Passivity-Based Self-Triggered Control: A Case Study On The Trajectory Tracking Control Of A Robotic Manipulator Over Wireless Network.

Emeka Eyisi
emeka.p.eyisi@vanderbilt.edu

Xenofon Koutsoukos
xenofon.koutsoukos@vanderbilt.edu

Institute for Software Integrated Systems
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN, USA

ABSTRACT

In this paper we present a case study on the trajectory tracking control of a robotic manipulator over a wireless network. We use passivity and self-triggered control to achieve two important desirable properties of Cyber Physical Systems (CPS), efficient use of resources and stability in the presence of network uncertainties. By using a passivity based approach, our design ensures the stability of the overall system even in the presence of network uncertainties while the self-triggered strategy ensures efficient use of network resources. We present preliminary simulation results to demonstrate our approach for the case study.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics—*Manipulators*

General Terms

Design, Experimentation, Theory

Keywords

Passivity, Self-Triggered Control, Networked Control System

1. INTRODUCTION

Cyber Physical Systems (CPS) are characterized by the integration of physical systems through computing and networking. CPS are typically composed of multiple feedback loops, some may be local while others may be connected over a network. Feedback loops are increasingly deployed over wireless communication networks, as they provide great convenience in terms of deployment and mobility support. On the other hand, uncertainties in wireless networks such as time-varying delays and data dropouts present significant challenges and require the development of novel analysis and synthesis methods in order to achieve overall desirable system properties.

In this paper, we introduce a case study on the trajectory tracking control of a robotic manipulator over a wireless network. We focus on the challenges of the uncertainties and limitations of the network. The robotic manipulator receives information from a collocated local controller about the desired trajectory it needs to track. It has no additional information about it's environment. A separate agent called networked controller, has additional information about the robotic arm's environment. In the event that an obstacle is observed in the path of the robotic arm, the networked controller sends control signals to the local controller to adjust the robotic arm's trajectory. In our approach, we show that based on the control command, the robotic arm still maintains the pattern of the desired trajectory but with an offset correponding to a function of the modelled obstacle as observed by the networked controller. We also integrate a self-triggered strategy based on [4] in order to reduce the amount of network resources utilized.

This case study presents a CPS application because it is composed of multiple feedback loops, both local and over the wireless network, that involves the interaction of the physical system through computing and network with the ultimate goal of achieving overall desirable system behavior under various network uncertainties.

Our contribution are the following: We introduce a passivity-based self-triggered network control architecure for the trajectory control of a robotic manipulator over a wireless network. We provide simulations using Simulink/TrueTime to illustrate our approach.

2. PASSIVITY-BASED SELF-TRIGGERED CONTROL ARCHITECTURE

The proposed passivity-based self-triggered control (PBSTC) architecture used in the trajectory tracking control of a robotic manipulator is shown in Fig. 1.

2.1 Composite System

The enclosed box denoted composite system in Fig. 1 represents the model resulting from a transformation of the robotic manipulator dynamics into a desirable passive input-output mapping. Fig. 2 shows the components of composite system. The resulting passive input-output mapping is used to derive an overall passivity based self-triggered networked control system to enable tracking.

In the absence of friction and disturbances, the Euler-Lagrange equations of motion for an n-degree-of-freedom robotic manipula-

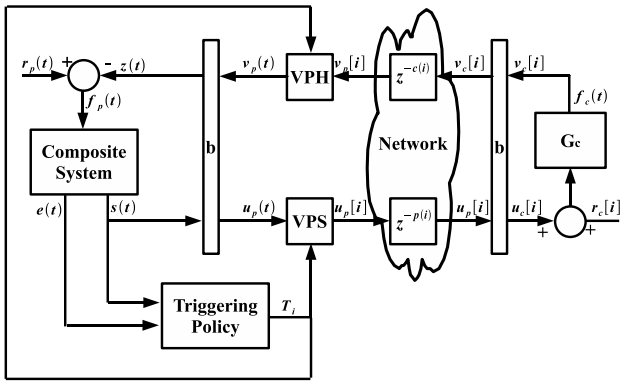


Figure 1: PBSTC Networked Control Architecture.

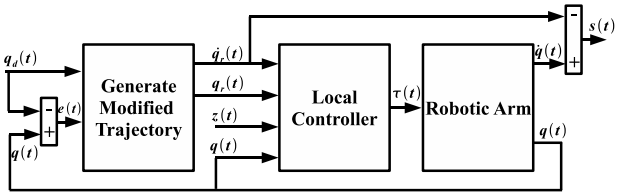


Figure 2: Composite System Model.

tor can be generally described by the following dynamic model [5]:

$$M(q)\ddot{q} + C(q,\dot{q})(\dot{q}) + g(q) = \tau; \tag{1}$$

where $q(t) \in \mathbb{R}^n$ is the vector of joint angles, $\tau(t) \in \mathbb{R}^n$ is the input torque vector, $M(q) \in \mathbb{R}^{n\times n}$ is the inertia matrix, $C(q,\dot{q}) \in \mathbb{R}^{n\times n}$ is the matrix of centrifugal and coriolis effects, and $g(q) \in \mathbb{R}^n$ is the gravity vector.

It is well-known that a passive proportional derivative (PD) controller can be used in set-point tracking control of a robotic manipulator in a passivity-based framework but it is not an appropriate controller for the case of trajectory tracking [10] [3].

In order to achieve the goal of trajectory tracking in a passivity-based framework, we employ the approach proposed in [9]. We introduce a sliding variable based on sliding-mode technique. The sliding variable is described in terms of the tracking error dynamics such that when the sliding variable converges to zero the tracking error dynamics converges to zero as well. This new sliding variable introduces an integral action via the reference trajectory. The composite system is the transformation that results in a passive input-output mapping from a virtual input $z(t)$ to the sliding variable output $s(t)$.

Let q_d be defined as the desired trajectory that is twice differentiable. The tracking error is given by $e = q - q_d$.

We choose the local controller as:

$$\tau = M(q)\ddot{q_r} + C(q,\dot{q})\dot{q_r} + g(q) + z; \tag{2}$$

where z is the control signal from the networked controller and q_r is the modified reference trajectory defined by

$$q_r = q_d - \Lambda\int_0^t e; \quad \dot{q_r} = \dot{q}_d - \Lambda e;$$

where Λ is a positive diagonal matrix. Then by combining (1) and (2), the closed-loop dynamics can be derived. Based on the definition of q_r, the sliding variable, s appears in the closed-loop dynamics as

$$M(q)\dot{s} + C(q,\dot{q})s = z; \tag{3}$$

where s is defined as

$$s = \dot{q} - \dot{q_r} = \dot{e} + \Lambda e. \tag{4}$$

2.2 Self-triggered Control Policy

The triggering policy box in Fig. 1 outputs sampling intervals, T_i, using a self-triggered policy inspired by the the work in [4]. The algorithm is adapted and presented in the framework of passive systems and for the purpose of achieving tracking performance rather than stability. This is possible because our passivity-based framework inherently guarantees stability. The system model used for our self-triggered policy is derived from the relationship between tracking error, e, and the sliding variable, s. This model can be represented as

$$\dot{e}(t) = -\Lambda e(t) + s(t); \quad y = Ie; \tag{5}$$

where I is the identity matrix.

$$s(t) = s(t_i), \quad t \in [t_i, t_{i+1}) \tag{6}$$

where $[t_i]_{i\in\mathbb{N}}$ is a sequence of sampling times. In this system model, the output y is the tracking error, while the sliding variable, s, is the input.

A self-triggered implementation for the system given by (5) and (6) is given by a map Γ, determining the sampling instants, t_{i+1}, as a function of the tracking error output, y at the time t_i, i.e., $t_{i+1} = t_i + \Gamma(y(t_i))$. If we denote by T_i, the sampling interval $T_i = t_{i+1} - t_i$ we have $T_i = \Gamma(y(t_i))$. Similar to the approach in [4], design of a self-triggered policy involves a sequence of steps. First, an output function to describe the evolution of the system's tracking error needs to be determined. The output function for our self-triggering policy is the storage function of the system defined by (5).

$$V(y) = y^{\mathsf{T}}Py \tag{7}$$

where P is a positive definte matrix. The output function defined in (7) has an estimated decay rate, λ_0 [8]. Next, we define a performance specification in terms of the tracking error output. Our performance function is an exponentially decaying function of the output function with an initial value as the current sampled error. The perfomance specification can be formally defined as:

$$D(t) = V(y(t_i))exp^{-\lambda(t-t_i)} \tag{8}$$

In order to guarantee that the performance function bounds the output function, the decay rate of the performance specification is choosen as $\lambda < \lambda_0$.

Using the output function and performance specification we determine a continuous time event-triggering function. From (7) and (8), the event-triggering condition is given as

$$h_c(t_i, y(t), y(t_i)) := V(y(t)) - V(y(t_i))exp^{(-\lambda(t-t_i))} \leq 0; \tag{9}$$

for some $0 < \lambda < \lambda_0$. From the event-triggering function we can then determine a self-triggered policy. In order to check the event triggering condition defined in (9) $\forall t \in \mathbb{R}^+$, we consider a discrete-time implementation based on a discrete step size, $\Delta \in \mathbb{R}^+$ since no digital implementation can be used to perform this check. With, T_i defined as the sampling interval, let T_{min} and T_{max} be defined as the minimum and maximum sampling intervals respectively. Also $\forall i \in N$, $N_{min} := \lfloor T_{min}/\Delta \rfloor$, $N_{max} := \lfloor T_{max}/\Delta \rfloor$. The discrete-time implementation can be defined as follows:

$$h_d(y(t_i), n) := h_c(t_i, y(t), y(t_i)) \leq 0 \quad \forall n \in [0, \lfloor T_i/\Delta \rfloor) \tag{10}$$

From this discrete-time implementation, the map $\Gamma_d : \mathbb{R}^n \mapsto \mathbb{R}^+$, for computing the next sampling interval or time for the tracking error system model given in (5) is given by:

$$\Gamma_d(y(t_i)) := \max\{T_{\min}, n_i\Delta\} \tag{11}$$

$$n_i := \max_{n\in\mathbb{N}}\{n \leq N_{\max} | h_d(y(t_i), c) \leq 0, c = 0, ..., n\} \tag{12}$$

$T_{\min}$ and $T_{\max}$ explicitly enforce lower and upper bounds, respectively for the sampling intervals. The upper bound enforces robustness of the implementation and limits computational complexity. The discrete time step, Δ, can be choosen based on desired accuracy and computational complexity.

2.3 Wave Variables

In [1], the authors showed that by transmitting wave variables over the network, the communication channel maintains passivity even in the presence of time-varying delays and packet loss. We take advantage of this notion in order to ensure that the passivity of the transmitted information across the network is preserved.

On the composite system's side of the network, the control input signal, $z(t)$ and sliding variable, $s(t)$ are transformed into the wave domain through the scattering transformation which produces the continuous power waves $u_p(t)$ and $v_p(t) \in \mathbb{R}^m$ and is represented by the block b in Fig. 1. On the networked controller's side of the network, the scattering transformation produces digital power waves $u_c[i]$ and $v_c[i] \in \mathbb{R}^m$. Due to limited space, the details of the scattering transformation is omitted, see [2] for more details.

2.4 Variable Passive Sampler and Variable Passive Hold

In Fig. 1, the variable passive sampler (VPS) and variable passive hold perform the task of sample and hold devices respectively but are implemented in such a way to preserve passivity. These two components execute their tasks based on the adaptive sampling interval provided by the self-triggering policy described above instead of the traditional fixed sampling interval as the equivalent components described in [2]. The VPS and VPH are designed such that the following two inequalities are satisfied:

$$\int_0^{t_n} u_p(t)^\mathsf{T} u_p(t) dt \geq \sum_{i=0}^{n-1} T_i u_p[i]^\mathsf{T} u_p[i] \tag{13}$$

$$\sum_{i=0}^{n-1} T_i v_p[i]^\mathsf{T} v_p[i] \geq \int_0^{t_n} v_p(t)^\mathsf{T} v_p(t) dt \tag{14}$$

Denote the jth element of the column vectors $u_p(t)$ and $u_p[i]$ as $u_{pj}(t)$ and $u_{pj}[i]$, respectively, $j = 1, ..., m$ and let $u_{pj}(t) = 0$ if t$<$ 0. An implementation of the VPS that ensures condition in (13) is given by

$$u_j[i] = \frac{1}{\sqrt{T_{i-1}T_i}} \int_{t_{i-1}}^{t_i} u_j(t) dt, \ \forall\, j \ \in \ \{1, \ldots, m\}. \tag{15}$$

In a similar manner, an implementation of the VPH that satisfies condition in (14) is given by

$$v_j(t) = \sqrt{\frac{T_{i-1}}{T_i}} v_j[i-1], \ t \in [t_i, t_i + T_i). \tag{16}$$

2.5 Networked Controller

The networked controller is denoted as G_c in Fig. 1. The networked controller is a passive event-based proportional controller with gain, K_c. The networked controller is located remotely away from the robot and is assumed to have more information about the robot's environment than the robot itself. There are two types of event that will trigger the execution of the controller, one such event is the arrival of a control update request from the composite system for error correction and the other event is the presence of an obstacle in the robot's path, in which case the controller sends an update with a control command to modify the system's path to ensure safety.

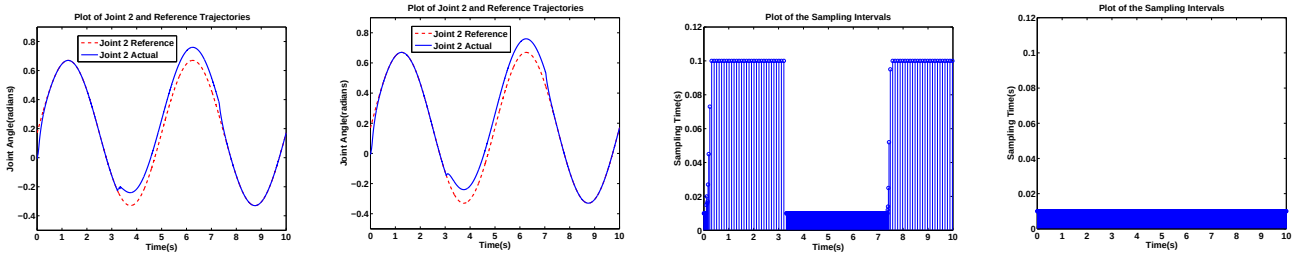
3. SIMULATION

The experimental setup involves a networked passive composite system and an event-based networked controller communicating over a wireless network as shown in Fig. 1. The dynamics of the composite system as well as the networked controller are implemented using Simulink models while TrueTime is used to simulate the wireless network dynamics and the communication between the two subsystems. The network protocol used is 802.11b, with a speed/bandwidth of 11Mbps. The CrustCrawler robot has four degrees of freedom with AX-12 smart servos at each joint. The robot can be modeled using four points of mass. These points of mass are located at the mid-point of each joint. 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. We derived the dynamic model of the CrustCrawler robotic arm for our simulation. Using this model, we implemented the transformation discussed in Section 2. The design parameters for the self-triggered policy are $T_{\min} = 0.01$,$T_{\max} = 0.1$ and $\Delta = 0.001$. The goal of these experiments is for the robotic arm to track a specified trajectory while using fewer network resources and also maintaining stability in the presence of network uncertainties.

Experiment 1: PBSTC Approach vs Traditional Approach. In this experiment we consider, nominal network conditions with no additional delays and packet losses. We model the presence of an object as a step reference input to the networked controller. An obstacle modelled as a step input of magnitude 0.9 is introduced in the robot's environment between the time interval from 3 seconds to 7 seconds. For our PBSTC approach, Fig. 3a shows the resulting trajectory while Fig. 3c shows the sampling times, or controller activation intervals.

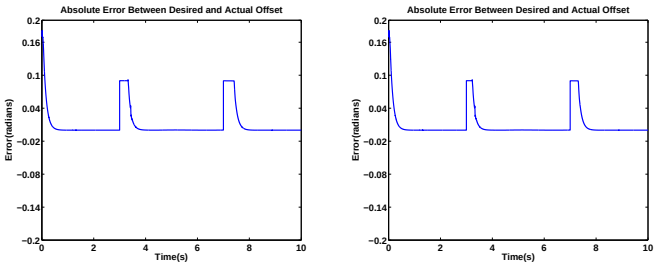
We compare the plots from our nominal scenario to the traditional alternative whereby a control update is requested at a fixed sampling time or interval of $T_{\min}$. For the traditional approach, Fig. 3b shows the resulting trajectory while Fig. 3d shows the sampling times, or controller activation intervals. Comparatively, it can be seen that before the obstacle was introduced both approaches closely track the specified trajectory. When the obstacle was introduced the traditional approach reacted much quicker than the nominal case but both tracked the modified path of the robot trajectory with an offset, $\frac{r_{cj}}{\Lambda(j,j)}$, while still maintaining the pattern of the desired trajectory. r_c is the corresponding modelled obstacle reference for joint j and $\Lambda(j, j)$ is the diagonal component of the matrix described in Section 2. On the other hand using PBSTC, fewer control law updates are required compared to the traditional approach, hence utilizing fewer network resource. After the obstacle has been cleared from the robot's path the traditional approach continued sampling at a constant sampling interval unecessarily wasting network resources.

Experiment 2: Time-Varying Delays to the PBSTC Approach. In this experiment, we consider the effect of time-varying delays. We incorporate a disturbance node in the network to simulate delays. The sampling period of the disturbance node is set to a value of 0.01 seconds, and the disturbance node floods the network with disturbance packets based on a Bernoulli process with parameter d. If X[k] $>$ d, a disturbance packet is forced on the network. Fig. 4a



(a) Joint 2 for PBSTC. (b) Joint 2 for Traditional. (c) PBSTC Sampling Intervals. (d) Traditional Sampling Intervals.

Figure 3: PBSTC Approach vs. Traditional Approach in Nominal Case.



(a) PBSTC With Time Delays. (b) PBSTC Nominal Case.

Figure 4: PBSTC Approach with Time-Varying Delays.

and Fig. 4b show the difference between the desired adjustment in the presence of an obstacle and the actual adjustment of the robot's joint 2 for the time-varying delays and nominal cases respectively. The impact of the time-varying delays can be seen in Fig. 4a, from the delayed response of the system compared to the nominal case in Fig. 4b, in adjusting robot's trajectory in the presence of an obstacle. This is leads to the slightly more accrued error than in the nominal case.

4. CURRENT WORK

In this paper, we assumed a hybrid supervisory control model where the composite system is continuous and interfaces with the event-based controller using an event generator and actuator interfaces which essentially represent the VPS and VPH respectively. Our simulation as well as preliminary results are based on this notion. In reality, in order to realize an actual implementation of the overall system we need a formal framework to define the correct semantics of our system which results in a mixed event and time triggered system presented. Handling mixed event and time-triggered systems is a topic of on-going research. The authors in [7] and [6] presented some approaches on this area of study. We are currently working on a formal framework for our approach to aid in realizing an actual implementation.

5. REFERENCES

[1] N. Chopra, P. Berestesky, and M. Spong. Bilateral teleoperation over unreliable communication networks. *IEEE Trans. on Control Systems Technology*, 16(2):304 – 313, 2008.

[2] N. Kottenstette, X. Koutsoukos, J. Hall, P. Antsaklis, and J. Sztinapovits. Passivity-based design of wireless networked control systems for robustness to time-varying delays. In *29th IEEE Real-Time Systems Symposium (RTSS 2008)*, 2008.

[3] W. S. Levine. *Control System Applications*. CRC Press, 2000.

[4] M. Mazo and P. Tabuada. Input-to-state stability of self-triggered control systems. In *48th IEEE Conference on Decision and Control*, pages 928 –933, Dec. 2009.

[5] R. Ortega and M. Spong. Adaptive motion control of rigid robots: A tutorial. In *27th IEEE Conference on Decision and Control*, pages 1575 – 84, 1988.

[6] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. *International Workshop on Hardware/Software Co-Design*, 0:187, 2002.

[7] N. Scaife and P. Caspi. Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems. In *16th Euromicro Conference on Real-Time Systems*, pages 119–126, 2004.

[8] J. E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[9] J.-J. Slotine and L. Weiping. Adaptive manipulator control: A case study. *IEEE Trans. on Aut. Control*, 33(11):995 –1003, Nov. 1988.

[10] M. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley and Sons Inc., New York, USA, 2006.