Flexible Real-time Embedded Architecture for Advanced Service Robots

David Ramada, Carlos Domínguez, Houcine Hassan, Alfons Crespo
Department of Computer Engineering
Universidad Politécnica de Valencia
husein@disca.upv.es

Abstract

In most advanced real-time control applications such as service robots, the tasks have different criticality, flexible timing constraints and variable execution time. For instance, autonomous service robots perform their activities in dynamic environments, share resources and have to cooperate to fulfill their objectives. These changing environmental conditions produce a variability of the system load. Firstly, the computational requirements of recognition tasks are variable and dependent on the number of objects perceived in scenes. Secondly, the application processes are executed at different frequencies with varying periods and deadlines that are dependent on robot speeds. To tackle these aspects a flexible real-time architecture is implemented in rt-linux in this paper. Moreover, the architecture permits to extract the slack time, enabled in the system due to load variability, and to invest it in improving the communication performances of the robots. Experimental evaluations of the architecture have been carried out with real autonomous robots.

1. Introduction

Service robots move in spaces (i.e. hospital, campus) and can interact with humans and other robots. They manipulate objects and share resources. Their tasks consist of inform, guide, care and accompany people, control of traffic, surveillance, transportation, etc. Most of these tasks have variable computation characteristics. For instance, the service robots perform their activities in dynamic environments, they share resources and have to cooperate to fulfill the objectives. The conditions of the environment have influence on the variability of the computation time of the vision system and the recognition tasks. These tasks are variable and dependent on the number of objects that are perceived in the spaces. On the other hand, some processes of the applications (i.e. sensory system) are executed at different frequencies, with varying temporal requirements (periods and deadlines) depending on the speed that the vehicle applies. To ensure the robot safety requirements, the periods and deadlines of real-time tasks have to be gradually adjusted and proportionally to the actual robot speed [1].

This paper presents a real-time architecture that supports the computational variability of service robotic applications. A feedback control scheduler (FCS) is designed to carry out the execution of the variable hard-real time tasks that represent the critical processes of robots (i.e. obstacle_avoidance).

On the other hand, the variability of the load enables sufficient slack time in the system that could be used to improve the communication performances of the robotic system. Enhanced communication processes can be scheduled as optional tasks during the slack time periods. Therefore, a flexible server (FS) is implemented to cope with the optional communication real-time tasks. Moreover, heuristic strategies are designed to permit the

distribution of the optional time extracted accordingly to the communication requirements of the robotic applications, and hence improve the communication performances of the robots.

After the introduction, section 2 reviews previous work performed on variable computation and QoS management. In section 3, the components of the real-time architecture are described. The implementation of the architecture in rt-linux is presented in section 4. Experimental evaluations of the different subsystems in real environments are carried in section 5. Finally, conclusions are summarized in section 6.

2. Related work

In the literature, different techniques focusing in the adaptation of the characteristics of the load to the environmental conditions and in improving the quality of service of real-time applications have been emerged as major research issues. In [2, 3, 4] a feedback control scheduler based on the feedback control theory is used with a proper QoS actuator for adjusting the task QoS levels in order to minimize the deadline miss ratio, when task's execution time are not known. In [5, 6] periodic computations are modeled as springs with given elastic coefficients and minimum lengths. The spring elastic coefficients are used to change the rates of the periodic tasks under overload conditions or when variations in task execution rates are requested. However, [7] proposes the use of concepts such as task values and priorities rather than spring coefficients. Beccari [7] proposed a rate adaptation scheduling framework for soft real-time periodic tasks characterized by a range of admissible rates. When the rate of some task is required to change by the high-level control system, the remaining soft real-time tasks can be adapted based on heuristic criteria. Abdelzaher [8] use QoS contracts specifying degraded acceptable QoS levels to obtain a graceful degradation of the communication subsystem. Under overload or under-utilisation conditions, a QoS optimization process attempts at maximizing the aggregate reward, given rewards/penalties information specified by QoS contracts. A method that integrates a reservation mechanism with an application level QoS adaptation strategy is presented in [9]. The method controls the CPU bandwidth reserved to a task and allows each task to change its QoS requirements (enlarging the periods or skipping task instances) if the amount of reserved bandwidth is not sufficient to accomplish its QoS. A scheduling algorithm maximizing the effective processor utilization during overload, given a minimum slack factor for all tasks, is presented in [10]. The algorithm is based on EDF, hence it does not guarantee which tasks will not be affected by overload. In [11] the skip approach assumes that certain tasks could abort some instance during a periodic execution, especially for applications, such as process control architectures..

3. Real-time architecture

3.1 Service robots

The services that the robots have to perform can be appreciated in the Figure 1.

Figure.1. Robot services.

The robots move in the space of the schools and can interact with humans and other robots. They manipulate objects and share resources with the former agents. Some of the tasks that the robots have to perform are to inform and guide people, to care about persons and to accompany them if they are lost. They can work in the control of traffic when it is heavy and they perform the surveillance of the spaces of the campus and objects and their transportation. Robots can perform diagnosis of systems and proceed to the reparation of failures.

The communication protocol is established between two agents. An agent can be either a robot or a human. An agent i asks the closest agent j for some service. The agent j assumes the requirement of agent i. Then, agent j evaluates the situation, and it can solve it or it looks for cooperation of other agents. The cooperation consists of Agent-i delegates/exchange a goal and agent-j assumes/refuses the goal.

Figure 2 shows the organisation of the tasks in the system. The periodic tasks support the execution of the reactive processes of the robot. The optional tasks are used to improve the communication quality of the periodic tasks. The communication tasks are executed in the slack periods as optional tasks to improve the collaboration between the agents.

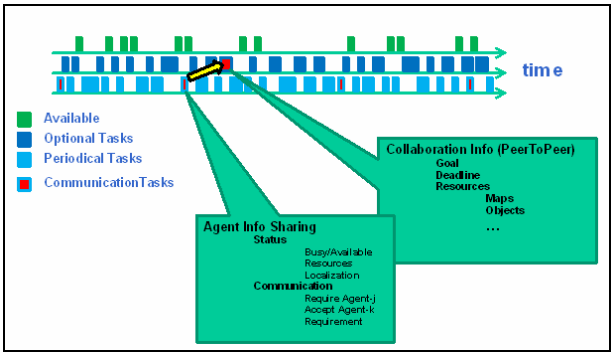


Figure.2. System scheduling.

3.2 Task model

In order to cope with the process temporal and computational requirements identified in Figure 2, a task model, according to the hardness of the timing constraints of these processes is proposed. Three task sets define the task model:

$$T = T^f \cup T^v \cup T^o$$

Where: T^f is the fixed periodic task set, T^v the variable periodic task set and T^o the optional task set.

The requirements of processes related to low-level reactions (i.e. detecting and avoiding obstacles) must be strictly guaranteed. Since these temporal requirements can be either fixed or variable they are mapped to either fixed or variable periodic tasks.

The fixed periodic task set, T^f, represents the periodic processes that don't depend on any application parameter. A fixed periodic task, T_i^f, is characterized by the temporal parameters of the definition (1).

$$T_i^f = \{C_i, P_i, D_i, \mathrm{Pr}_i, \phi_i, m_i\} \quad (1)$$

Where, C_i is the worst case execution time, P_i is the period, D_i is the deadline, Pr_i is the priority, ϕ_i is the offset and m_i is the motivation of the *i* task.

The variable periodic task set, T^v, supports the execution of processes that are dependent on the environment and/or on application parameters. The characteristics of variable periodic task, T_i^v, are expressed with the temporal parameters of the definition (2).

$$T_i^v = \{C_i(k_1), P_i(k_2), D_i(k_2), \mathrm{Pr}_i, \phi_i, m_i\} \quad (2)$$

In this case, the computational time, C_i is variable and depends on the application parameter k_1 (i.e. number of obstacles in the scene). P_i and D_i attributes are also variables and depend on the parameter k_2 (i.e. robot speed). These variable temporal parameters should accomplish the conditions of the definition (3).

$$\begin{aligned} C_i^{\min} \le C_i(k_1) \le C_i^{\max} \\ P_i^{\min} \le P_i(k_2) \le P_i^{\max} \\ D_i^{\min} \le D_i(k_2) \le D_i^{\max} \end{aligned} \quad (3)$$

$C_i^{\min}$ and $C_i^{\max}$ are the minimum and maximum computational time corresponding respectively to an under-loaded and overloaded environment. $P_i^{\min}$ and $D_i^{\min}$ are the minimum task periods and deadlines respectively, these are applied for the maximum robot speed. When the speed is minimum, the maximum temporal requirements $P_i^{\max}$ and $D_i^{\max}$ are applied.

The high-level deliberative processes of the application do not require strict guarantees (i.e. *collaboration info task* (see Fig. 2)) and their execution is not mandatory. The mandatory communication (i.e *Agent info Sharing* task of Fig 2) is executed as periodic tasks. To deepen in the collaboration, optional tasks are scheduled.

Consequently, the enhancing communication processes are modelled with optional soft aperiodic tasks (Liu 1991). Each optional task, T_i^o, is composed of different quality levels. The first level generates a minimum communication quality response of the task. By increasing the level, the quality of the result is improved. The characterisation of an optional task, T_i^o, is given in (4).

$$T_i^o = (L_{i,j}, \mathrm{Im_i}) \tag{4}$$

Where $L_{i,j}$ is the estimated computational time of the j^{th} level of the i^{th} task. Im_i is the importance of the optional task.

3.3 Feedback control scheduler

The execution of the fixed and variable periodic tasks is supported by the feedback control scheduler (FCS) of the Figure 3. The FCS regulates two control loops. In the first loop the FCS has to adapt the periods and deadlines of tasks accordingly to the requested robot speed. And in the second loop, the utilisation of the reactive processes is maintained between predefined utilisation limits (i.e. $U_{\min}$ =40% and $U_{\max}$ =70%).

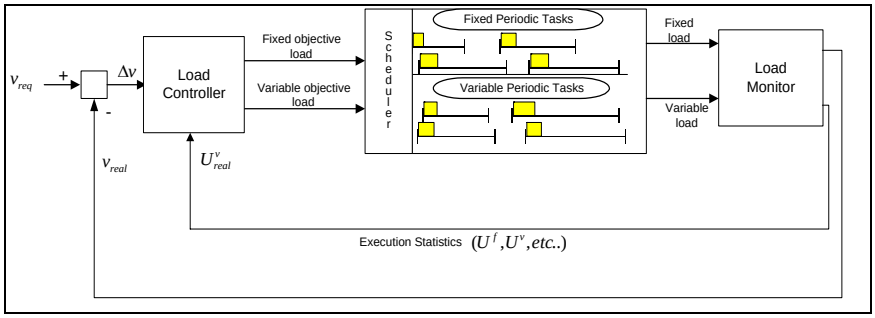


Figure 3. Feedbak control scheduler.

The FCS is composed of a monitor, a controller and a fixed priority-based pre-emptive scheduler (rate-monotonic scheduler [13]. The monitor performs the monitoring of the actual load of the system, it updates the actual variable utilisation and the actual speed and sends them to the controller. The controller receives, in addition, speed requests from the application and is responsible for the regulation of the system to satisfy the requested speed and to preserve the utilization limits.

The regulation of the system takes place when the actual utilisation doesn't respect the utilisation limits, and/or a robot speed change is required. Hence, if the actual utilisation doesn't respect the limits and no robot speed change is required, the controller analyses the causes of the load rejection –because of the system saturation or laxity. If the system is overloaded, the controller performs a load relaxation by increasing the periods. Otherwise, the load is increased by reducing the variable task periods. Instead, if the actual utilisation respects the limits and the agent requires a speed change, the controller should analyse whether the utilisation generated by the required speed is accepted or not. If it is accepted then it is executed by adjusting the

periods, accordingly to that speed. Otherwise, the rejection causes are analysed and the regulation is solved as has been done previously. Finally, when the two control loops are not fulfilled, the controller will perform the regulation of the speed. Once the controller has decided that a regulation of the system should be performed, it would calculate the new periods of the tasks.

3.4 Flexible server

The flexible server (FS) schedules the optional tasks in the slack time of the system. To extract this time, the FS incorporates a slack time controller which has been adapted to cope with the variable periodic tasks. To improve the performances of the communication, heuristic strategies are implemented to distribute the extracted idle time among optional tasks, accordingly to the application requirements.

3.4.1. Slack time controller. The slack time is calculated with a variation of the dynamic approximate slack stealing algorithm [12]. This algorithm has been extended to take into account the variable temporal and computational characteristics of the system load. The actual available slack time, s_{real} , is calculated by means of the equation (5).

$$S_{real} = \min_{\forall j \in lp(i) \cup \{i\}} S_j(t) \geq 0 \tag{5}$$

where *lp(i)* is a task set whose priorities are lower than task *i* priority level and to cope also with the variable characteristics of the load, the calculation of $S_i(t)$ is extended as can be seen in the equation (6).

$$S_i(t) = \left(e - \left(\sum_{\substack{j \in hp(i) \cup i \\ j \in T^f}} I_j^f(t,e) + \sum_{\substack{k \in hp(i) \cup i \\ k \in T^v}} I_k^v(t,e) \right) \right) \tag{6}$$

Being *hp(i)* the tasks with higher priority levels than *i* task, $I_j^f(t,e)$ the interference produced by the *j* fixed periodic task and $I_j^v(t,e)$ the interference produced by the *j* variable periodic task obtained with the equation (7).

$$I_j^v(t,e) = c_j^{\max} + f_j^v(t,e) \cdot C_j^{\max} + \min(C_j^{\max}, (e - x_j(t) - ST_j(t,e))_0) \tag{7}$$

Where $C_j^{\max}$ is the maximum computational time of the *j* variable task. $f_j^v(t,e)$ is the number of complete invocations of the *j* variable task in $[t, t+e]$. $ST_j(t,e)$ is the total sum of the variable periods of the *j* variable task in $[t, t+e]$. The calculation of $f_j^v(t,e)$ and $ST_j(t,e)$ is performed with a specific algorithm, taking into account that the variation of the actual period P_j^{real} during the transition phase is performed at Δt_j rate; P_j^{real} is updated in each activation such as $P_j^{real} + \Delta t_j$. The overhead introduced by the new

dynamic approximate slack stealing algorithm is *O(n)*, *n* is the number of tasks. This complexity is increased to *O(mxn)*, when the system is in a transition phase. Where *m* is the number of activations of a *j* variable task within [t, t+e].

3.4.2. Heuristics. Different heuristic strategies permit to improve the performances of the communication by executing the optional tasks depending on different profiles of the application: *most-important* strategy, *importance-rotation* strategy, *balance* strategy. By means of the heuristic strategies it can be distributed differently the slack time among the different optional tasks. The strategy applied depends on the missions that service robots have to carry out in each situation. For instance, when the robot is exploring the environment and is looking for a specific material, or place, then the robot will require more attention and hence the *most-important* strategy that grants the biggest part of the optional time for the communication, is applied.

3.5. Schedulability analysis

The FCS guarantees the execution of the fixed and the variable periodic tasks while the FS guarantees the execution of the optional tasks without jeopardizing the execution of the periodic tasks. Before run-time, fixed priorities are assigned to the periodic tasks. The priorities of the mandatory and the action parts of periodic tasks should be the same. During the system execution these tasks keep their priority fixed. To guarantee the execution of these tasks, the Rate Monotonic Scheduler is used and schedulability analysis for such algorithm is applied [13]. Although variable periodic tasks introduce changes in the run-time workload, this should not exceed the maximum system utilization that is guaranteed before run-time by the RMS, and therefore, the variable run-time workload is always guaranteed. In the FS, the adapted dynamic approximate slack stealing algorithm is responsible of obtaining the available idle time in the system, while the shedulability of the periodic tasks is preserved. The gained time is invested in scheduling optional tasks at the priority of the FS, which is in turn executed at the priority of the periodic task that previously invoked it. Therefore, the optional tasks don't interfere in the system schedulability guarantee because they are scheduled in the slack time provided by the FS.

4 Architecture Implementation

The architecture has been embedded in the real-time linux kernel [14] as can be seen in the Figure 4. Service robotic applications (agent.conf) are specified with the lex and yacc tools. A specific parser (agent.ini) transforms the application entities to real-time linux data structures and tasks. After the initialization, the components of the emotional agent can be changed or updated with a linux process (agent.mod). Likewise, monitoring tools (log_event and log_app) have been developed in the space of the Linux O.S to monitor the state of the objects and task requirements of the robot during the execution performed in the rt-linux space. The communication between Linux and rt-linux is performed through *fifo* channels.

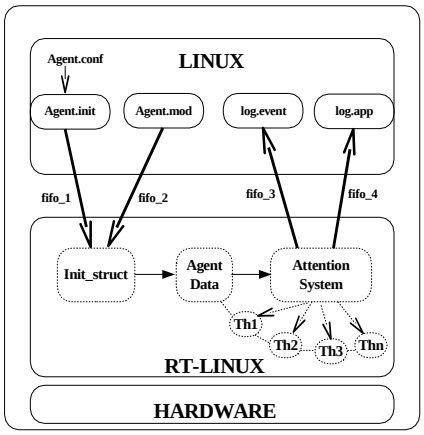


Figure 4. Architecture implementation.

Taking into account the computational requirements of the entities of the service robot, the task model composed of fixed and variable periodic tasks has been incorporated to the rt-linux kernel. The attention system embeds the rate-monotonic scheduling algorithm to guarantee the execution of fixed and variable periodic tasks, and it self performs the schedulability analysis based on fixed priority pre-emptive theory. The slack server is executed as a real-time Linux task. When it starts the execution it calculates the available slack time by executing the algorithm of equation (6). This time is sent to the heuristics executed as rt-Linux tasks. The selected heuristic shall plan the execution of the optional communication tasks in the rt-Linux space.

5. Experimental evaluation

Different properties of the real-time architecture are evaluated in this section. Firstly, the FCS flexibility under different environmental conditions of the robots are shown. Secondly, the idle time extraction capacity of the slack time controller when robot speed variations take place, is evaluated. Finally, the heuristic strategies have been assessed to show how they distribute the optional load according to the robot profiles.

5.1. FCS: a flexible real-time scheduler

The FCS has been evaluated to show how it adjusts the periods and deadlines to preserve the laxity and saturation utilization limits [30%, 70%] when the computation time of the tasks varies due to the obstacles and other robots found in the environment. For this experiment, real autonomous vehicle processes of the Table 1 have been used.

Table 1. Temporal characteristics of vehicle processes

Reactive Tasks	$[C_{i,m}^{\min}, C_{i,m}^{\max}]$	$C_{i,a}$	$[D_i^{\min}, D_i^{\max}]$	$[P_i^{\min}, P_i^{\max}]$	α_i
Obstacle_avoidance	[10,20]	1	[250, 2500]	[250, 2500]	0.25
Go_objective	2	0	[200, 2000]	[200, 2000]	0.2
Odometry	1	0	[100, 1000]	[100, 1000]	0.1
Local_map	[70,120]	0	[225, 2250]	[225, 2250]	0.225
Trajectory	1	0	[285, 2850]	[285, 2850]	0.285
Speed	1	0	[275, 2750]	[275, 2750]	0.275
Mission_planner	[5,15]	0	[300, 3000]	[300, 3000]	0.3

The computation variations have arisen in the time instants of the Table 2.

Table 2. Load variation instants

Time (ms)	$C_1(t)$ (ms)	$C_4(t)$ (ms)	$C_7(t)$ (ms)
t=0	10	70	5
t=2025	10	90	5
t=2100	10	90	10
t=2250	15	90	10
t=4050	15	100	10
t=4200	15	100	15
t=4275	15	130	15
t=11940	15	90	15
t=11966	15	90	10
t=12089	10	90	10
t=14520	10	70	10
t=14540	10	70	5
t=14571	5	70	5

The first column shows the time instants where the computation of the variable tasks (1, 4 and 7) are changed, due to environmental conditions. The three other columns represent the actual task computation.

The results of the adaptation of the system utilisation to the workload computational variations of the Table 2 can be appreciated in the Figure 5.

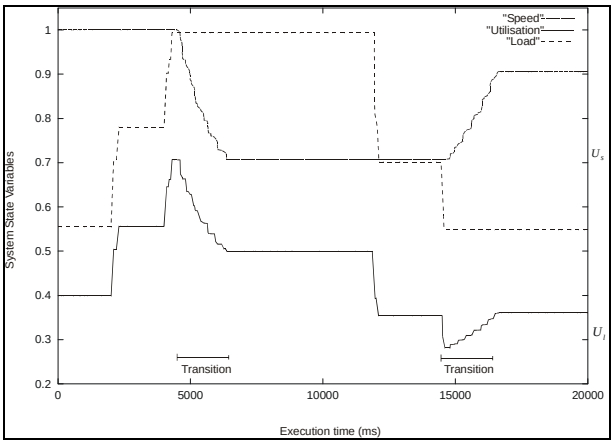


Figure 5. System adjustment (T=2s).

Initially, the environment contains obstacles and the load is about 55.48%. The speed is maximum (1m/s) and hence the utilisation is 39.89%. The environment becomes crowded in t=2100 ms, the load increases to 78.4% and the utilisation to 55.64%. In t=4395ms the vehicle is in a zone where the number of objects is maximum. The task computations is 100% and the utilisation increases to 70.64% - exceeding the saturation

utilisation limit. To allow the correct recognition of this zone, the speed is required to be reduced from 1m/s to 0.7 m/s and therefore the system is being regulated by reducing the utilisation (because of the increase of the periods and deadlines). The transition phase, due to agent dynamics (to fulfil security requirements), is considered to be 2s. It can be seen how the speed and the utilisation take 2s to reach their steady state -from 4700ms to 6700ms. After, in t=12112ms the vehicle detects that the environment is becoming under-loaded. When the load is 54.81% (speed=0.71 m/s) the utilisation drops to 28.1% -under the laxity utilization limit. Consequently, the controller performs an increase of the vehicle speed to 0.9 m/s which will cause a growth of the utilization.

The FCS maintains adequately the real utilization of the system between the laxity and saturation pre-established system limits and, the utilization adjustment is performed accordingly to the required dynamics of the vehicle.

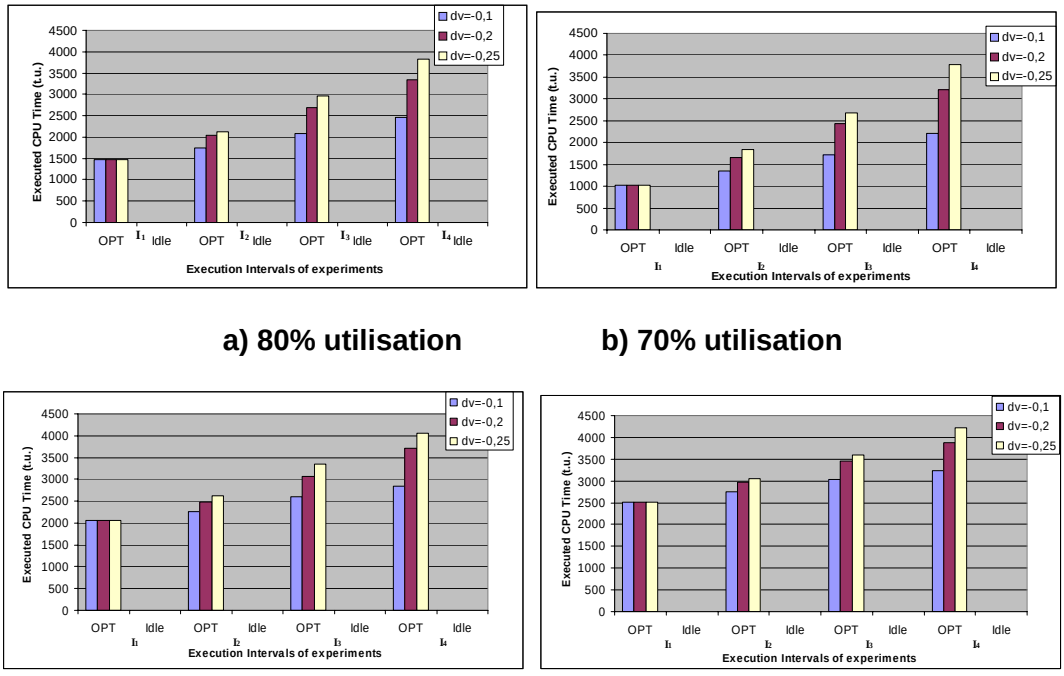
5.2. Slack time controller: Extracting idle capacity

This section evaluates the performance of the system regarding the extraction of idle time when the vehicle speed changes. The load model is composed of variable periodic tasks sets of 80%, 70%, 60% and 50% utilisation levels. Each set is composed of 10 tasks. The periods are comprised in the range of 10 and 1000 t.u. The maximum hyperperiod considered is 20000 t.u. A synthetic load generator has been used. For each utilization level, the experiments have been repeated 10 times. The *OPT* task executes the extracted idle time. The capacity that cannot be extracted is consumed by the *idle* task. In each experiment E_i, the system is submitted to the speed changes of the Table 3 during time intervals of 5000 t.u.

Table 3. System speed variations.

Time intervals (*ms*)	E_1	E_2	E_3
	$v_{req}\,(m/s)$	$v_{req}\,(m/s)$	$v_{req}\,(m/s)$
$I_1=[0-5000]$	1	1	1
$I_2=[5001-10000]$	0.9	0.8	0.75
$I_3=[10001-15000]$	0.8	0.6	0.5
$I_4=[15001-20000]$	0.7	0.4	0.25

The changes are performed in the instants: t=5001, 10001 and 15001. Three experiments have been carried out: E_1 , E_2 and E_3 . The experiments have been executed for different transition phases T=2000 t.u., 3000 t.u. and 4000 t.u. The results of extracting the idle time for the generated utilisations of 80%, 70%, 60% y 50% can be appreciated in the Figure 6.



a) 80% utilisation **b) 70% utilisation**

c) 60% utilisation **d) 50% utilisation**

Figure 6. Idle time extraction.

The different simulations show that the extracted idle time increases progressively for each of the time intervals because the temporal requirements of the tasks increase as the system speed is being reduced. For all the experiments, the *OPT* task consumes all the available idle time in the system, while the *idle* task doesn't execute any time. For example, in the first interval, *OPT* consumes 1017.5 t.u. what corresponds to all the idle time of the interval [0, 5000], since the mean utilization is 79.65% (3982.5 t.u.). It can be concluded that all the available idle time has been extracted when the system performs speed changes.

5.3 Heuristic scheduling strategies

In this experiment the heuristic strategies have been evaluated to show how they distribute the idle time among the optional communication tasks according to different robotic application profiles. For the experiment, the periodic load has been obtained for utilization levels of 40%, 50%, 60% and 70%. The hyperperiods are 25000 t.u. maximum. The 10 sets of tasks are composed of 10 tasks each. Regarding the optional tasks, it has been considered 5 refinement levels per task. The computation time of each level has been randomly generated between the *wcet* of the periodic task plus one and its *deadline* minus its total computation.

Figure 7a shows the results of scheduling the optional tasks when the *most important* strategy is selected. It can be observed that the idle time is granted to the more important tasks: *OPT1* to *OPT5*. This heuristic can be interesting when a *master* robot needs to communicate with *slave* robots.

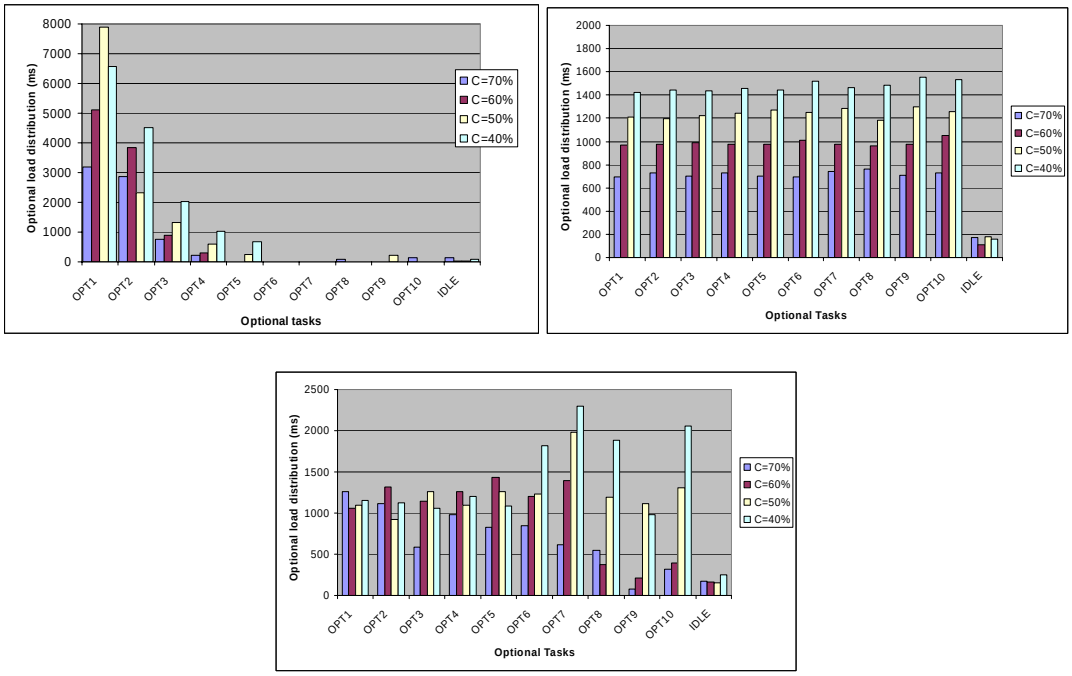


Figure 7. Heuristics: a) Most important and b) Importance_rotation c) Balance

To solve the greediness of the previous heuristic the *importance-rotation* heuristic has been implemented (Fig. 7b). It can be noted that the more frequent tasks have smaller optional levels and vice-versa. This situation is due to the fact that when the system is overloaded the less frequent tasks have big optional parts that cannot fit in the small available slack times, however the optional part of the more frequent tasks are smaller and fit better. On the other hand, when the system is relaxed, the less frequent tasks can execute their maximum optional times, consequently, the more frequent tasks have less slack time to execute their optional levels. Although the *importance-rotation* heuristic distributes better the idle time among the different tasks than the *most important*, the load is not yet totally balanced. This strategy can be interesting to prioritise the communication of the more critical tasks under overloaded conditions.

To balance the consumption of the idle time among the optional tasks, the *balance* heuristic is designed. This strategy gives more importance to the optional task that consumes less idle time. The results obtained by this strategy are plotted in the Figure 7c. It can be observed that in each of the experiments, the execution of the idle time is more suitably balanced among all the optional tasks. This strategy is useful when it is convenient that all the robots transmit information.

5.3.1 Strategy comparison. The heuristic criteria should schedule high amount of optional tasks to be useful for the improvement of the robot performances. Figure 8 shows, for each heuristic, the utilization of the optional load related to the periodic utilization.

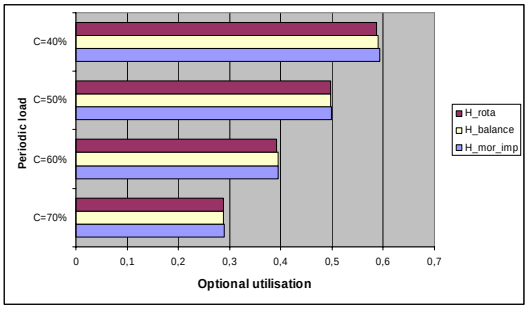


Figure 8. Optional load

The heuristics take very well advantage of the available idle time as can be seen in Figure 8. The *most-important* heuristic consumes a mean of 99.5% of the available idle time left by the periodic tasks, the *balance* heuristic is the second one that profits better the idle time since it uses a 98,5%, and finally the *importance-rotation* heuristic consumes a mean ratio of 98.2% of the available idle time. In fact, the consumed idle time is very significant for the three evaluated heuristics. The *most-important* heuristic dedicates greater idle time for the optional tasks because it allocates the time to the most important tasks that have smaller refinement levels.

6 Conclusions

The analysis of the temporal, computational and communication requirements of the processes executed in advanced real-time robotic applications and the identification of their dependency on the changing environmental conditions has been performed. To cope with the computational variability of the real-time vehicle applications a flexible real-time architecture has been developed. To schedule the variable hard-real time tasks that represent the reactive load, a feedback control scheduler that permits the adaptation of the task periods and deadlines, has been designed. A flexible server, responsible of the execution of soft real-time tasks representing the communication processes, has been implemented. To improve the performances of the communication system of the robots, accordingly to application profiles, a set of heuristic strategies have been defined. The implementation of the architecture has been performed in Linux and rt-linux O.S. The load regulation of the FCS, the idle time extraction capacity of the FS and the scheduling heuristics have been evaluated using real autonomous robotic application performing their tasks in dynamic environments. In the future work, the design of a task model better representing the environmental conditions and parameter applications is planned. The evaluations of other dynamic scheduling policies of the FCS are being considered, to improve the FS implementation different soft real-time servers are being studied, and a multiprocessor based platform to embed the architecture will be analyzed.

Acknowledgments

This work has been partially funded by the Spanish government under grant DPI2006-15320-C03-013 of the national research commission of science and technology.

Reference

[1] Domínguez C., Hassan H., Crespo A, (2005). Real-Time Robotic Agent Based on Emotions. *International Journal of Software Engineering*. pp. 79-90, Vol2. Nº 1. January 2008.

[2] Stankovic, J. A., Lu, C., Son, S. H. and Tao, G. 1999. The case for feedback control real-time scheduling. In Proc. Euromicro Conference on Real-Time Systems, ECRTS'99, York, UK.

[3] Ch. Lu, J. Stankovic, G. Tao, S. Son. "Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms". Special Issue on Control-Theoretical Approaches to Real-Time Computing, *Journal of Real-time Systems*. 23(1/2), May 2002.

[4] Lu, C., Stankovic, J. A., Abdelzaher, T. F., Tao, G., Son, S. H. and Marley, M.. Performance specifications and metrics for adaptive real-time systems. In Proc. IEEE Real-Time Systems Symposium, RTSS'00, Orlando, FL.

[5] Buttazzo, G. C., Lipari, G. and Abeni, L.. Elastic task model for adaptive rate control. In Proc. IEEE Real-Time Systems Symposium, RTSS'98, Madrid, Spain.

[6] G. Buttazzo, G. Lipari, M. Caccamo,L. Abeni, "Elastic Scheduling for Flexible Workload Management" IEEE Transactions on Computers, Vol. 51, No. 3, pp. 289-302, March 2002.

[7] G. Beccari, S. Caselli, F. Zanichelli, " A Technique for Adaptive Scheduling of Soft Real-Time Tasks". Journal of Real-Time Systems, vol. 30, pp. 187–215, 2005.

[8] Abdelzaher, T. F. and Shin, K. G 1998. End-host architecture for QoS-adaptive communication. In Proc. IEEE Real-Time Technology and Application Symposium, RTAS'98, Denver, CO.

[9] L. Abeni and G. Buttazzo, "Hierarchical QoS Management for Time Sensitive Applications", Proceedings of the IEEE Real-Time Technology and Applications Symposium, Taipei, Taiwan, May 2001.

[10] Baruah, S. K. and Haritsa, J. R 1997. Scheduling for overload in real-time systems. IEEE Transactions on Computers 46(9):1034–1039.

[11] Caccamo, M. and Buttazzo, G. 1997. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In Proc. IEEE Real-Time Systems Symposium, RTSS'97, San Francisco, CA

[12] R.I. Davis. "Approximate Slack Stealing Algorithms for Fixed Priority Pre-emptive Systems". *Report Number YCS217*. Real-time Systems Research Group. Department of Computer Science. University of York, U.K. 1993

[13] N.C. Audsley, A. Burns, R. Davis, K. Tindell and J. Wellings "Fixed Priority Pre-emptive Scheduling: An Historical Perspective". *The Journal of Real-Time Systems, **Vol**. 8.* pp. 173-198. 1995.

[14] M. Barabanov and V. Yodaiken, "*Introducing real-time Linux*". Linux Journal 34, Feb 1997.

Authors

David Ramada received the M.S. degree in computer engineering from the Polytechnic University of Valencia (UPV), Valencia, Spain, in 2006. He has been pursuing his Ph.D. at the Department of Computer Engineering, UPV, since 2006. His research interests include the development of real-time control systems, scheduling and real-time operating systems. Mr. Ramada has been involved in different projects in Hewlett Packard, Barcelona since 2006.

Carlos Domínguez received the M.S. degree in electrical engineering from the Polytechnic University of Valencia (UPV), Valencia, Spain, in 1991. He has been an Assistant Professor of the Department of Computer Engineering, UPV, since 1992. His research interests include the development of intelligent agents for mobile robots, real-time systems, and planning and scheduling integration. Mr. Domínguez has been involved in different projects in the Industrial Informatics group since 1989.

Houcine Hassan received the M.S. and Ph.D. degrees in computer engineering from the Polytechnic University of Valencia (UPV), Valencia, Spain, in 1993 and 2001, respectively. He has been an Associate Professor of the Department of Computer Engineering at UPV since 1994. His research interests focus on a number of aspects of the development of hardware and software architectures for embedded processors, embedded systems, real-time systems and their application to control and robotic systems. Dr. Hassan joined the Industrial Informatics group of the UPV in 1993, where he is presently participating in several research and educational projects.

Alfons Crespo is Professor of the Department of Computer Engineering of the Universidad Politécnica de Valencia. He received the PhD in Computer Science from the Universidad Politécnica de Valencia, Spain, in 1984. He held the position of Associate professor in 1986 and full Professor in 1991. He leads the group of Informática Industrial and has been the responsible of several European and Spanish research projects. His main research interest include different aspects of the real-time systems (scheduling, hardware support, scheduling and control integration, ...).