Real Time control of 20 DoF humanoid robot.

Manuel Muñoz, J. Francisco Blanes, Miguel Albero, Javier O. Coronel, José E. Simó, Alfons Crespo, and Salvador Peiró.

Instituto de Automática e Informática Industrial, AI2
Universidad Politécnica de Valencia, Spain
{mmunoz, pblanes, mialgil, jacopa, jsimo, acrespo, speiro} @ ai2.upv.es

Abstract: Robot control is a typical scenario for real time applications benchmarking. Control quality many times is observed in terms of robot interaction and mobility performance. This is more obvious and complex when the robot is inherently unstable like a biped robot, where unaccomplished time requirements could trigger instability and even falls. In this paper is described the work performed to control 20 servomotors actuators, ensuring time accuracy and output actuator quality. The time restrictions about this are a clear case of system which operability is decreasing when control deadlines are violated. Thus, various solutions are analyzed that implements the 20 PPM signals software generation which control the 20 joins in a biped robot. For this work has been used an ARM7 microcontroller in an embedded system that provides robot autonomy in terms of computational resources for control. Different solutions have been implemented that guarantee time restrictions performed in signal generation. This provides a correct join positioning. Finally, a RTOS based version has been implemented and developed on PaRTiKle that make available tasks like: debug, programming, code maintenance, all this assuring time requirement compliance.

Keywords: Real Time Control. Robotics. Real Time Operating Systems. Embedded Systems.

1. INTRODUCTION

The work has been tested on MicroBIRO (Albero, 2007), a humanoid robot with 20 degrees of freedom developed in the Instituto de Automática e Informática Industrial (AI2), at Universidad Politécnica de Valencia. It is a small biped robot (24 cm height) composed by 20 actuators, and small aluminium links between the actuators. The computing capabilities are provided by an ARM7 processor (ARM, 2001)(Seal, 2001), specifically LPC2136 system (NXP, 2008). This CPU can process all the information from sensors, execute control loops and perform the low level controllers for every Degree of Fredom (DoF) Additionally, the embedded system include several communication buses like USB (Peacock, 2005), SPI and Zigbee. For sensors 8 free analog/digital channels are used, and advanced sensor systems are used as feedback in the control loops, like infrared sensor, 3axis accelerometer sensor, and pressure sensors in the sole of their feet. The robot carries a Lithium Polymer battery, which supplies the energy for the actuators and the control system, to dispose of complete autonomy.

To obtain a properly motion, accuracy position management of the 20 joins of the robot is needed, each one of these actuated by one servomotor. To perform this control the resources available are those provided by the embedded system previously described which not include enough PWM (Pulse Width Modulation) outputs in the CPU architecture. For this reason a set of different software solutions have been implemented and tested to accomplish these real time requirements. On this are focused on the main work efforts.

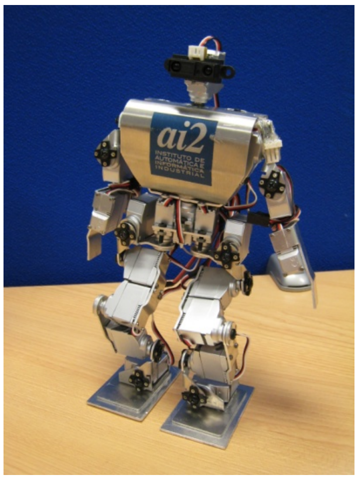


Fig. 1. MicroBIRO.

It's important to highlight that the control system must to be able to guarantee the signal time properties associated with PPM (Pulse Position Modulation, it's a particular case of PWM signaling) motor control to ensure robot integrity. But the case of small signaling errors affects directly the suitable walking of the robot which is inadmissible because degrades robot walking gaits.

This work was supported by the project SIDIRELI (Sistemas Distribuidos con Recursos Limitados) DPI2008-06737-C02-01/02.

Thus, in the case that the time restrictions wasn't accomplished, the robot movements will be degraded, even arriving to the instability, entailing the robot falling. This is a typical scenario for real time control applications, where the system operability decreases when control deadlines are unaccomplished.

Additional, the system has to be capable of attending to other activities like: sensorization, behavior planning, trajectory generation and movements, communications with others. Therefore, the solutions will take in account also the suitable management of the available resources, trying to minimize the utilization of these, so that the rest of activities of the system could be attended.

The organization of the paper is as follows. Section 2 presents the requirements of the servo-motors PPM signal. Section 3 describes the four different solutions for generate the real time PPM signals. Section 4 discusses the results, which include a comparison between different implementations, and discus on benefits of each one. Conclusion and future works are made in section 5.

2. ACTUATORS & PPM SIGNALS.

MicroBIRO robot is configured with 20 articulations, each of them composed by a small model servomotor. These actuators include an internal position control loop, which feedback comes from a potentiometer joined to the axle of the motor. The reference for this regulator is proportioned through an external PPM (Clark, 2002) signal. Thus, it's approached the problem of generating 20 PPM signals by means of a microcontroller. Each of the signals will be used to encode the position of each one of the servomotors.

The PPM signal is a digital pulse modulation wave that is used to encode the position. This is a periodic signal of frequency $F = 50Hz \equiv T = 20ms$, where the position is encoded using the stretch comprised between $850\mu s$ corresponding to -85° and $2550\mu s$ corresponding to $+85^{\circ}$, as it's observed in the figure 2. In this way, the minimum pulse value in the beginning of each period has a duration of $850\mu s$. In the opposite, the maximum duration will be $2550\mu s$, which corresponds to the $12'75\%$ of the period of the signal. This part of the period contain the position information, the rest of this remain on low level, and is empty of information.

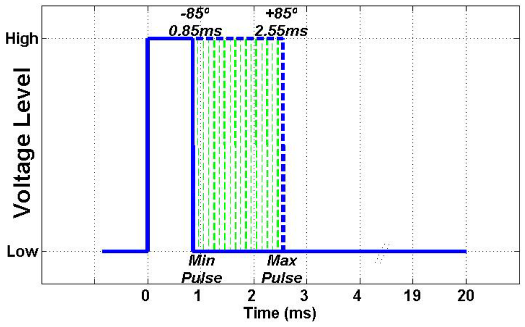


Fig. 2. PPM Pulse.

As it has been explained in the section 1, the 20 PPM signals are software generated. For this, it is necessary to use a clock/counter available in the microcontroller and general purpose input/output ports (GPIO). As brief review (Chakravarthy, 2006)(Mai, 2007)(Li, 2007), it must be mentioned currently, many advanced microcontrollers have PWM signal generators specific units, with these the PPM signal could be generated without difficulty and freeing the main CPU of this task. But, the main inconvenient is that due to the number of signals to generate, there isn't any commercial device able to generate the 20 required signals simultaneously, maintaining the same resources and computation capability that ARM7 and with the reduced enough to be integrated in the board control of the robot either. In many cases, designers of robots have opted for the generation of PPM signals using on board embedded FPGA. Since these devices have a large number of digital outputs, and are able to perform multiple hardware operations in parallel, this are very useful to obtain signals PWM-PPM-type with very good resolution ($250ns$). In this case, due to the requirements of the application, and available resources, the solution is addressed by software dealing with a typical real time software scenario in terms of time restrictions and accuracy controller output. In the same way, the ARM provides sufficient capabilities and resources, to be able to approach the resolution by means of software, thus, it allows evaluating different strategies of real time PPM signal generation, that make possible the 20 DoF control.

2.1. Temporal restrictions imposed by the control

The duration of the pulse width precisely encodes the position of the servomotors, thus has to fulfill the requirements imposed by the task that is going to be realized by the actuators and for themselves. Since the actuators manage the positioning of the joints, and with this one the location of the robot extremities in a certain space position, the requirements that are going to be impose to the signal will be those that do the end position of the robot, proper and stable. Experimentally has been proved that to obtain the accurate positioning of the robot extremities, the position error in each of the articulations must be smaller than $0'5^{\circ}$. This determines the precision timing that the software program that manages the PPM signal generation has to be capable of guarantee. Thus, since the equation (1) is deduced that to obtain this angular resolution in the axel servo positioning, at least a $5\mu s$ of timing resolution is required in the PPM generation in each of the signals.

$$S = \frac{T_{P\max} - T_{P\min}}{Rank} = \frac{1700\mu s}{170^{\circ}} = 10 \, {}^{\mu s}\!/_{\circ} \; ; \qquad (1)$$

On the one hand, a wrong timing when generating the signal would produce a displacement in the time of falling edge of the PPM signal, which would cause a displacement in the positioning of the axel of the servomotor. While on the other hand, to reduce resolution in timing management, relaxing timing restrictions which decrease computer efforts, would cause uncertainty in the generation of the signals. This would cause a vibration in the axel of the servomotor, and an improper positioning of it.

With all that, it is necessary approach a task with high timing requirements that guarantee the correct timing in the signals generation, but with the most reduced resources utilization, CPU and Energy demand.

Additionally, is important to attend the proper synchronization of the signals. Since this is a control system, and an incorrect synchronization of this signals, could affect the correct movement of the robot. This is due to the position of the robot extremities are calculated since the assumption that all the actuators change their position in synchronized mode. Otherwise, the trajectories will not be tracked appropriately.

3. PPM-PWM GENERATION.

In the generation of the group of signals, have been tested several approaches, these will resolve the problem with different difficult grades and achieve result that satisfy the time requirements needed for the control and minimize the use of system resources.

Some of the solutions are executed on the embedded system without intermediation of operating system. At start the execution, is realized a system initializing in which are configured the interrupt managers, peripherals and devices. Concretely, the Timer T1 interruption is configured in order to execute one interruption per every $20ms$. And, so it could be used in PPM signal generation. This interruption will execute with the maximum priority of the system, in this way are guaranteed that will not be interrupted and that will be fulfilled the timed stipulated in the implementation of the program. Once finished the initial configuration, is started the execution of the main loop, in which there will be realized the rest of activities of the system, as example: trajectory planning, behavior decisions, communications, and others. All of them coordinate their execution according to a cyclic scheduler.

In the following sections there will be described each of the proposed solutions.

3.1. Periodic interrupt with exhaustive polling

The initial solution is based on the system described in the previous paragraph. Therefore the generation of the PPM signals is performed inside the Timer1 interruption manager. The starter supposition is that the movements' planner has transformed by means a linear transformation, the position into time of pulse duration (2).

$$T_{PULSE} = T_{P\min} + (pos + 85^{\circ}) \cdot S = 1700 + 10 \cdot pos \quad \mu s \quad (2)$$

Inside the interruption manager, initially, all the signals are established to high level, since at least it is necessary to generate the minimal pulse as is explained in the section 2. Once passed this initial time, inside a loop, is verified in exhaustive way for every signal, if the time since the start of the interrupt until the current moment is superior to the duration of the needed pulse, and in this case, the value of this signal is modified to low level. Passed the time of maximum pulse, the interruption will return.

```
INT_Timer1:
    All_Servo_Signals = 1;
    wait (Minimum_Pulse_Time).
    loop (Until_Maximum_Pulse_Time)
        if (Servo1_Time == pass)
            Servo_Signal1 = 0;
        if (Servo2_Time == pass)
            Servo_Signal2 = 0;
        ...
        if (Servo20_Time == pass)
            Servo_Signal20 = 0;
    end loop;
fin INT;
```

Since the interruption manager disposes the maximum priority, no interruption will be triggered during the execution of this, therefore, the duration of this will be the duration of the maximum pulse. This entails a $13'25\%$ of the processor utilization. On the other hand, in each loop iteration is realized the comparison for each of the signals, as result, is elapsed a considerable time in it. Attending to this, the resolution obtained is not fulfilling in this case the specifications marked.

3.2. Periodic interrupt with sorted list polling

The second approach, tries to improve the first one in the way to obtain a better temporary resolution in the PPM signal generation. For it two basic modifications are introduced.

The main problem in the previous solution is produced by the excessive duration of each one of the loop iterations in which the signals state is updated. This is due to the fact that in each iteration 20 comparisons are computed, one per each of the servomotors. The new proposed, makes a dynamic sorting of the signals according to the duration of the pulse. Thus, in the comparison loop, only is necessary to realize one comparison, the corresponding signal that is nearest to end, stored in a previously sorted vector.

```
Servos_Time_Sorted();
INT_Timer1:
    All_Servo_Signals = 1;
    wait(Minimum_Pulse_Time).
    cont = First_Servo;
    loop (Until_Maximum_Pulse_Time)
        if (Servo_Time[cont] == pass)
            Servo_Signal[cont] = 0;
            cont=Next_Servo;
    end loop;
fin INT;
```

With this modification the temporary resolution, is notably improved. A resolution of $0'09$ degrees is obtained, and in this way, the restrictions imposed by the process are fulfilled. Like main drawback, the utilization of the CPU is increased in $2'8\%$ owed to the sorting before every execution of the interrupt manager.

3.3. *Aperiodic interrupt and distributed PPM*

Analyzing the main disadvantages of the previous solutions is deduced that these are:

- The overload of computation produced by the iterative consultation during the generation the pulse length of $2'55ms$
- The energy consumption peak, due to that the servomotors synchronize its main energy consumption with the rising edge of the PPM control signal, and these signals are synchronized with the beginning of the control period.

Since this, in a period, there are cycles with high discharge density, and others practically empty. With that, the strong currents reduce sharply the voltage in the batteries, and due to this is forced the reset of the microcontroller, like can be seen in the Figure 3. This also produces the reduction of the life cycle of the batteries due to the intensive use of them.

To solve both problems at time, has been implemented a solution in which each one of the PPM signals appears displaced $850\mu s$ seconds in the period from the start of the previous PPM signal. Thus, the consumption of energy it is distributed along the whole period of control. This displacement of coordination in the period of control will be taken into consideration in the moment of trajectories generation.

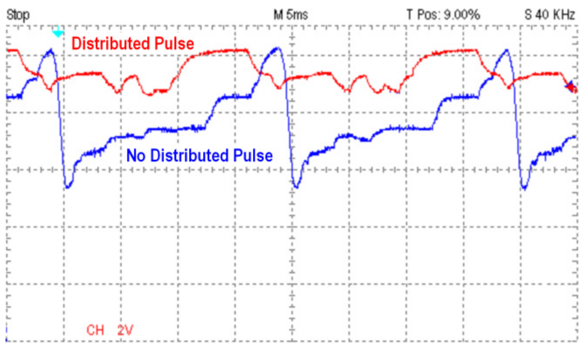


Fig. 3. Battery voltage level.

Like in previous solutions for signals generation, in this case those signals are generated using Timer 1 interrupt manager too, but in this case, the time between one interrupt and the next is variable. Concretely, an interruption is generated whenever it is necessary modify the state of someone of the signals. Assuming that, a total of 40 interruptions will be generated in each of the control periods, 2 for each of the signals, for the rising and falling edges correspondingly.

Since the time between interruptions is variable depending of the pulse duration, at the beginning of each one of the periods of control, is necessary to execute a routine that calculates the time in which each of the interruptions must take place. In return, this will introduce an increase of the utilization of the CPU of $70\mu s$.

```
Activation_Times_Calculation();
INT_Timer1:
        Servo_Signal[cont] = act_level[cont];
        cont = next_activation;
        Timer1_cnt =
                Time_Until_Next_Activation[cont];
fin INT;
```

This entails a $1'09\%$ of the processor utilization. It must be highlighted, that this is approximately 10 times less. And likewise the resolution is $0'1432$ degrees, good enough for the control.

3.4. *Periodic interrupt with sorted list polling, SORT PaRTiKle*

Another solution has been implemented using the support of a real time embedded operating system PaRTiKle (Masmano, 2005) (Masmano, 2008) (Peiró, 2008) which is a new RTOS designed to be compatible with the POSIX.51 (O.Group, 2003) standard.

PaRTiKle has been designed bearing the following ideas in mind:

- Being as portable, configurable and maintainable as possible.
- Support for multiple execution environments, allowing, thus, to execute the same application code (without any modification) to be executed under different environments (so far): in a bare machine, a Linux regular process and as a hypervisor domain.
- Support for multiple programming languages, currently PaRTiKle supports Ada, C, C++, Java (the current support of this last language is only supported when GCC compiler version 3.4 is used).

Other PaRTiKle's interesting characteristics are:

- Small memory requirements (kernel: 60-70Kb).
- System configurable to embedded systems.
- Programming API standard: POSIX PSE.51, this facilitates the code portability from existing systems.
- Efficient use of the available hardware.

This section describes the most relevant aspects of PaRTiKle's porting to the LPC2000 system (NXP, 2007).

PaRTiKle uses the UART0 as serial terminal connected by ftdi-usb interface (Peacock, 2005) to the development computer serial port this allows the sending/receipt of char streams through the functions `printf/scanf`.

In the timing management The LPC systems includes a real time clock (RTC) and two timers (T0 and T1). The RTC works at a frequency of 32KHz, providing a resolution of $\frac{1}{32}\text{Khz} = 30.5\mu s$ For this reason it's used as a system clock.

While the timer T0 can be used also as counter, and work at the same frequency as the peripheral bus PCLK (Peripheral Bus Clock). This supposes that working at 60MHz, it can obtain a timing resolution of $0.06\mu s$. In this way the Timer 1 (T1) remains free and can be used in other applications

Among the implementation details on PaRTiKle, can be highlight:

In this section is presented the solution 3.2 (Periodic interrupt with sorted list polling) implemented on the SOTR PaRTiKle. For it the application is re-designed using the primitive that the SOTR offers, like: periodic tasking and its communication (synchronization) by software monitors.

Especially the PPM signals are implemented by means of a periodic task which period is 20ms, This thread uses the same mechanism that was explained in section 3.2. Including the Timer 1(T1) utilization like timing reference.

The rest of activities in the system, like the trajectory, movements' generation, and the behaviours planning, are implemented by means of tasks of the SOTR, and software monitors are used for its communication. Concretely a software monitor has been implemented that provides all the mechanisms for the synchronization between the behaviours planning task, and the trajectory and movements generation task.

The conclusions about development on SORT Partikle are discussed below:

In this section are analyzed and presented the conclusions that have been extracted in the application development on the SOTR PaRTiKle and how it concerns the different phases of the development: Design, Programming, Debugging and Maintenance.

Design: One of the main advantages of RTOS using, is that a (POSIX) standard API is in use, which allows to be abstracted of the underlying hardware. Allow fulfilling the application on the basis of tasks, protected objects, and other primitives offered by the OS.

- Tasks control (threads of execution).
- Tasks Synchronization and temporization.
- Resource management

Programming: The Operating system use during the design stage redounds to that during the programming it is not necessary to worry of the details of the hardware, like:

- Initialization and configuration of the hardware.
- Management of devices and interruptions.
- Dynamic memory Management.
- Specific Details of the Compiler and Assembler.

On the other hand, makes free the programmer from the task from implementing the communication and synchronization mechanisms ad-hoc for a specific application, since these have been implemented correctly in SO's level, which also reduces the number of mistakes.

Debugging: In case of mistakes/exceptions of the processor taking place, the operating system takes charge of providing a diagnostic message (exception produced, register state, stack …) that allows determining the mistakes reasons. This is possible since there is provided the program counter (PC) that provides the address of the instruction that produced the failure.

Maintenance: Finally, the use of a homogeneous (POSIX) API during the development facilitates the modifications and changes along the useful life of the system.

4. RESULTS

The three previous solutions proposed (sections 3.1, 3.2, 3.3), have been evaluated by means of simulation tools that are provided by the programming environment Keil™ Development Suit for ARM®, and all of them performing measurements directly in the system using the general purpose in/out digital ports. And finally, on the biped robot for which the real time PPM generation has been developed. Since that, the following results have been obtained:

For the two firsts solutions, the system invests a fixed time for the generation of the control signals. This comes given by the duration of the maximum pulse. In the equation (3) can be estimated the computational cost that this task carries out.

$$U = \frac{C}{T} = \frac{2650\mu s}{20ms} = 13'25\% \qquad (3)$$

The second solution, introduces an extra charge of calculation owed to the array signal sorting. In the equation (4) can be estimated the additional calculation that must execute the processor.

$$U_O = \frac{C}{T} = \frac{560\mu s}{20ms} = 2'8\% \qquad (4)$$

The maximum error in the servomotors axel positioning produced by the timing resolution, is calculated in the equation (5) for the first case, and in the equation (6) for the second proposed.

$$R = 19\mu s = 19\mu s \cdot \frac{170°}{1700\mu s} = 1'9° \qquad (5)$$

$$R = 0'9\mu s = 0'9\mu s \cdot \frac{170°}{1700\mu s} = 0'09° \qquad (6)$$

It is necessary to emphasize that these results have been obtained working on clock frequency of 60MHz, and since in both proposed the resolution depends directly on the capacity of execute the loop iterations, is influenced directly for the microcontroller processing speed.

For the third solution, the computational cost is distributed along the whole period of control, is possible to see this in the figure 4. Thus, in the beginning of every control periods is executed a routine that calculates the timing between interrupts generation for the whole period. The system utilization, produced by the PPM signals generation (9), can be calculated like the sum of all these small interruption times (7), and the time corresponding to the initial calculation (8).

$$U_{INT} = \frac{C}{T} = \frac{3'7\mu s \cdot 2 \cdot 20}{20ms} = \frac{148\mu s}{20ms} = 0'74\% \qquad (7)$$

$$U_{C.T.} = \frac{C}{T} = \frac{70\mu s}{20ms} = 0'35\% \qquad (8)$$

$$U_{PPM} = U_{INT} + U_{C.T.} = 1'09\% \qquad (9)$$

The error in the positioning of the system depends on two sources of delay. On one hand the resolution of the timer T1, due to on this, is implemented the interruption (10), and on the other hand, the attended latency time of the interruption. (11) In this way the maximum error will be produced by the sum of both previous ones is 0'1432º (12).

$$R_{T1} = 1\mu s = 1\mu s \cdot \frac{170^\circ}{1700\mu s} = 0'1^\circ \qquad (10)$$

$$R_{L.INT} = 432ns = 432ns \cdot \frac{170^\circ}{1700\mu s} = 0'0432^\circ \qquad (11)$$

$$R_{TOTAL} = R_{T1} + R_{L.INT} = 0'1432^\circ \qquad (12)$$

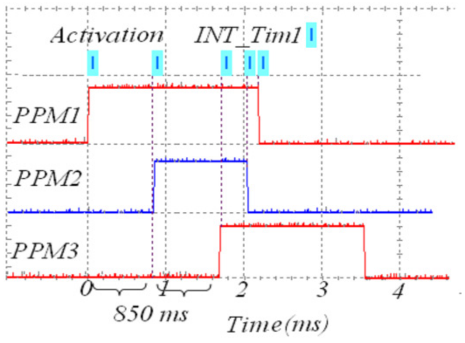


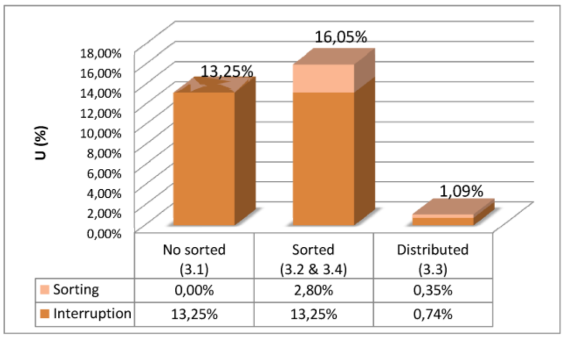
Fig. 4. Three Distributed PPM signals, and Timer1 interruption activations.

It must be highlighted, that, the distribution of the control pulses along the period reduces current peaks avoiding an excessive wear of the batteries and the sharp decreases of tension, like is possible to estimate in the Figure 3. Additionally this avoids the undesired resets of the microprocessor due to momentary low battery level.

For the fourth solution, showed in the section 3.4, (Periodic interrupt with sorted list polling, SORT PaRTiKle), the operating system overflow have no influence in the servomotors control. Since this, the results obtained for the solution that was explained in the section 3.2, are the same in respect to the PPM generation. Thus, the resolution obtained in that, also has no variation respect to the results obtained previously.

It is necessary to emphasize that the utilization of the operating system improves the possibilities and facilities of development and implementation. These have been commented in detail in the section 3.4.

A comparison between the utilization of the processor of the three first solutions (sections 3.1, 3.2, 3.3), can be observed in the figure 5.



	No sorted (3.1)	Sorted (3.2 & 3.4)	Distributed (3.3)
■ Sorting	0,00%	2,80%	0,35%
■ Interruption	13,25%	13,25%	0,74%

Fig. 5. Processor utilization (U) in approaches.

4.1. *Implementation conclusions*

Four real time solutions have been implemented that allow the generation of the PPM control signals, necessary to manage the 20 actuators that are included in the joints of the humanoid robot microBIRO.

The first solution (3.1), is able to generate the control signals, but no manages these with sufficient precision for the task requirements. Though, it is a valid method for many other tasks with less strict requirements respect to the servomotors positioning error.

In the case of the solutions in previous section (3.2, 3.3, and 3.4), the required restrictions are fulfilled for attain the correct placement of the robot extremities with the needed precision. These three solve with different complexity degrees the same problem of low level control through PPM signal generation.

Attending to the resources utilization in each of them we can find that:

- The second (3.2), provides simplicity in the implementation, but a highly computational cost.
- The third (3.3), reduces in much the computational cost, increasing in complexity of implementation.

The fourth (3.4), is an evolution of the second one, incorporating the advantages of using a real time operating system, these will be commented in next section.

5. CONCLUSION

In this paper, four real-time solutions have been implemented using the available system resources to allow the generation of the 20 PPM control signals, to manage the 20 DoF of MicroBIRO, a humanoid robot.

As conclusion, must be outlined that there has been achieved an implementation (3.3) that, being executed on the embedded system without operating system, provides good results for the servo-motor control. This could be concluded since the approach obtains a resolution good enough for the control task approached, minimizing at the same time system resources utilization. Notably, the relevance of these results in autonomous systems that dispose limited resources.On one hand, reducing the CPU utilization, allows that other activities can be executed with minimal influence over the control task. On the other hand reducing and managing energy consumption allows the system to adapt the control requirements to the source of energy. This is very important question when the energy source is a battery that must be long as much as possible to provide autonomy.

In the last solution (3.4) the utilization of the R.T.O.S PaRTiKle, provides the system a wide range of possibilities for the implementation of high-level control loops using programming abstractions like threads, mutex, timing management and many others. All this, maintaining the previous benefits.

Finally must be highlighting that this last solution is suitable to be used in a wide range of embedded systems due to the limited resources required for the R.T.O.S., which is specially interesting in micro humanoid robot.

REFERENCES

M. Albero, V. Nicolau, F. Blanes, J. Simó. (2007). *microBiro: UN ROBOT BÍPEDO PARA LA ENSEÑANZA Y LA INVESTIGACIÓN.* Instituto de Informática y Automática Industrial AI2, Universidad Politécnica de Valencia.

ARM Limited. (2001). *Advanced RISC Machines, ARM7TDMI-S (rev r4p3) Technical Reference manual.*

Chakravarthy, N. Jizhong Xiao. (2006). *FPGA-based Control System for Miniature Robots,* Intelligent Robots and Systems, IEEE/RSJ International Conference on.

Dennis Clark, Michael Owings. (2002). *Building Robot Drive Trains,* McGraw-Hill Professional.

Li, T.-H.S. Yu-Te Su Cheng-Hsiang Kuo Chi-Yang Chen Chia-Ling Hsu Ming-Feng Lu. (2007). *Stair-Climbing Control of Humanoid Robot using Force and Accelerometer Sensors,* SICE, 2007 Annual Conference.

Jingeng Mai, Qining Wang, and Long Wang. (2007) *FPGA-Based Gait Control System for Passive Bipedal Robot.* Humanoid Robots, 2007 7th IEEE-RAS International Conference on.

M. Masmano; I. Ripoll, A. Crespo. (2005) *An overview of the XtratuM nanokernel*, Workshop on Operating Systems Platforms for Embedded Real-Time applications.

M. Masmano, I. Ripoll, A. Crespo. (2008). *PaRTiKle OS User Manual.* Real-Time Systems Group, Universidad Politécnica de Valencia.

NXP. (2007). *LPC2000 Application Notes: Boot sequence, Interrupts, Spurious Interrupts*. NXP, founded by Philips.

NXP. (2008). *LPC2131/2/4/6/8 User manual.* NXP, founded by Philips.

The Open Group. (2003). *The core of the Single UNIX Specification, Version 3*, ISO/IEC 9945.

Craig Peacock. (2005). *USB with the simplicity of RS-232.*

S. Peiro, M. Masmano, I. Ripoll, and A. Crespo. (2008). *PaRTiKle OS, a replacement of the RTLinux core*. Real-Time Systems Group, Universidad Politécnica de Valencia.

David Seal. (2001) *The ARM Architecture Reference Manual.* 2nd Edition, Addison-Wesley Longman Publishing Co..