Control kernel based adaptive control implementation

Raúl Simarro, Pedro Albertos
Department of Systems Eng. and Control
I.U. de Automática e Informática Industrial
Universitat Politècnica de València
Valencia, 46071, Spain
{rausifer,pedro}@isa.upv.es

José E. Simó
Department of Computer Engineering
I.U. de Automática e Informática Industrial
Universitat Politècnica de València
Valencia, 46071, Spain
jsimo@disca.upv.es

ABSTRACT

A control system with distributed computing resources always should guarantee the safe control of the plant. In this contribution, the concept of control kernel is used for that purpose. Two types of nodes with different resources are defined: the powerful server node and the resource-constrained light node. This architecture allows to split the control tasks into two blocks. Those demanding strong computing resources are allocated in the server nodes and those compelling tasks required to ensure the safety of the controlled plant are allocated in the light nodes. Resource limitations lead to control adaptation. Two simple applications illustrate some of the benefits of this architecture with one server node and one light node, even the architecture can be extended to several nodes. In the first case, an adaptive control is implemented in the server node, providing the control algorithm to the light node, which is also able to compute a local safe control action. In the second experiment, two different control tasks requiring different resources are implemented in a mobile robot control. To keep bounded the computing time at the local level, the supervisor decides the time allocated to each activity, providing the resulting controller to the light node.

Categories and Subject Descriptors

C.3 [**Special-Purpose and application-based systems**]: Process control systems, Real-Time and emdedded systems

General Terms

Performance, Experimentation, Design

Keywords

Distributed control systems, adaptive control, embedded systems, control middleware

1. INTRODUCTION

Adaptive control requires, in a direct or indirect way, to carry out a parameter estimation task to compute and update the controller parameters [3]. In many cases, adaptation also implies changes in the controller structure as well as past data retrieval. All these tasks may require a lot of computation time, not being suitable to be implemented in an environment with limitation of resources.

Moreover, safety is a crucial issue in embedded control systems [5], [1]. Independently of the number of variables to be controlled by the same processor, the systems with hard real time requirements must ensure the delivering of control actions to all actuators. The quality of the delivered signal may depend on the processing level: data, computational algorithms and resources availability, among others, but always must ensure the safe system operation [2]. Besides components malfunction, in complex control systems, safety can be affected by either the appearance of high priority aperiodic tasks, the variation of the controlled system dynamics requiring switching controllers, the missing of execution deadlines and messages or the variation of communications delays. In this context, in order to run control applications in a safe mode, if the control action has not been delivered one time by the current controller, a back-up control action should be delivered at the time required by the process. This signal may be the result of a simple calculation (but sufficiently safe), an emergency shutdown or simply a safe response such as: *keep unchanged*. Note that this operation can be interpreted as a controller switching.

There are many different approaches to design and implement embedded control systems, (see, for instance, [6], [8]). In this work, the concept of *control kernel* [2] is used to compute the control in two stages. This control architecture has been implemented on multiple fully automated mobile vehicles performing activities requiring coordination between them to avoid obstacles, path tracking, scanning, data collection, etc.. To do this one of the robots acting as the supervisor calculates the trajectories to be followed by others, and adapts to control environmental conditions, so the others can use their computation time for data collection, processing products, or other tasks, delegating the calculation of the trajectory to the supervisor. In case of communication problems or excessive computation time they can apply a safe control action.

The control kernel approach presents some novel properties based on the isolation provided by the middleware implementation. Control tasks are moved to nodes where its execution is more efficient, based on the observed availability

of resources. As a consequence, the architecture provides a transparent framework to combine different controllers to be applied as decided.

In this paper the control kernel architecture is used to easily adapt the control goal and performance according to the existing resources. The adaptation can be motivated by external disturbances, changes in the process parameters or in the control goals.

The control kernel architecture is summarized in the next section. Then, the adaptive control algorithm is split into two parts to be implemented in different nodes. The proposed approach is tested on two simple experimental systems to evaluate its possibilities. Some results are reported. Finally, discussion is motivated based on these preliminary results.

2. CONTROL KERNEL ARCHITECTURE

Two node types can be defined, [10]: Light nodes and Service nodes (see Figure 1). Service nodes are powerful embedded computers running a full featured RTOS with complete networking with I/O capabilities. Light nodes are small and low power consumption SoC (System on a Chip) processors with limited computing and networking capabilities but complete I/O features.

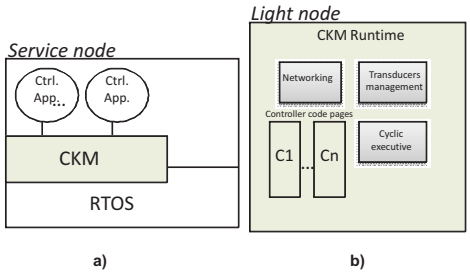

Figure 1: Control nodes.

A control kernel middleware is implemented in both nodes, [10]. In a distributed embedded control architecture, many of these nodes can be interconnected in a wired or wireless network, see Figure 2.

Control applications run in service nodes on top of a full featured Control Kernel Middleware (CKM). This middleware offers abstractions and functionalities related to control tasks real time execution, access to sensors and actuators, and communications management. The programming model of CKM follows the concept of code delegation. In this sense, a control application delegates the execution of some control code to the CKM that provides computational resources to execute it. Note that a control task, once inside the CKM, can run on whatever service node of the distributed control system (DCS) having access to the communications space of the task.

Light nodes are a cost-effective solution in order to allocate some computer power as close as possible of each actuator. This is mandatory in order to reduce the nondeterminism in the time delivering of the control actions to the plant. Light nodes run a retail of the CKM: the CKM Runtime.

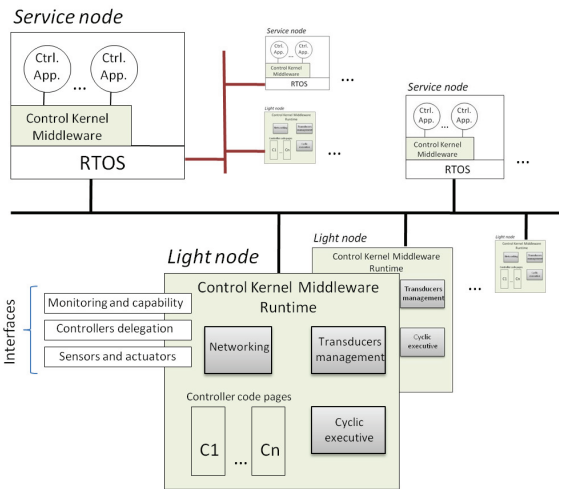

Figure 2: Control Kernel based architecture.

This Runtime communicates with the CKM offering interfaces for management, sensing and acting as well as code uploading. A light node can be used as simple slave component to interface DCS or it can run locally its own controllers in a cyclic executive environment. Ensuring the delivery of an appropriate control action it guarantees the safety of the system, even if this action is to stop the plant operation.

In a control application, any control task that has been delegated to the CKM can be transferred to a light node by uploading the native code page and asking for switching. Controller pages can be uploaded through the CKM Runtime without any interference with the controllers currently running in the node. The uploaded pages are activated for running by the switching mechanism provided by the CKM Runtime. Attention should be paid to the system schedulability [4].

In particular, service nodes may include supervising and optimizing control activities and light nodes can run activities to drive the system to a safe position or run a simple algorithm that guarantees a minimum of performance in the system at any time. In this sense, Light node ensures that a control action ($u(k)$) to be sent to the process always exists. This signal may be just a safe action (disconnect, open, close, unchange, etc.) or the result of a simple calculus (computed locally in the node) ($u_l(k)$) or it may be the signal calculated ($u_s(k)$) and received from a service node.

3. ADAPTIVE CONTROL IMPLEMENTATION

A typical structure of adaptive control involves two loops (Figure 3). The classical feedback loop is keeping the required performance of the controlled plant, whereas the extra loop is in charge of evaluating the quality of the control, performing a parameter estimation algorithm and determining the actual control to be applied to the plant. The actual control may be also rather complex.

Through control kernel architecture, adaptive control struc-

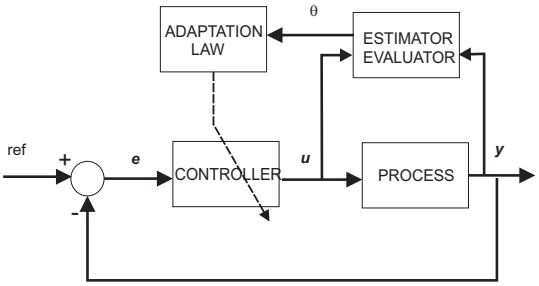

Figure 3: Typical adaptive control structure.

ture can be implemented by a service node, which estimates the signal quality and can change the control law based on the identified needs, while the controller to apply the law control can be implemented in a light node. Thus, tasks requiring a high computational cost are delegated to the service node, while the final application of the control is in the light one.

The most desirable control action, as provided by the adaptive controller which is allocated in the server node, will be delivered if there are enough available resources (communication bandwidth, computing time, memory accessibility and so on). In the end, this control action will be based on a part related to the current measurement (proportional action), one part computed from the past measurements and errors (the integral action) and one (or more) parts due to the error prediction (derivative action). This is clearly illustrated in the case of using a PID adaptive control [9].

In this case the light node will compute the local control action based on a local gain K_l (1)

$$u_l(k) = K_l e(k) \quad (1)$$

where $e(k)$ is the current error. Depending on the local resources, the local control law can be also based on a *frozen* PID controller, provided by the service node but being updated from time to time, not continuously. On the other hand, in the server node the control action $u_s(k)$ (2) will be evaluated

$$u_s(k) = K_s e(k) + K_1 e(k-1) + K_2 e(k-2) + u(k-1) \quad (2)$$

by applying the full control structure and adapting the controller parameters.

Under normal operation, the actual control will be $u(k) = u_s(k)$. In the event of a disturbance (lack of communication, lost of measurement, or just not enough time to compute and adapt the controller parameters $\{K_s, K_1, K_2\}$), the light node will provide the local control action to both the plant, as well as to the server, to be used in future computations.

The adaptation will update the controller parameters and/or structure. By using the control kernel concept, a slightly different alternative is proposed in this paper. Two different control actions are computed. One is very simple and fast, also requiring very few resources, and it could be just a proportional action. It is a *reactive control*. For that, no data retrieval is necessary and the computing time is very short. Unfortunately, the controlled plant behavior will not be the best achievable if a more sophisticated controller were used, but it must ensure the controlled plant stability. This action will be provided by the light node in charge of the process.

4. EXPERIMENTAL WORK

Due to space shortage, two simple applications are presented to illustrate the possibilities of the control kernel approach. As already mentioned, the same approach can be used for more complex applications where there is a general coordination and adaptation at the server level and a local reactive control.

First, an adaptive control of a simulated process suffering high priority communication interferences is evaluated. In this case, the reactive local control is used if the adaptation takes too much time. Then, for a mobile robot with two different tasks (tracking a trajectory and avoiding obstacles) two different control algorithms are used. They are implemented in the server node, being combined in a weighted way according to the operating conditions. The time allocated to each one is adapted to keep constant the total computing time available for this control. At the local node, the final control action is computed and delivered.

4.1 Adaptive control under interferences

A simulated process is controlled to get fast response and adaptation capabilities. The adaptive control algorithm runs in the service node. When the control algorihtm proposes a new controller to keep the control of the system, it is sent to the local controller node involving a controller switch. In the event of a high priority aperiodic task (acting as an interference to the control process computation), the local node keeps running the last updated controller or a simpler back-up one.

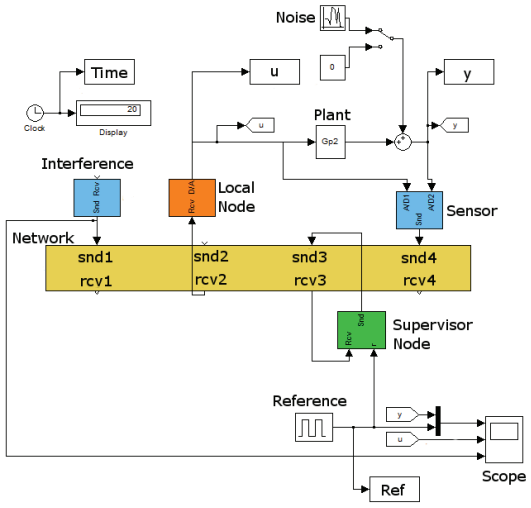

Figure 4: Control kernel implementation and simulation.

The process transfer function for this application (3) maybe time-variant, requiring the control adaptation. In this case

the transfer function is considered constant, the control adaptation being motivated by the appearance of an interference. It has been simulated and evaluated in a Matlab/Simulink environment, using the Truetime tool [7] to evaluate the behavior, as shown in Figure 4.

$$G(s) = \frac{66.777}{s^2 + 9.391s + 77.16} \quad (3)$$

A faster and less performing controller (1) is implemented in the local node to be directly applied to the plant if there is no action coming from the server node. Initially, the plant is controlled through the control action sent by the "complex" controller (2) implemented in the server node (Figure 5).

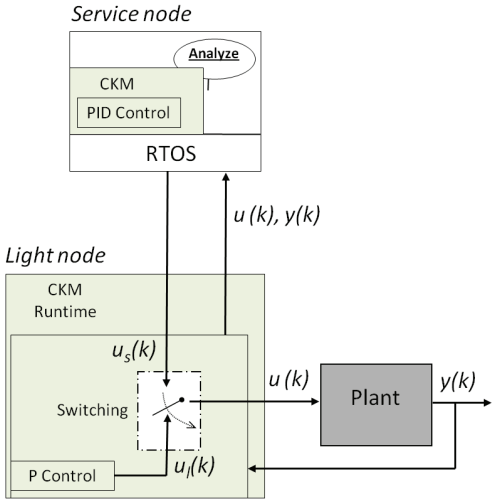

Figure 5: Control structure. Experiment 1

At $t = 6s$, an interference forces the control to be transferred to the basic control implemented in the light node. The control is softer and the plant response is degraded (as can be seen in Figure 6) during the time the interference is active. In the first appearance (interval $\{6, 7\}$), the system is in the transient response and the behavior is degraded. During the second interval ($\{8 - 13\}$) a change in the reference happens and the response is again degraded but as soon as the interference disappears, the control is assumed by the service node and the response is stabilized and improved. It is worth to note that the basic controller does not include integral action, as can be seen in the steady-state error appearing in the plant response before $t = 13s$. Finally, the interference appears just at the time of a reference change, $t = 15s$, and the plant response is also degraded.

4.2 Controller adaptation due to goal changes

In the second experiment, a mobile robot is controlled to follow a trajectory and avoid unexpected obstacles [11]. The trajectory tracking algorithm involves a high computational cost, because it is necessary to estimate the robot position at each instant from sensory information. Thus this algorithm is implemented in the server node, where the information from sensors is sent by the light node. The light node implements a obstacle avoidance algorithm, thus in case of data loss between the server and light node, the robot can always

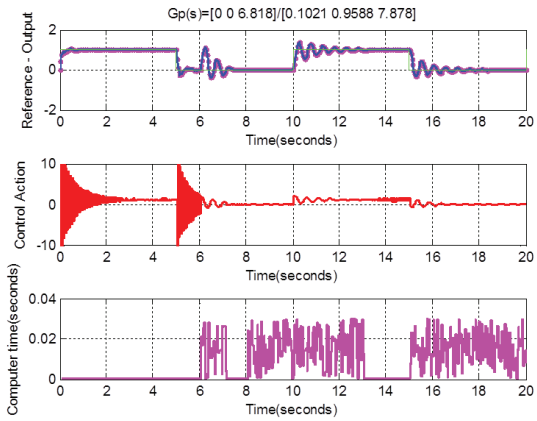

Figure 6: Switching control due to interferences.

avoid obstacles. The robot behavior consists in the combination of the two previous algorithms. The node supervisor is in charge of deciding how much weight must be given to each of the behaviors, in terms of sensory information and the position of the robot (Figure 7).

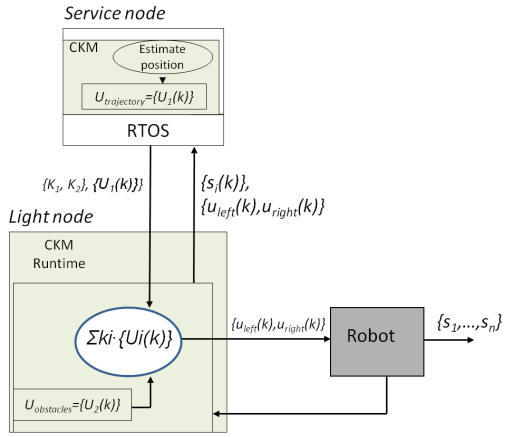

Figure 7: Control structure. Experiment 2

The performance of the control is illustrated in Figure 8, where there are two obstacles and a reference trajectory.

The computing time allocated for the robot in the light node is bounded. This is shown in the upper graphic in Figure 9, where the computing time is almost constant. In the lower graphic, the percentage of time devoted to evaluate the obstacle avoidance control algorithm is shown. As it can be seen, in the time intervals characterized by the cycles $\{0 - 70\}, \{100 - 130\}, \{385 - 410\}$, no obstacles are detected and all the computing time is allocated to follow the trajectory. During the cycles in between, the obstacle starts to be detected and the server node determines the use of the light node to compute the control action to mainly avoid the obstacle, degrading the trajectory tracking performance, (Figure 8).

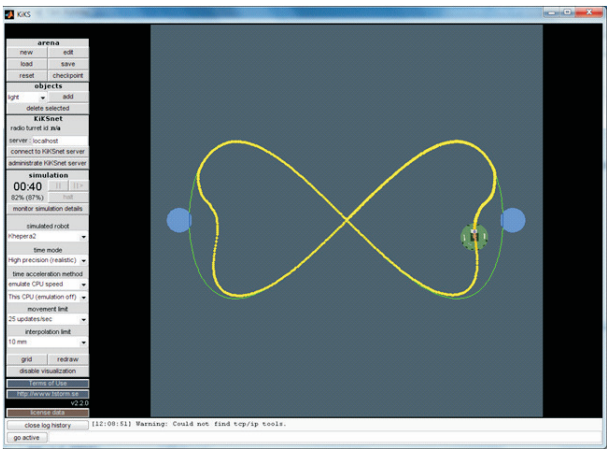

Figure 8: Mobile robot: trajectory tracking and obstacles avoidance.

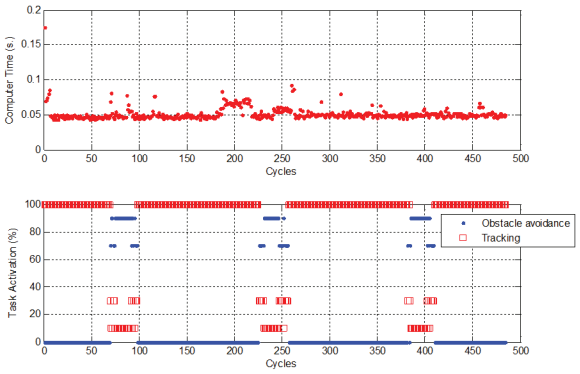

Figure 9: Computer Time in light node and task activation

5. CONCLUSIONS

The use of the Control Kernel structure offers many advantages in allocating different control tasks according to the operating conditions. Two types of nodes are defined: the powerful server node, probably taken care of many control loops and activities, and the resource limited light node, attached to a process, able to handle some related control loops. A number of these nodes can be connected in a network to implement a distributed control system.

In particular, for adaptive control, the more computing power demanding tasks are implemented in the server node where the actual controller structure and parameters are computed. This information is transferred to the local node, where there is also a back-up controller to ensure the delivering of a safety control action.

Two control scenarios have been considered. In the first case, based on a simulated plant, the appearance of aperiodic high priority tasks reduces the availability of computing time and provokes the switching between the adaptive control provided by the server and the basic control computed by the light node (Figure 5), leading to a degrading of the control performance (Figure 6). In the second experiment, where trajectory tracking and obstacle avoidance should be accomplished, the trajectory tracking algorithm task performed in the server node, while avoiding obstacles task and the composition of the two behaviors, with the rules provided by the server node is made in the light (Figure 7).

Many other options are open with this control kernel structure and it is a matter of further research and experimentation.

6. ACKNOWLEDGMENTS

This work has been partially granted by Consellería de Educación Generalitat Valenciana, under PROMETEO project number 2008-088, and Ministerio de Ciencia e Innovación under COBAMI project DPI2011-28507-C02-01/02.

7. REFERENCES

[1] P. Albertos, A. Crespo, M. Vallés and I. Ripoll. "Embedded Control Systems: Some Issues and Solutions." *16th IFAC World Congress*. Prague, Czech Republic. Elsevier, 2005.

[2] P. Albertos, A. Crespo and J. Simó. "Control Kernel: A key concept in embedded control systems." *4th IFAC Symposium on Mechatronic Systems*. 2006.

[3] K. Astrom and B. Wittenmark, *Adaptive Control*, 2nd ed., Addison-Wesley Longman Publishing Co., Boston,MA,USA, 1994.

[4] A. Crespo, P. Albertos, P. Balbastre, M. Vallés, M. Lluesma and J.E. Simó. "Schedulability issues in complex embedded control systems." *IEEE Conference on Computer Aided Control Systems Design*. Munich, Germany, 2006.

[5] Q. Li and C. Yao. *Real-Time Concepts for Embedded Systems*. CMP Books, 2003.

[6] P. Martí, M. Velasco, J. M. Fuertes, A. Camacho and G. Buttazzo. "Design of an Embedded Control System Laboratory Experiment." *IEEE Transactions on Industrial Electronics* 57, n.10 (October 2010): 3297-3307.

[7] M. Ohlin, D. Henriksson and A. Cervin. "TrueTime 1.5 - Reference Manual". *Lund University: Department of Automatic Control*, 2007.

[8] Z. Peng, M. Longhua and F. Xia. "A Low-Cost Embedded Controller for Complex Control Systems." IEEE/IFIP *International Conference on Embedded and Ubiquitous Computing*. Shanghai, 2008. 23-29.

[9] F. Radke and R. Isermann, "A parameter-adaptive PID-controller with stepwise parameter optimization". *Automatica* Vol 23, N 4, July 1987, Pages 449-457.

[10] R. Simarro, J. Coronel, J.E. Simó and J.F. Blanes. "Hierarchical and Distributed Embedded Control Kernel." *XVIIth IFAC World Congress*. Seoul, Korea, 2008.

[11] A. Valera, M. Vallés, P. Albertos, R. Simarro, I. Benítez, and C. Llácer. "Embedded Implementation of Mobile Robots Control." *17th IFAC World Congress*. Seoul, Korea, 2008. 6821-6826.