantSystem, ControllerSystem, PowerJunction and ReferenceSystem. Though the PhysicalPlant and PhysicalReference appear in this aspect, they represent physical entities rather than control design components.

4.1.3. Structural Semantics. The language semantics require constraints that cannot be captured with the metamodeling notations. Using the Object Constraint Language (OCL), we can specify well-formed rules for models, thereby guaranteeing that models created using our approach are "correct by construction". The power junction restricts the possible interconnections between controllers and plant in a negative feedback manner such that the total power input is always greater than or equal to the total power output and passivity is ensured. Only valid connections are allowed to the power junction, so any interconnected system of passive components will be globally passive.

The modeling language and its constraints specify the composition rules greatly reducing the analysis burden for determining passivity, and hence stability, of the composed system. Practically, the constraints are enforced by GME at design time when creating the models.

We implemented three classes of constraints Cardinality Constraints, Connection Constraints, and Unique Name Constraints. *Cardinality Constraints* ensure that the required and correct number of components are used in each relation in the design. For example, for each PlantSystem model, there must be one *LocalController* model. The *Connection Constraints* restrict the number of allowable connections between components. For example, for each PlantSystem model, there must be a bidirectional connection between the *RobotInterface* model and the *LocalController* model. The *Unique Name Constraints* essentially ensure the uniqueness of names of components in the various layers of the model design.

An example of the OCL constraint is shown below. This constraint specifies that for each PlantSystem there should be only one bidirectional connection between the *LocalController* model and the *BilinearTransformP* model.

```
Description: There must be only one bidirectional connection between the
LocalController and the BilinearTransformP

Equation:
self.connectionParts(''LCcontroller_Bilinear'').size()=1
```

4.1.4. Code Generation, Deployment, and Execution. We develop a code generator that can be used to synthesize software for integration, deployment, and testing of the networked system. The code generator is developed in C++ using the Builder Object Network (BON2) API provided with GME [Ledeczi et al. 2001b]. The code generator implements translation rules to produce executable Simulink models for the experiments. It traverses all the entities of an instance model and extracts model parameters. These parameters and the model structure are used to generate MATLAB files for configuring and building Simulink models. Simulink is selected because it provides convenient features for rapid prototyping and testing that include simple creation of controllers and interfacing with the robots. Other types of executable code can be generated in a similar manner.

The Simulink models alone are not sufficient to set up the network infrastructure. The deployment model, which can be visualized through the Processor Assignment Aspect, describes assignments of models to processors, so we also generate bash scripts from the deployment model (one for each PC) in order to set up and run the experiments. The scripts handle the network configuration on each node, and then start up the model with the proper parameters.

The network infrastructure utilizes Netcat and SSH. The power junction is configured with a group of servers, and the plant and controller models use client sockets to attach to the power junction. The client connections transmit data over TCP through a secure shell tunnel to the power junction. This technique hides many of the network configuration details from our setup. The Netcat instances serve the purpose of adapting socket types (i.e. client to server and TCP to UDP) as well as making the Simulink socket connections more robust to failures. A typical script sets up the SSH tunnel first, runs Netcat to adapt the sockets, and then runs the model using MATLAB. When the model comes up it finds all of the necessary socket connections available, whether or not the other models have started yet.

The network system follows a globally asynchronous locally synchronous execution model. Components are executed locally in a synchronous manner based on the local

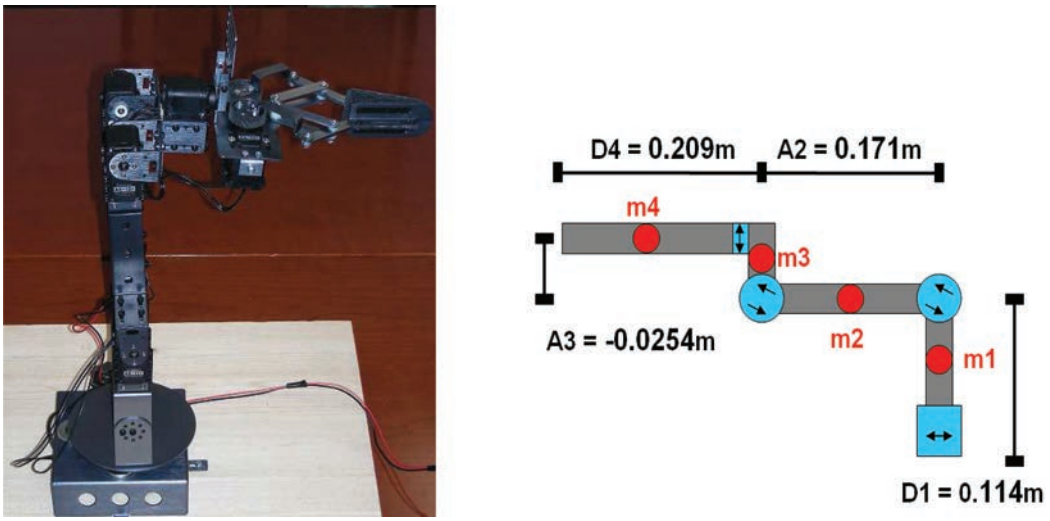


Fig. 11. CrustCrawler 4 DOF arm.

sampling period. The controller and plant receivers execute periodically. To preclude the possibility of blocking, zeros are supplied for missing data values to avoid introducing energy into the system, thereby preserving passivity. It is assumed that data messages will not arrive out of order. Our implementation uses secure TCP links between PCs. Except in extreme overload conditions, message order is maintained.

Another concern is buffer sizing. All data supply and consumption rates are known and adequate for nominal operation. As long as the PC clocks remain relatively close to each other, buffers will never grow without bound. However, currently we do not provide any guarantee for non-ideal operation. If a message receiver crashes or is otherwise delayed, then the sender could continue to fill the intervening buffer indefinitely (also leading to a crash). We can handle this contingency by having the sender drop unsent data instead of storing it.

5. NETWORKED MULTI-ROBOT SYSTEM

This section describes the experimental Networked Multi-Robot System (NMRS) used to demonstrate our approach. The system is modeled using our prototype modeling language and the model is used to generate and integrate the software components required for the experiments.

5.1. Experimental Setup

The experimental setup consists of two CrustCrawler robotic arms [Crustcrawler.com 2009] and one Novint haptic paddle [SIRSLab 2009] connected using a networked computing platform. The computing platform consists of five networked Windows PCs with Matlab/Simulink. The robotic arms and the haptic paddle are connected to three respective PCs via USB interface utilizing Matlab/Simulink APIs. The two additional PCs are used to implement various software components of the networked architecture. The distributed platform is modeled in PaNeCS (see Figure 14).

The CrustCrawler robot shown in Figure 11 has four degrees of freedom with AX-12 smart servos at each joint [Crustcrawler.com 2009]. Each of these servos has three inputs and five outputs. The inputs are position, velocity, and maximum torque value, and the outputs are actual position, actual velocity, temperature, load, and feedback

Fig. 12. Novint haptic paddle.

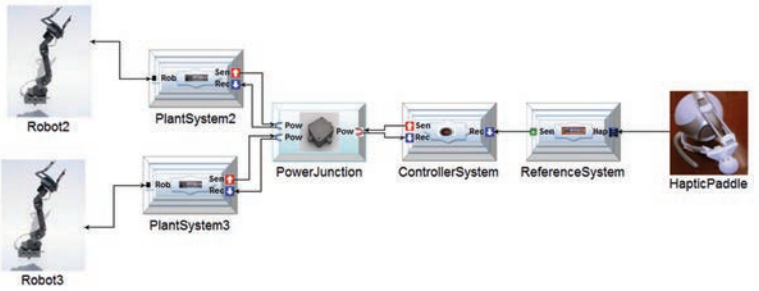


Fig. 13. Control design layer.

voltage. Joints one and four contain one servo while joints two and three each contain two servos for a total of seven servos per robot. In order to design an accurate controller for the CrustCrawler robot, we developed a Simulink model using the "Robotics Toolbox for Matlab" [Corke 2002]. As shown in Figure 11, the robot can be modeled using four points of mass. These points of mass are located at the mid-point of each link in order to create accurate kinematics and dynamics for control design. The point masses are: $m_1 = 0.362$kg, $m_2 = 0.401$kg, $m_3 = 0.059$kg, and $m_4 = 0.177$kg.

The Novint haptic paddle shown in Figure 12 provides the desirable trajectory to be tracked by the robotic arms in a synchronized fashion. The paddle requires the Haptik Library [SIRSLab 2009] that provides an interface between most haptic paddles and various computer languages like C/C++, Java, and MATLAB. The haptic paddle API includes forward kinematics software that transforms the joint positions of the three legs into x-y-z coordinates. When the paddle end effector is moved by the user, a position signal in x-y-z coordinates is sent into a Simulink haptic paddle block.

5.2. Modeling the Networked Multi-Robot System

In order to illustrate the model-based framework, we present the NMRS model developed in PaNeCS. Figure 13 shows the Control Design Aspect that visualizes the control design modeling layer and Figure14 shows the Platform Design Aspect which shows the physical components of the system as well as their interconnections. Our design is based on the assumption that each of the top level models, which include the PlantSystem, ControllerSystem, PowerJunction and the ReferenceSystem is implemented on a separate processor. The mapping of these components to respective processors is performed by the ProcessorAssignmentLayer of the model (that is not shown here).

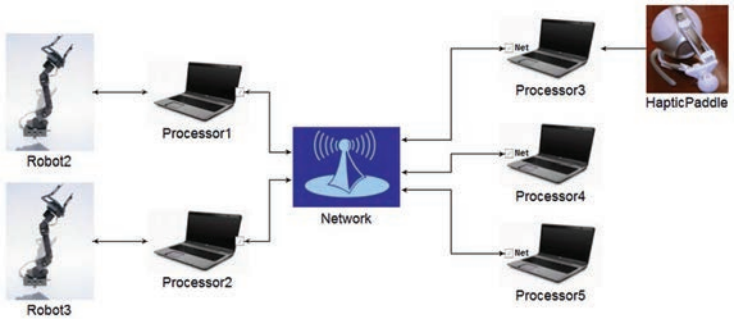


Fig. 14. Platform design layer.

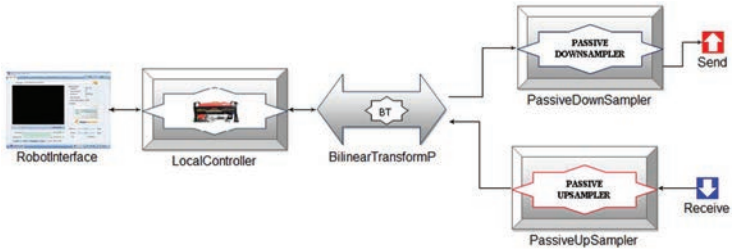


Fig. 15. Plant subsystem.

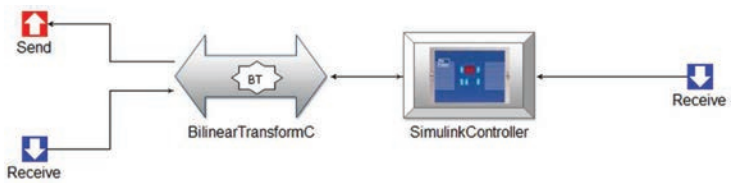


Fig. 16. Controller subsystem.

The details for implementing the PlantSystem which is identical for the two robots are shown in Figure 15. The components for implementing the ControllerSystem are also shown in Figure 16 Finally, Figure 17 shows the components for implementing the ReferenceSystem.

The required parameters for the PlantSystem and the ControllerSystem are listed in Tables I and II, respectively. After configuring and entering the desired parameters for the NMRS model, the code generator is used to generate Simulink models and network scripts. The Simulink models and network scripts are then deployed on the respective PCs based on the deployment model.

6. EXPERIMENTAL EVALUATION

In this section we present experimental results to evaluate the design of the NMRS. The goal in these experiments is for the tip of the end effectors of the robots to follow the trajectory of the human-controlled haptic paddle in a synchronized and stable manner. In order to effectively compare the results of different experiments, a single reference trajectory is generated from the haptic paddle and saved to a data file to be used in all experiments.

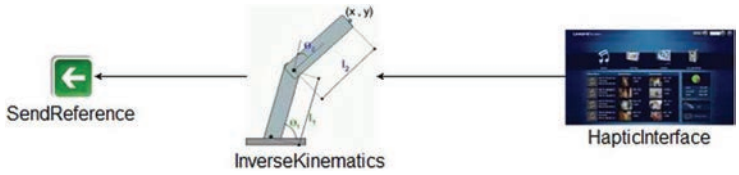


Fig. 17. Reference subsystem.

Table I. PlantSystem Model Parameters

$T_s = 0.04$sec	$q_0 = [0, -\pi/2, \pi/2, 0]$	$K_v = 0.15$
$\tau = 2$	Samplesize: $M = 2$	$b = 0.5$
DimensionOfInputVector: $N = 4$		

Table II. Controller Model Parameters

$T_s = 0.08$ sec	$K = 0.5$	$w = \pi/2$
$w_n = 2\pi/10$	$\zeta = 0.9$	

The PCs communicate asynchronously over the network. In order to plot the trajectory of each robot with respect to a global time reference and quantify the network delays, each PC queries a public NTP time server only once before each experiment. Using the offsets of the local clocks, the time stamps are adjusted and clock skew compensation is performed afterward offline. It should be noted that the timestamps are not used by the system but only for presenting the evaluation results.

We outline three experiments to demonstrate our framework. The first experiment consists of all PCs connected through an Ethernet network and is considered the nominal case. The second experiment demonstrates the robustness of the NMRS to persistent link interruptions. Finally, the third experiment demonstrates the robustness of the NMRS to intermittent link connectivity through an unreliable wireless connection. It also demonstrates the decoupling between layers; change of a wired to a wireless link does not require redesign of the networked system.

After demonstrating the passivity-based framework, we present a comparison study, which compares our approach to an approach that does not use network feedback. The comparison study illustrates the need for network feedback whenever synchronization of the robot arms is desired.

6.1. Experiment 1: Nominal Case

In this experiment all PCs communicate through a 100BASE-TX Ethernet network. Initially, the PCs are not connected to the network and the connection sequence is the power junction, robot 2, robot 3, and controller. Figure 18 shows the x-, y-, and z-coordinates of the robots, along with the reference trajectory, and also the angular position of joint 2 of each robot and the respective reference trajectory. Each robot initially adjusts to its home position. Once the controller is connected to the power junction, the robots begin following the reference trajectory. The robots track the reference trajectory in a synchronized manner. The trajectories of the robots are similar enough to be almost indistinguishable in the plots.

If there is packet loss, the components assume zeros so that no energy is introduced and the system remains passive. Note that other design decisions that preserve passivity can be used. Because of the zeros, the robots attempt to return to their home position, causing the robots to jerk while following the reference trajectory. To compensate for this undesirable behavior, a low-pass filter is added to eliminate this high

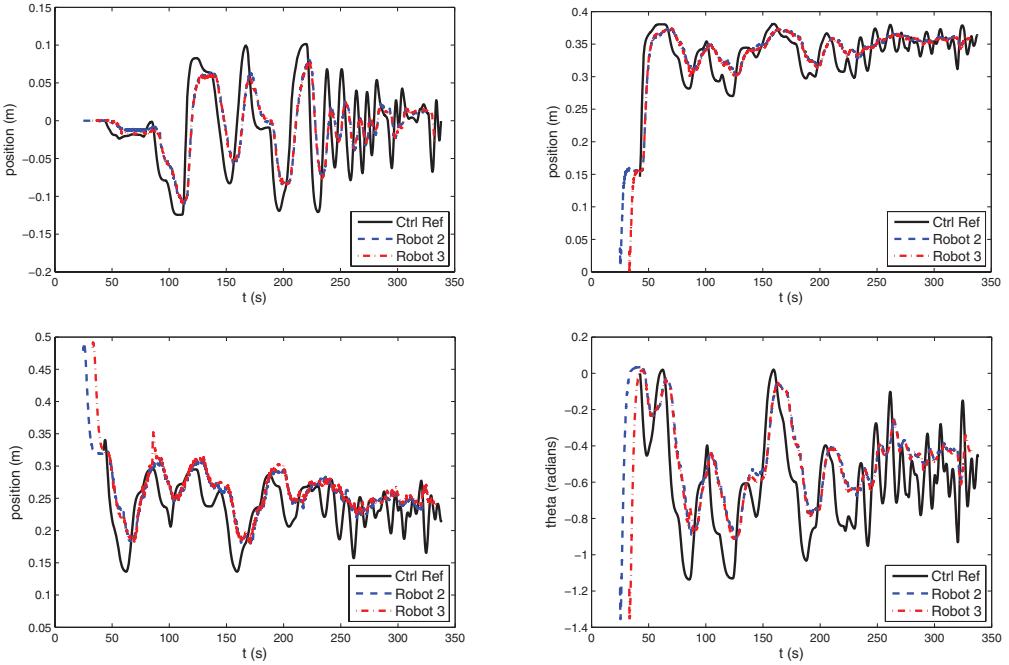


Fig. 18. *x-y-z* coordinates and angle of joint 2 of reference, robot 2, and robot 3.

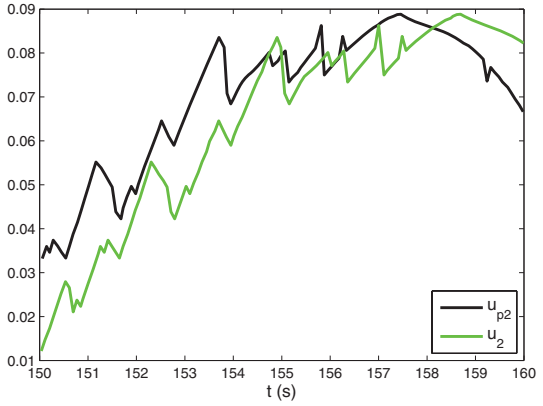


Fig. 19. Time delay between robot 2 and power junction.

frequency motion. The trade off is attenuation and smoothing of the trajectory. Despite this, the robots are well synchronized and the NMRS is stable.

To show the network delay, we plot the first dimension of the wave variable transmitted from robot 2 to the power junction and back (cf. u_{p2} and u_2 in Figure 6) in Figure 19. The delay is time varying and on the order of one second due to the network link and the buffering of data in the Netcat and SSH components.

6.2. Experiment 2: Persistent Link Interruptions

Experiment 2 demonstrates the robustness of the NMRS to persistent link interruptions. To emulate the link interruptions, a boolean variable is implemented in each

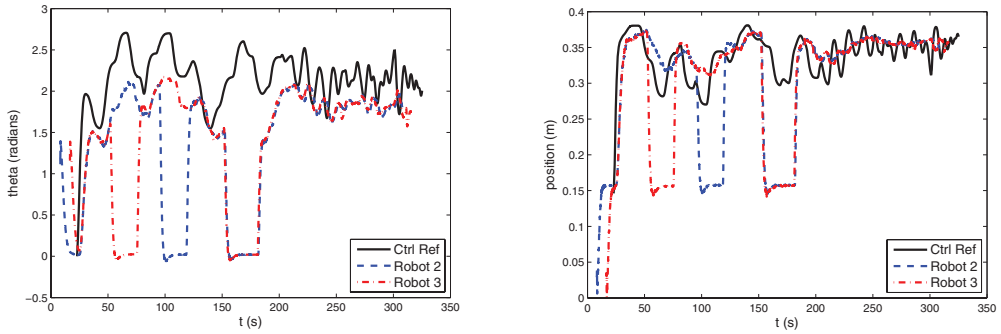


Fig. 20. Persistent link interuptions: Angle of joint 3 and y coordinate of reference, robot 2, and robot 3.

plant and the controller, which controls the data flow to the power junction, and can be toggled online. While the link is interrupted, the power junction simply sends zeros to the respective component. In this way, we avoid shutting down and restarting the network infrastructure that requires considerable time. During the experiment, robot 3 is interrupted, then robot 2, and finally, the controller.

The angular position of joint 3 of both robots and the reference trajectory are shown in Figure 20 to illustrate how each robot returns to its home position while its link to the power junction is interrupted. The figure also shows the y-coordinate of each robot with the reference trajectory. Since the connection sequence is identical to the nominal experiment, the order in which the robots are first seen is identical, and each initially adjusts to its home position. Once the controller is connected to the power junction, the robots begin following the reference trajectory until its link is interrupted. After approximately 52 sec the data flow of robot 3 is interrupted with the power junction. While interrupted, each of its servos returns to its home position. At approximately 75 sec robot 3 is reconnected and resumes following the reference trajectory. Next, around 95 sec, the data flow of robot 2 is interrupted with the power junction. Identically as before, the joints return to the home position while interrupted. Robot 2 is reconnected at approximately 118 sec. Finally, the data flow of the controller is interrupted with the power junction around 151 sec, causing both robots to return to their home position. Again, once the controller is reconnected at approximately 180 sec, the robots resume following the reference trajectory. Since the link interruptions are implemented in a very controlled manner, no additional network delay is caused by the link interruptions and the delays are similar to the nominal case.

6.3. Experiment 3: Intermittent Wireless Connection

In Experiment 3, the PC running the controller is connected to the Ethernet network through an 802.11b wireless connection. The experiment demonstrates the performance of the NMRS with an intermittent communication channel. Figure 21 shows the angular position of joint 3 of the robots along with the reference trajectory. Again, each robot initially adjusts to its home position. Due to the increased network delay, the plants do not begin tracking the reference trajectory until after approximately 29 sec into the experiment.

Because of the unreliable wireless connection between the controller and power junction much data is dropped, causing more high frequency components in the trajectories of the robots. Even with the low-pass filter, the trajectories are clearly not as smooth and accurate as in previous experiments. Observe, for example, the peak between 35 and 40 sec in Figure 20 and compare it to the behavior in Figure 21. Obviously data is being dropped from the controller during this time since in both robots there is more

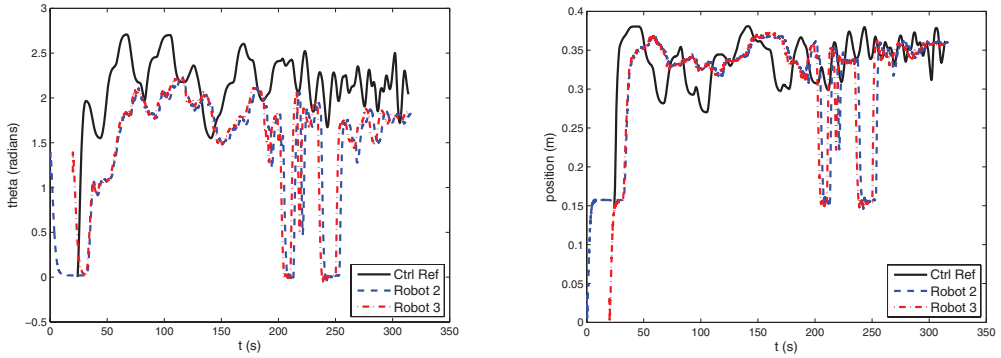


Fig. 21. Intermittent connection: Angle of joint 3 and y coordinate of reference, robot 2, and robot 3.

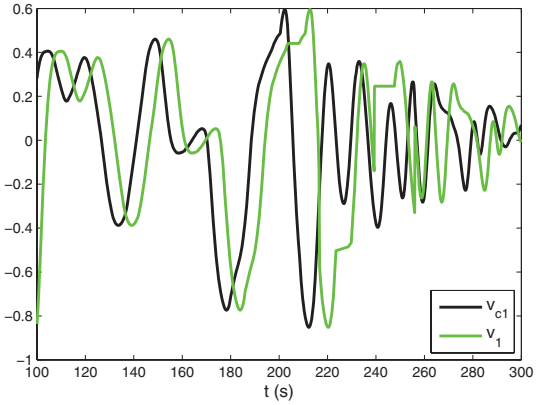


Fig. 22. Time delay between controller and power junction.

attenuation and no well defined peak as in Figure 20. Between 204 and 260 sec, the connection gets significantly worse. Despite the intermittent connectivity, the NMRS remains stable and still manages to track the reference reasonably well while the connection is established. Again, the y-coordinate of each plant, along with the reference trajectory, is shown in Figure 21.

To illustrate the larger and more uncertain network delay for this experiment, the first dimension of the wave variable sent to the power junction from the controller and back (cf. v_1 and v_{c1} in Figure 6) is shown in Figure 22 between 100 and 300 sec. Before the severe connectivity problems, the delay is approximately 5 sec. During the loss of connectivity seen around 205 seconds and 240 seconds, the data loss can be seen in Figure 22 by the horizontal lines in the wave variable received at the power junction (v_1). The delay grows up to 17 sec before the corrective mechanism of data dropping reduces it back to about 8 sec toward the end of the experiment.

6.4. Comparison Study

In this section, we compare the passivity-based approach to an approach where the robots are not coupled through the network but they follow the same reference trajectory received independently over the network. We call this approach "open-loop", however, it should be noted that the robots locally employ the same feedback controllers as in the passivity based approach described in Section 3.

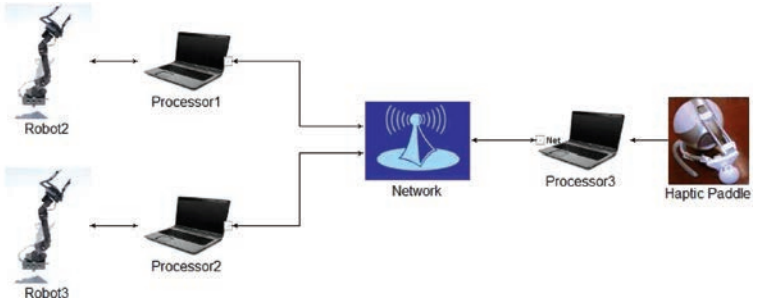


Fig. 23. Platform design layer for open-loop approach.

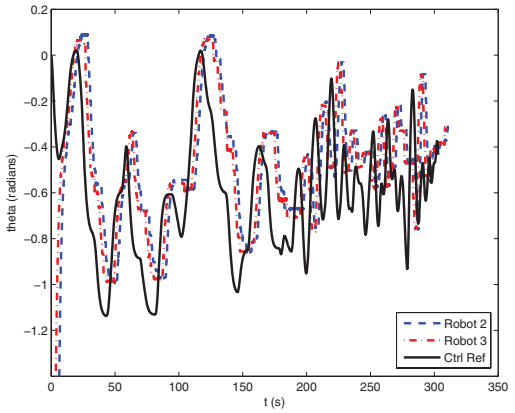
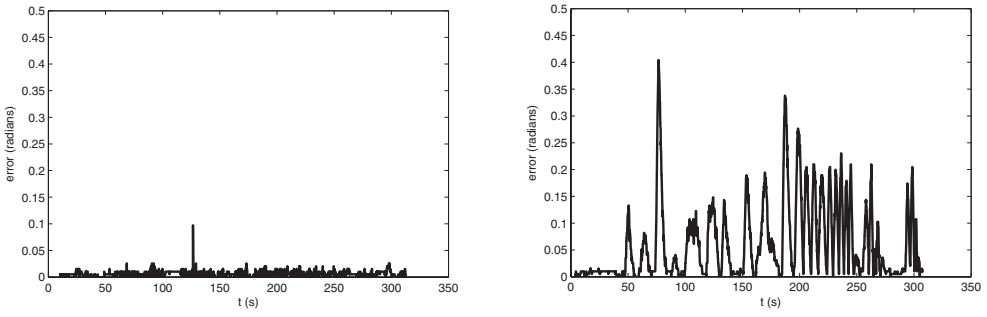


Fig. 24. Angle of joint 2 for the open-loop approach.

Our objective is to demonstrate how the passivity-based approach facilitates better synchronization of the robots than the open-loop approach. Without feedback, any disturbances or model uncertainties can easily disrupt the robots from following the reference trajectory in a synchronized manner. Coupling the behavior of the robots through the network can provide advantages in the presence of uncertainty. The passivity-based approach ensures that stability is preserved when this coupling is introduced even in the presence of network delays and packet loss.

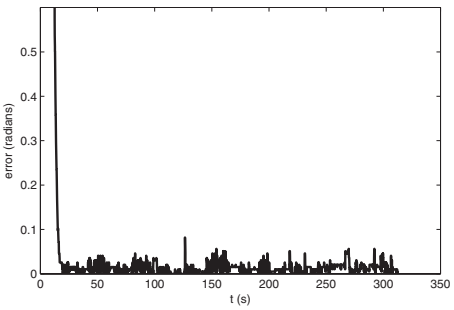
Specifically, we consider two scenarios: (1) network delay that affects the communication of the reference trajectory to the robots and (2) uncertainty in the physical plant that affects the behavior of the robots. The first scenario is the same as the nominal case of Section 6.1 which assumes no precise time synchronization between the processors. In the second scenario, a small weight is strapped to the end effector of Robot 2. For the passivity-based approach, the results are obtained using the same experimental set up described in Section 5. For the open-loop approach, we use the architecture shown in Figure 23. A processor connected to each robotic arm receives the reference trajectory sent by a third processor over the network, and implements a local controller whose objective is to follow the reference trajectory independently of the other robot. In order to use exactly the same reference trajectory, the input from the haptic paddle is saved to a file which is used for all the experiments.

Scenario 1: Network Delays. As in the experiment in Section 6.1, all PCs communicate through a 100BASE-TX Ethernet network. In order to illustrate the need for network feedback to satisfy the synchronization objective, we introduce different network delays

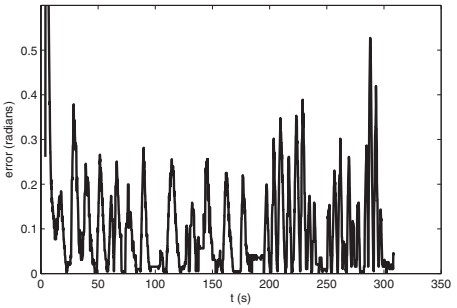


(a) Joint 1 for passivity-based approach

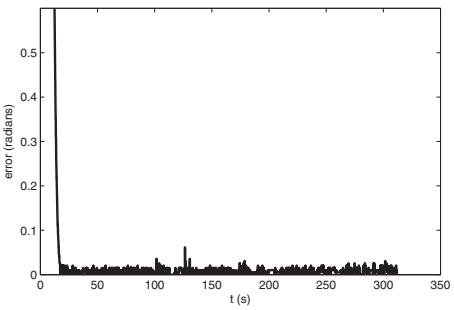
(b) Joint 1 for open-loop approach



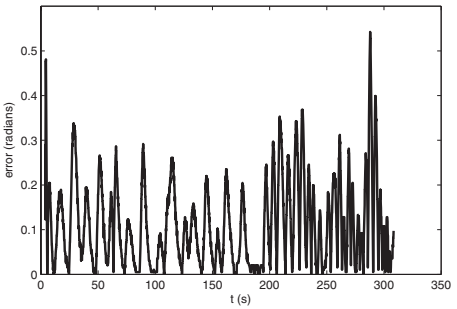
(c) Joint 2 for passivity-based approach



(d) Joint 2 for open-loop approach



(e) Joint 3 for passivity-based approach

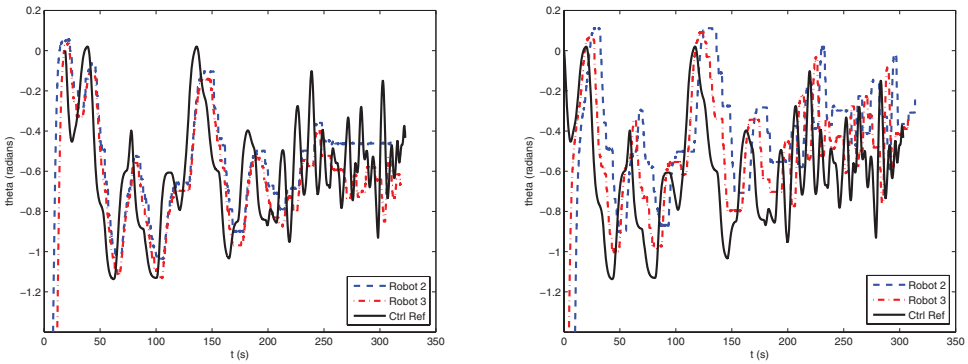


(f) Joint 3 for open-loop approach

Fig. 25. Pairwise output errors for nominal experiment.

between the processor that sends the reference trajectory and the processors linked to the robots. A simple way to do this is inherent in the experimental setup. Namely, the startup sequence naturally imposes different delays because once the reference trajectory begins broadcasting, each robot's computer begins buffering the input data. Once the robot is started, it begins processing the packets. Therefore, any delay between starting the robots results in the desired delay. The connection sequence for the passivity-based approach is the same as the first nominal experiment, namely, the power junction, robot 2, robot 3, and finally the network controller. For the open-loop approach, the connection sequence begins with the processor sending the reference trajectory, then robot 3, and finally robot 2.

Fig. 26. CrustCrawler arm with weight (remote control).



(a) Passivity-based approach (b) Open-loop approach

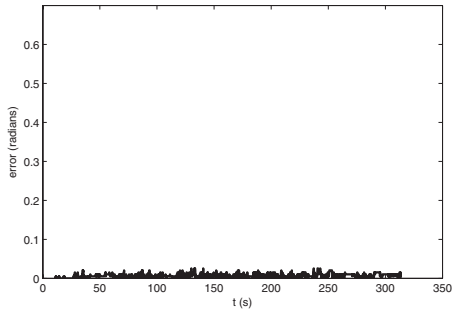
Fig. 27. Joint 2 for robots with model uncertainty.

Figure 24 shows the angle trajectories for Joint 2 of the two robots for the open loop approach and should be compared with Figure 18. Initially, there is a delay between the motion of the robots which is approximately 2 sec and the two robots are not synchronized. Because of the coupling in the passivity-based approach, the robots quickly synchronize with each other. In the open loop approach, however, the robots continue to follow the reference trajectory without synchronizing. It would be straightforward to synchronize the local controllers so that the two robots start exactly at the same time and periodically synchronize to account for network delays. However, this requires coupling between the local controllers and can address only uncertainty in the network delays.

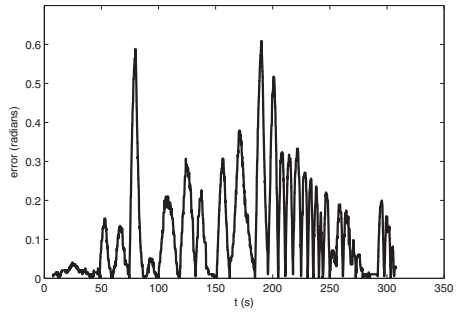
In order to carefully compare the two approaches, we define the absolute error between the trajectory of each joint of the robots by

$$e_{p23_l} = |\theta_{p2_l} - \theta_{p3_l}|,$$

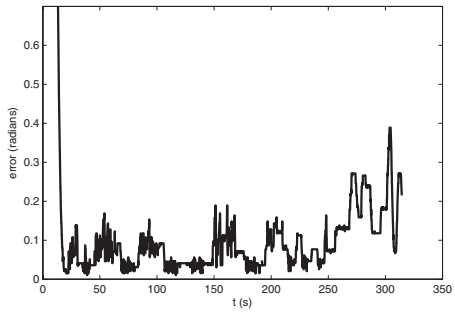
where $l \in \{1, 2, 3\}$ denotes the l-th joint angle. Figure 25 shows the absolute error for the 3 joints. Clearly, the passivity-based approach enables the robots to synchronize, whereas the open-loop approach is unable to compensate for the network delay. Without coupling, the robots cannot adjust to account for different delays between the processors that sends the reference trajectory and the local controllers.



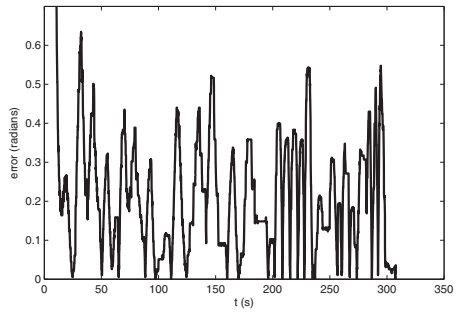
(a) Joint 1 for passivity-based approach



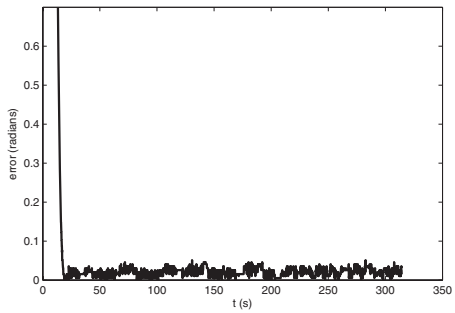
(b) Joint 1 for open-loop approach



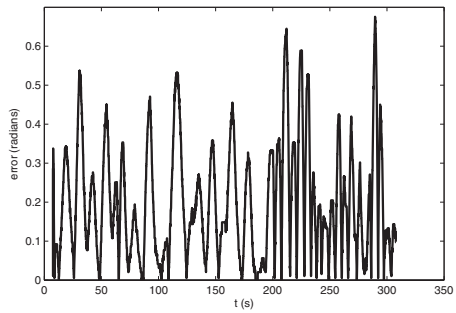
(c) Joint 2 for passivity-based approach



(d) Joint 2 for open-loop approach



(e) Joint 3 for passivity-based approach



(f) Joint 3 for open-loop approach

Fig. 28. Pairwise output errors for model uncertainty experiment.

Scenario 2: Plant Uncertainty. To illustrate how uncertainty in the physical plant can affect synchronization of the robots, a small weight (remote control) is strapped to the end effector of robot 2, as shown in Figure 26. Figure 27 shows the angle of Joint 2 of the two robots for the passivity-based and open loop approach. The absolute error between the trajectories of each joint of the robots is shown in Figure 28. Due to the position of the weight, joint 2 bears the majority of the load, and hence performs the worst with respect to tracking the reference trajectory. There is some degradation of performance in the passivity-based approach, but the robots still follow the trajectory in a synchronized way. In the open loop approach, the uncertainty due to the load results in additional synchronization error.

The conclusion of this comparison study is that the passivity-based approach provides advantages when synchronization between the two robots is important. If there is no uncertainty, synchronization can be achieved without closing the loop over the network. However, feedback is required to deal with uncertainty both locally at each robot and globally for coordinating the robots.

The passivity-based approach offers a systematic way for design of the networked multi-robot system based on decoupling of the control design from the network implementation. This decoupling allows us to build a networked control systems using off-the-self components and ensure stability in the presence of network delays and packet loss. Although tailored solutions may improve system performance, typically they are complex, expensive, and rigid. Integration of existing components by decoupling of the design concerns is of significance in CPS.

7. CONCLUSIONS

The article presents a model-based framework for design of NCS using passivity that improves decoupling between the controller design and implementation design layers. Our results show that passivity decouples the control design of networked systems from network uncertainties. We have demonstrated the approach using an experimental networked multi-robot system. Our results show that two robots can follow a reference signal in a synchronized manner in the presence of time delays and data dropouts. A major concern with passivity-based approaches is that although they can tolerate network uncertainties, they may lead to a conservative design that limits the responsiveness to fast reference signals because of the constraints imposed on the controllers. In our approach, high frequency signals are filtered and tuning of the filters and the control gains is required for improving performance. On the other hand, the proposed model-based design with the prototype domain-specific modeling language and automated code generation tools on top of passivity facilitate effective system configuration, deployment, and testing.

REFERENCES

Anderson, R. and Spong, M. 1992. Asymptotic stability for force reflecting teleoperators with time delay. *Int. J. Robotics Res. 11,* 2, 135–149.

Antsaklis, P. and Baillieul, J., Eds. 2007. Technology of Networked Control Systems (Special Issue). *Proc. IEEE, 95*, 1.

Arcak, M. 2007. Passivity as a design tool for group coordination. *IEEE Trans. Auto. Control 52,* 8, 1380–1390.

AS-2 Embedded Computing Systems Committee. 2004. Architecture analysis and design language (AADL). Tech. rep. AS5506, Society of Automotive Engineers.

Bai, H., Arcak, M., and Wen, J. T. 2008. Rigid body attitude coordination without inertial frame information. *Automatica 44,* 12, 3170–3175.

Baillieul, J. and Antsaklis, P. 2007. Control and communication challenges in networked real-time systems. *Proc. IEEE 95,* 1, 9–28.

Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Paserone, C., and Sangiovanni-Vincentelli, A. L. 2003. Metropolis: an integrated electronic system design environment. *IEEE Computer 36,* 4, 45–52.

Bao, J. and Lee, P. L. 2007. *Process Control : The Passive Systems Approach*. Springer-Verlag.

Bhave, A. and Krogh, B. 2008. Performance bounds on state-feedback controllers with network delay. In *Proceedings of the 47th IEEE Conference on Decision and Control*. 4608–4613.

Brockett, R. and Liberzon, D. 2000. Quantized feedback stabilization of linear systems. *IEEE Trans. Auto. Control 45,* 7, 1279–1289.

Chopra, N., Berestesky, P., and Spong, M. 2008. Bilateral teleoperation over unreliable communication networks. *IEEE Trans. Control Syst. Technol. 16,* 2, 304–313.

Chopra, N. and Spong, M. 2006. Passivity-based control of multi-agent systems. In *Advances in Robot Control: From Everyday Physics to Human-Like Movements*, 107–134.

Corke, P. I. 2002. Robotic toolbox for Matlab, Release 7.1. Tech. rep., CSIRO.

CRAIG, J. J. 1989. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley.

CRUSTCRAWLER.COM. 2009. Dynamixel AX-12 Manual. http://www.crustcrawler.com/products/bioloid/docs/AX-12.pdf.

EYISI, E., PORTER, J., HALL, J., KOTTENSTETTE, N., KOUTSOUKOS, X., AND SZTIPANOVITS, J. 2009. PaNeCS: A modeling language for passivity-based design of networked control systems. In *Proceedings of the 2nd International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES-MB '09)*. 27–41.

FETTWEIS, A. 1986. Wave digital filters: theory and practice. *Proc. IEEE 74,* 2, 270–327.

GAO, H., CHEN, T., AND CHAI, T. 2008. Passivity and passification for networked control systems. *SIAM J. Control Optimiz. 46,* 4, 1299–1322.

HADDAD, W. M. AND CHELLABOINA, V. S. 2008. *Nonlinear Dynamical Systems and Control: A Lyapunov-Based Approach*. Princeton University Press, Princeton, NJ.

HESPANHA, J., NAGHSHTABRIZI, P., AND XU, Y. 2007. A survey of recent results in networked control systems. *Proc. IEEE 95,* 1, 138–162.

HIRCHE, S. AND BUSS, M. 2007. Transparent data reduction in networked telepresence and teleaction systems. part ii: Time-delayed communication. *Presence: Teleoper. Virtual Environ. 16,* 5, 532–542.

HIRCHE, S., MATIAKIS, T., AND BUSS, M. 2009. A distributed controller approach for delay-independent stability of networked control systems. *Automatica 45,* 8, 1828–1836.

HUDAK J. AND FEILER P. 2007. Developing AADL models for control systems: A practitioner's guide. Tech. rep. CMU/SEI-2007-TR-014, CMU SEI.

IHLE, I.-A. F., ARCAK, M., AND FOSSEN, T. I. 2007. Passivity-based designs for synchronized path-following. *Automatica 43,* 9, 1508–1518.

KARSAI, G., SZTIPANOVITS, J., LEDECZI, A., AND BAPTY, T. 2003. Model-integrated development of embedded software. *Proc. IEEE 91,* 1, 145–164.

KOTTENSTETTE, N. AND ANTSAKLIS, P. 2010. Relationships between positive real, passive dissipative, & positive systems. In *Proceedings of the American Control Conference*. 409–416.

KOTTENSTETTE, N., HALL, J., KOUTSOUKOS, X., ANTSAKLIS, P., AND SZTIPANOVITS, J. 2011. Digital control of multiple discrete passive plants over networks. *Int. J. Syst., Comm. Control 3,* 2, 194–228.

KOTTENSTETTE, N., KOUTSOUKOS, X., HALL, J., SZTIPANOVITS, J., AND ANTSAKLIS, P. 2008. Passivity-based design of wireless networked control systems for robustness to time-varying delays. In *Proceedings of the Real-Time Systems Symposium (RTSS 08)*. 15–24.

KOTTENSTETTE, N. AND PORTER, J. 2009. Digital passive attitude and altitude control schemes for quadrotor aircraft. In *Proceedings of the 7th International Conference on Control and Automation (ICCA'09)*.

LEBLANC, H., EYISI, E., KOTTENSTETTE, N., KOUTSOUKOS, X., AND SZTIPANOVITS, J. 2010. A passivity-based approach to deployment in multi-agent networks. In *Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics (ICINCO '10)*. 53–62.

LEDECZI, A., BAKAY, A., MAROTI, M., VOLGYESI, P., NORDSTROM, G., SPRINKLE, J., AND KARSAI, G. 2001a. Composing domain-specific design environments. *IEEE Computer*, 44–51.

LEDECZI, A., MAROTI, M., BAKAY, A., KARSAI, G., GARRETT, J., IV, C. T., NORDSTROM, G., SPRINKLE, J., AND VOLGYESI, P. 2001b. The generic modeling environment. In *Proceedings of the Workshop on Intelligent Signal Processing*.

LI, P. Y. AND HOROWITZ, R. 1997. Control of smart machines, Part 1: Problem formulation and non-adaptive control. *IEEE/ASME Trans. Mechatron. 2,* 4, 237–247.

LIAN, F.-L., MOYNE, J., AND TILBURY, D. 2002. Network design consideration for distributed control systems. *IEEE Trans. Control Syst. Technol. 10,* 2, 297–307.

MONTESTRUQUE, L. A. AND ANTSAKLIS, P. 2004. Stability of model-based networked control systems with time-varying transmission times. *IEEE Trans. Aut. Control 49,* 9, 1562–1572.

NEŠIC, D. AND LIBERZON, D. 2009. A unified framework for design and analysis of networked and quantized control systems. *IEEE Trans. Auto. Control 54,* 4, 732–747.

NIEMEYER, G. AND SLOTINE, J.-J. E. 2004. Telemanipulation with time delays. *Int. J. Robotics Res. 23,* 9, 873 – 890.

OPPENHEIM, A., WILLSKY, A., AND NAWAB, S. 1997. *Signals and Systems*. Prentice hall Upper Saddle River, NJ.

ORTEGA, R. AND SPONG, M. 1988. Adaptive motion control of rigid robots: A tutorial. In *Proceedings of the 27th IEEE Conference on Decision and Control*. 1575–84.

PORTER, J., KARSAI, G., VOLGYESI, P., NINE, H., HUMKE, P., HEMINGWAY, G., THIBODEAUX, R., AND SZTIPANOVITS, J. 2008. Towards model-based integration of tools and techniques for embedded control system design, verification, and implementation. In *Proceedings of the Workshops and Symposia at MoDELS*. Lecture Notes in Computer Science, vol. 5421, Springer.

SEILER, P. AND SENGUPTA, R. 2005. An H-infinity approach to networked control. *IEEE Trans. Auto. Control 50,* 3, 356–364.

SIRSLAB. 2009. Haptik library overview. http://sirslab.dii.unisi.it/haptiklibrary/overview.htm.

SKAF, J. AND BOYD, S. 2007. Analysis and synthesis of state-feedback controllers with timing jitter. *IEEE Trans. Auto. Control 54,* 3, 652–657.

STRAMIGIOLI, S., SECCHI, C., VAN DER SCHAFT, A. J., AND FANTUZZI, C. 2005. Sampled data systems passivity and discrete port-hamiltonian systems. *IEEE Trans. Robotics 21,* 4, 574–587.

VAN DER SCHAFT, A. 1999. *L2-Gain and Passivity in Nonlinear Control*. Springer-Verlag, Berlin, Germany.

WALSH, G. C., YE, H., AND BUSHNELL, L. G. 2002. Stability analysis of networked control systems. *IEEE Trans. Control Sys. Technol. 10,* 3, 438–446.

Received December 2009; revised August 2010, February 2011; accepted May 2011