An Embedded Real-Time System for Autonomous Flight Control

Mauro Marinoni*, Tullio Facchinetti*, Giorgio Buttazzo* and Gianluca Franchino*
*University of Pavia, Pavia, Italy

Abstract— **Unmanned Autonomous Vehicles (UAV) represent an attractive solution for those monitoring applications in hazardous environments, where direct human intervention is difficult or impossible to achieve. Small autonomous aircrafts represents a convenient possibility for monitoring large areas, for example for detecting fires and following their evolution. Controlling such systems, however, is very challenging due to the limited resources available onboard and to the high number of constraints, including weight, space, time, energy, and cost.**

This paper presents a real-time flight control system for an autonomous aircraft model. The control system runs on an embedded hardware platform that includes the microcontroller, the sensors for navigation and environment monitoring, the power management circuit and the wireless communication system. The applications is implemented on top of a real-time operating system suitable for embedded microprocessors, which manages a set of concurrent activities dedicated to sensory acquisition, flight control, communication, and resource management (including energy).

I. Introduction

In the last years, electronic embedded devices (e.g., cameras, mobile phones, portable music players, navigation systems) have been growing exponentially, becoming more and more complex in terms of system functionality. Features are added at an unprecedented speed by carefully programming general-purpose embedded microcontrollers and the application code typically consists of several thousands lines of code involving a large number of parallel activities. Embedded systems based on microcontrollers are also installed on airplanes, cars, factory implants, where it is common to find tens or hundreds of inter-connected devices, each dedicated to specific tasks [1].

Developing small embedded system presents several challenges, due to size and weight constraints [2]. Size reduction implies the use of small microprocessors with limited memory and processing power, which represents a severe constraint for the software, which must be carefully designed to efficiently exploit the available resources. Moreover, if the system is powered by batteries, energy management policies must be applied at different levels of the architecture to reduce power consumption and prolong the system lifetime as much as possible [3].

When embedded control systems tightly interact with the environment, as for a flight control system, the tasks running on the microcontroller are subject to stringent timing constraints, which must be enforced by the operating system to guarantee stability and achieve a desired level of performance. Real-time constraints are typically specified by deadlines, activation periods, response times, input-output delays, and jitter requirements. Their specific values are dictated by the dynamics of the system-environment interactions.

The operating system has a crucial role in guaranteeing the application performance, because the timing behavior of the application strictly depends on task scheduling, interrupt handling, synchronization protocols, and resource management algorithms. The simplest way for enforcing timing constraints to the application is through a static schedule implemented by a cyclic executive. However, this solution is not flexible to changes and is very fragile under overload conditions. Hence, a priority-based kernel is more suitable to support dynamic control applications with variable computational requirements [4].

There are several real-time operating systems available in the market, both free and proprietary, but very few of them are suitable for small embedded microcontrollers with limited processing resources. VxWorks [5] and QNX Neutrino [6] are two examples of commercial kernels commonly used in real-time control applications. Another real-time kernel widely used in the industry is MicroC/OS [7], which is a preemptive real-time kernel written in C, available for more than twenty different microprocessors. Among the open source kernels related to Linux, RTLinux [8] and RTAI [9] use a small real-time executive as a base and execute Linux as a thread in this executive, whereas Linux-RK [10] directly modifies the Linux internals. Most of these kernels, however, do not implement the state-of-the-art features derived from the real-time scientific literature, and are not easy to modify. More flexible kernels are MarteOS [11] (written in ADA), allowing the user to specify the scheduler at the application level, and Shark [12]) (written in C), which is a modular kernel handling tasks with different criticality and allowing the user to select the algorithms for task scheduling and shared resource management. However, all the kernels mentioned above are designed for medium size applications and are not suited for small microcontrollers. On the other hand, very small kernels have limited real-time features. For example, TinyOS [13] is widely used in sensor networks to support the tasks running on the Motes, but it uses a FIFO queue for task scheduling and it cannot handle timing constraints.

This paper describes a low-cost embedded control system for a small Unmanned Autonomous Vehicle for surveillance and fire prevention. Several constraints have been taken into account in the design, including light weight, small size,

This work has been partially supported by the Italian Ministry of University Research under contract 2004095094 (COFIN 2004).

low cost, low energy consumption, small memory usage, and sufficient computational power for performing navigation and sensory processing in real-time. The weight constraint is particularly relevant, since the overall payload of the aircraft is only 120 grams. Other autopilot systems available on the market have been considered for the purpose. For example, the *MP2028*^{g} from Micropilot [14] has a weight of 28 grams only, but it is too expensive and its software is not open to modify the control algorithm and integrate other sensors.

The solution presented in this paper performs both autonomous navigation and mission related activities, and it is flexible enough to allow easy interchange of sensors and for adapting the vehicle to different monitoring applications. The aircraft model has two flight modes: manual and autonomous. While in manual mode, a human operator can control the flight speed and direction using a remote control from the ground station, which is useful during take off and landing phases. While in autonomous flight mode, the onboard controller is programmed to follow a desired trajectory (specified through a number of set points) without human intervention.

To predictably manage concurrent activities with periodic and aperiodic activations and explicit timing constraints, the application is implemented on the Erika Enterprise real-time kernel [15]. It is an advanced real-time operating system available on the market that has been specifically designed for minimal embedded systems with limited onboard resources.

The paper is organized as follows: Section II introduces the onboard control system, including the microcontroller, sensors, actuators, wireless communication system, and the real-time kernel; Section III describes the control application, from data capturing to actuators control and related real-time issues; Section IV discusses the power-aware issues, including processor and peripheral device power saving strategies; and, finally, Section V states our conclusions.

II. System description

The onboard system architecture is illustrated in Figure 1. It performs sensory acquisition and uses such data to trigger the motors for speed and direction control during autonomous flight, control the power consumption of the external devices, and send the relevant sensor information to the ground station.

A. Onboard controller

The onboard processing unit is a Microchip's *dsPIC* 30F6014 microcontroller [16], which integrates the control features of a Micro-Controller Unit (MCU) with the processing and throughput capabilities of a Digital Signal Processor (DSP). It is a 16-bit microcontroller with a maximum processing power of 30 MIPS. The model selected for prototyping includes a program memory space of 144 KBytes, a data memory space of 8 KBytes, and a non-volatile data EEPROM of 4 KBytes. In terms of peripherals the chip supplies Capture/compare/PWM functionality, 12-bits Analog-to-Digital Converters (A/D) with 100 Ksps conversion rate and up to 16 input channels. Connectivity is provided through a full range of channels: I2C, SPI, CANbus, USART and Data Converter Interface (DCI), which supports common audio Codec protocols as I2S and AC'97.

The MCU power management system allows two reduced power modes, *idle* and *sleep*. The *idle* mode disables the CPU, but keeps the system clock source operative. Therefore, the peripherals continue to operate, even though they can optionally be disabled. In the *sleep* mode, both the CPU and the system clock source are disabled, as well as every peripherals that need the system clock to work. The *sleep* mode consumes less power than the idle mode, but, while the former has a latency of 10-130 μs to resume the system, the latter has no wake-up delay. The microcontroller supports a method to disable a peripheral device which consists in stopping all the clock sources supplied to that module. When this method is adopted, the device has a minimum power consumption. This method requires 1 instruction cycle delay for both disabling and enabling the peripheral device.

B. The real-time kernel

The control application is developed on top of the ERIKA (Embedded Real tIme Kernel Architecture) Enterprise real-time kernel [15]. The kernel is organized in a modular fashion and it is fully configurable both in terms of services and kernel objects (tasks, resources, and events). It allows the user to include only those services strictly required by the application, thus achieving a minimal memory footprint of 2 Kbytes, up to more complete configurations. It is available for a wide variety of 8, 16, and 32 bit CPUs and supports advanced scheduling mechanisms, such as Rate Monotonic, Earliest Deadline First [17] and resource reservations [4]. The kernel modular design allows reusing the software modules in different applications, speeding up the development of new projects or the upgrade of existing projects to more powerful architectures, and simplifying the maintenance.

The kernel consists of two layers: the Hardware Abstraction Layer (HAL) and the Kernel Layer. The HAL represents the very low level kernel layer; therefore, different HALs are required for different processors (notice that the Kernel Layer does not change when the ERIKA system is ported on different platforms) The HAL contains the hardware dependent code to manage the context switches and to handle the interrupt requests. It supports mono-stack and multi-stack models, where the former is used in systems with severe memory constraints: the system designer can choose the best solution depending on the application. The Kernel Layer is composed by a set of modules for task management and real-time scheduling policies. Fixed priority with preemption threshold and Earliest Deadline First (EDF) with preemption threshold are currently supported by the kernel. Both use the Stack Resource Policy (SRP) [18] to share resources between threads and to share the system stack among the threads while preserving time predictability.

C. Sensors and actuators

The microcontroller is connected to several sensors and actuators to perform autonomous flight and environment monitoring. Figure 1 shows the sensors and actuators managed by the microcontroller. The onboard configuration includes:

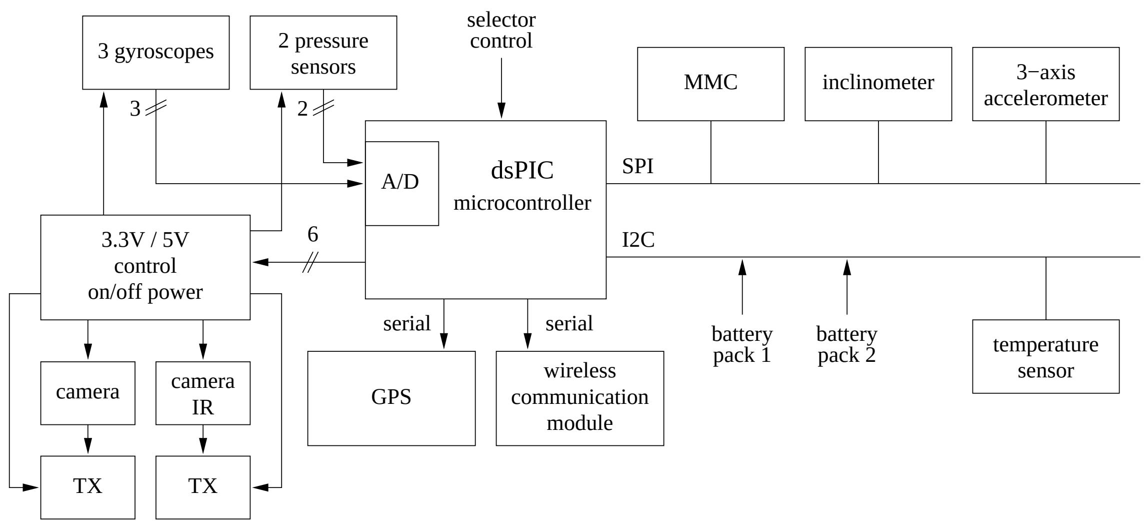

Fig. 1. Onboard system architecture.

- 3 gyroscopes, one for each axis;
- 2 pressure sensors, which are needed to evaluate the pressure at the ground level and the flight speed through the difference between the pressures detected at aircraft head and tail;
- one inclinometer;
- an MMC (MultiMedia Card) memory module, used to store the data logging during the flight;
- a digital 3-axis accelerometer;
- a temperature sensor;
- a GPS module, to determine the absolute aircraft position;
- a wireless communication module, to exchange information with the ground station;
- a video camera, for monitoring the environment and facilitating the manual control when the aircraft is not visible;
- an infrared camera, for increasing sensitivity in fire detection.

The microcontroller integrates the bus to communicate with the sensors. Two different busses, SPI (Serial Peripheral Interface) and I^2C (Inter-Integrated Circuit), together with the serial line, are used to connect sensors with digital I/O, depending on the digital communication technology available on each specific sensor. Since the gyroscopes and the pressure sensors are only supplied with analog output, they are connected to the microcontroller's A/D converters, and their values are sampled at the frequency of 80 Hz.

The microcontroller monitors the power consumption through a custom power control board, which allows to switch on/off the cameras for saving energy.

D. Wireless communication system

The wireless communication system is split into different components: the data communication module, the receiver for manual control, and two independent channels for video streaming.

The data communication module, connected and controlled by the microcontroller (see Figure 1), allows the aircraft to exchange information with the ground station. While the information received from the ground station consists of trajectory set-points only, the data sent to the ground station includes all the navigation data that allows a human operator to monitor the aircraft behavior, and observe specific application parameters (e.g., pressure, humidity, wind speed, and temperature).

The current onboard setup uses two independent video channels for the normal and infrared cameras, since they include an integrated radio-modem, so that the wireless communication board is dedicated to sensory data.

The receiver for manual control is the regular radio device used for normal manual operations. In standard aircraft models, the receiver is directly connected with the motors controlling the throttle and the flaps. To support the autonomous flight, this scheme has been slightly modified, resulting in the circuit illustrated in Figure 2. The new scheme introduces a custom circuit that intercepts the signals coming from the receiver and the motor control signal generated by the microcontroller for autonomously controlling the flight. A dedicated channel from the receiver is used to select the manual/autonomous navigation mode.

The solution adopted for manual/autonomous control completely separates the manual control system from the microcontroller. Therefore, problems that might occur in the microcontroller (hardware and software failures) do not affect the manual control.

III. THE CONTROL APPLICATION

The control application for autonomous flight running on the microcontroller is illustrated in Figure 3. It consists of a closed loop where the information captured by onboard sensors is used to control the aircraft direction and orientation in order to follow the desired trajectory. Each block in Figure 3 is dedicated to a specific task.

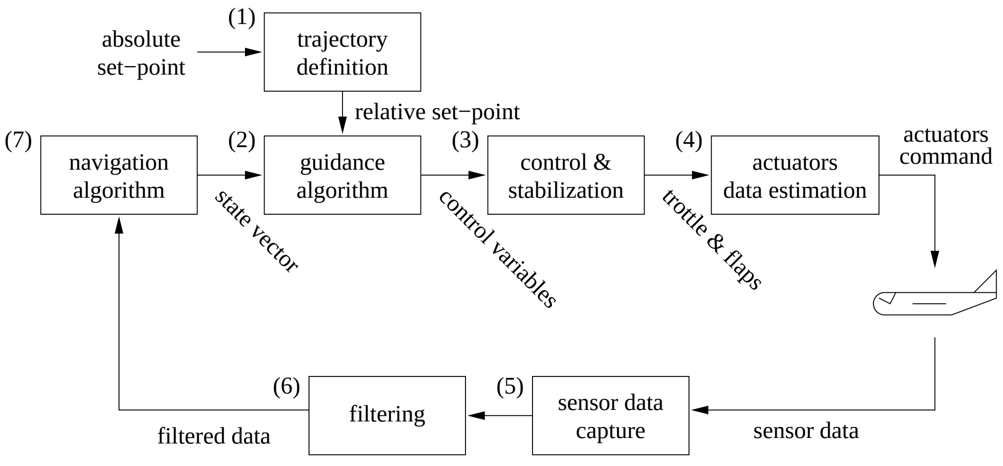

Fig. 3. Block diagram of the control application.

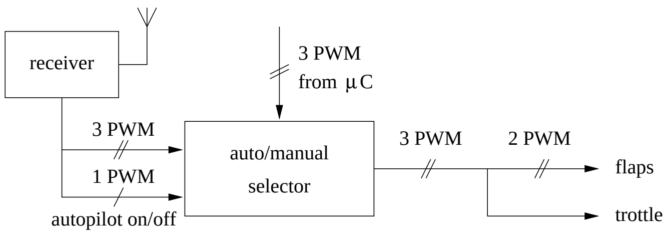

Fig. 2. Auto/manual selection sub-system.

Block (1) implements the algorithms for defining the reference trajectory set-points, which can be set before taking off and changed during the flight. The set-points can be either absolute or relative to the current aircraft local axis. Since the control algorithm works with local coordinates, in the former case the way-point coordinates have to be converted into local coordinates.

The guidance algorithm in Block (2) defines the command variables, including speed, height and direction, suitable to follow the given trajectory. Such variables are used in Block (3) for stabilizing the aircraft and controlling its direction/speed through a set of PID controllers. Block (3) returns the output values to control the flaps and the throttle to approach the next set-point. Then, Block (4) converts such values into the correct PWM signals for driving the actuators (details on motors control are illustrated in Section III-B).

During autonomous flight, there are several sensors used to monitor the aircraft position and direction. Block (5) is dedicated to sensory data capture through a set of periodic real-time tasks, whose period is related to the sensor sampling rate and, in some cases, to energy considerations.

In Block (6), sensory data are filtered to remove the noise affecting the measurements. This operation is particularly important for those signals that need to be integrated to derive the parameters of interest, like those produced by gyroscopes and accelerometers (see Section III-A), since the integration accumulates the errors, causing a drift on the estimated location.

	Absolute	Relative (80 Hz)
Linear	GPS (1 HZ)	Accelerometer
Rotational	Inclinometer (6 Hz)	Gyroscopes

TABLE I
NAVIGATION SENSORS CHARACTERIZATION.

Finally, Block (7) implements a set of Kalman filters to determine the state vector components as a function of the sampled sensory data, which include linear and angular positions, speeds, and accelerations.

A. Sensor data capture

Most of the sensors depicted in Figure 1 are used to estimate the aircraft position and orientation, for supporting the autonomous navigation and allowing the ground station to track the aircraft motion. The UAV has six degrees of freedom, 3 linear and 3 rotational, that must be estimated to guarantee a correct navigation. The final goal is to estimate the absolute amount of such values, with a sufficiently high frequency (80 Hz) to allow the correct trajectory control.

While the GPS and the accelerometers are used to keep track of the linear motion, the inclinometer and gyroscopes determine the aircraft orientation (Table III-A). The GPS and the inclinometer return absolute values for position and orientation, but they work at a frequency (1 Hz and 6 Hz, respectively) that is too low for controlling the aircraft. On the other hand, accelerometers and gyroscopes are fast enough for the control application, but they can only be used for relative measurements (Table III-A). Moreover, their values have to be integrated to get the desired parameter: accelerometers returns accelerations that have to be integrated twice to get the position; gyroscopes provide angular velocities that have to be integrated to get the angles. Since sensors introduce a measurement error, the integration of the sampled values accumulates such errors, resulting in an unacceptable drift on the estimations. The solution is to control the aircraft using the high frequency values provided by accelerometers and gyroscopes, opportunely filtered, and exploit the low frequency

Task	Period [ms]	Variable
GPS	1000.0	Yes
Accelerometer	12.5	No
Inclinometer	200.0	Yes
Gyroscopes	12.5	No
Control	12.5	No
Temperature	1000.0	Yes
Power	1000.0	Yes
Communication	100.0	Yes

TABLE II
ACTIVATION PERIODS OF THE APPLICATION TASKS.

values (from GPS and inclinometer) to periodically reset the drift on relative measurements [19].

Finally, the system also includes other sensors for mission specific purposes. Since the main goal of the system is to detect and monitor fires, the aircraft is equipped with an infrared camera to detect fire sources in forests and woods, and a temperature sensor to measure the air temperature in specific locations in case of extended fires.

B. Actuators control

The actuators consist of two servomotors for the flaps and a DC motor for the throttle. All the motors are controlled through Pulse Width Modulation (PWM) signals. While for servomotors the PWM signals define the motor shaft angular position, the throttle PWM signal controls the motor speed. During manual operations the PWM signals are generated by the receiver based on the commands received from the remote control. While the aircraft is flying autonomously, the signals are generated by dedicated output ports on the microcontroller. The selection between the two PWM signal sources is performed by the custom board showed in Figure 2 and described in Section II-D.

C. Real-time issues

The control application consists of a set of periodic real-time tasks interacting through shared memory buffers protected by mutually exclusive semaphores. A simplified version of the Stack Resource [18] available in the Erika Enterprise kernel is used to access the critical sections in a predictable fashion, preventing unbounded priority inversion [20] and allowing an off-line guarantee of the application.

Application tasks are periodically activated by the kernel to perform sensory acquisition and control. Tasks periods are shown in Table III-C and defined based on the specific sensor feature and on control considerations. Note that some periods are fixed and others may change during the flight depending on specific situations. For example, the sampling rate of the "fast" navigation sensors is related to the one of the control task, which is fixed and defined by the sampling frequency used in the control algorithm. On the other hand, the periods of remaining tasks have no such a constraint and can be changed to balance the microcontroller load (and therefore its speed and power absorption, see Section IV-A) with the accuracy of the sensed information.

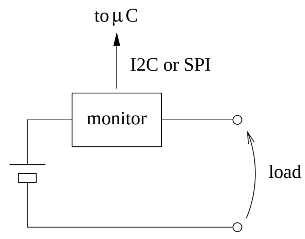

Fig. 4. Power consumption monitoring system.

IV. POWER MANAGEMENT

The power management strategy is performed by the microcontroller in different ways, depending on the energy-management features available on the peripherals. Hardware and software components have been integrated to meet several goals: onboard sensory data processing, real-time computation, and communication features, while achieving low power consumption. To achieve significant energy saving, power-aware strategies are adopted within every system module and are coordinated at the operating system level [21].

All the aircraft onboard systems, including motors, sensors and embedded control board, are powered by batteries. There are 2 different battery packs. All the devices required for manually controlled operation are grouped and powered by one battery pack; they include the receiver, the manual/auto selection custom board, the throttle motor and flaps servomotors. The second battery pack feeds sensors and control board.

The battery charge level is monitored by the microcontroller to adapt the energy management strategy. Figure 4 shows the custom circuit designed to monitor the charge level for each battery pack. It is based on the Texas Instruments's *bq27200* chip, expressly designed for monitoring Li-Ion and Li-Pol batteries.

A. Dynamic voltage scheduling

A widely adopted method to obtain power-aware systems is Dynamic Voltage Scheduling (DVS) [22]. In DVS techniques, the processor voltage supply can be decreased to save energy, since the power absorbtion depends from the third power of the input voltage supply level. However, decreasing the supply voltage also limits the maximum clock frequency, hence the processor speed. Therefore, proper strategies must be adopted to minimize energy consumption under real-time constraints.

Power-aware scheduling algorithms for a discrete set of clock frequencies are implemented on top of the *dsPIC*-based platform, exploiting the DVS features described in Section II-A. The simplest strategy exploits the processor *sleep* state when there is no work to be executed. In this case, the processor is completely disabled, and it is resumed by the system timer for a task activation or after an interrupt generation from a peripheral device. The *sleep* state is also considered as an actual operating mode, and the corresponding zero speed is included in the set of available speed levels.

A simple approach for power-aware real-time scheduling uses a fixed processor clock speed, computed off-line to guarantee system feasibility [22]. However, due to the limited

Module	I_{on}	I_{off}	V_{in}	Ctrl
Accelerometer	750 μA	10 μA	3.3V	Yes
GPS	170 mA	10 μA	3.3V	No
Temperature	28-550 μA	1 μA	3.3V	Yes
TX/RX	35-80 mA	5 μA	3.3V	Yes
MMC	25-60 mA	150 μA	3.3V	Yes
Gyroscope [x3]	6 mA		5.0V	No
Pressure [x2]	10 mA		5.0V	No
TX A/V [x2]	60 mA		11.2V	No
Camera IR	500 mA		5.0V	No
Camera	40 mA		5.0V	No

TABLE III

POWER-RELATED PROPERTIES OF ONBOARD PERIPHERAL DEVICES.

number of available discrete frequencies for the system clock, this solution may cause a waste of energy, because the optimal clock speed must be rounded to the closest higher speed. Better results can be achieved by alternating two discrete clock frequencies, as a PWM signal, to obtain the optimal clock value [23]. Other approaches calculate different speeds for each task and set the system clock to the appropriate value when a context switch occurs [24]. A combination of the two methods is also possible: the off-line calculation of the optimal working speed is integrated by on-line reclamation techniques to further reduce the processor speed and exploit the unused computation time [25], [26].

B. Peripheral I/O

Significant energy saving is achieved by a careful management of peripheral devices. The MCU can turn off each single device in different operating modes, where each device/mode pair is characterized by known energy requirements. Table IV-B shows the power-related properties of peripheral devices. For each device, Table IV-B reports the current absorbtion in both enabled (I_{on}) and disabled (I_{off}) mode, the power supply voltage (V_{in}); the *Ctrl* value indicates whether it is possible to turn the device in power-saving mode through a dedicated digital line directly from the microcontroller. A digital power-control line allows resuming the device faster, without the need of dedicated circuits. Notice that, while the I_{off} value is available for devices with digital power control only, the GPS module represents an exception, due to its peculiar power management, which needs a dedicated (but no digital) line.

The absorption values suggest that, since power consumption ranges from 1 up to 4 orders of magnitudes between enabled and disabled modes, paying attention to the peripheral devices power states may lead to dramatic power saving.

Since a relevant amount of energy is consumed for wireless communication by the radio module, a careful power-aware design of the communication protocol allows considerable energy saving (see the TX/RX field in Table IV-B). There are several issues that can be considered for reducing the energy consumption in a radio module. A source of energy wasting is *idle listening*, which is due to the energy required for listening to the channel possible incoming messages. A second source of wasting is due to *packet collisions*: when two or more messages are sent at the same time by different nodes, packet collisions may be experienced, thus producing corrupted packets; since such packets have to be re-transmitted, extra-energy has to be consumed. Power-aware communication protocols have also to take into account the *overhearing*, occurring when a node picks up packets that are not sent to it, and the *protocol overhead*, due to control packets used in several protocols to manage the communication (e.g., RTS/CTS packets in the IEEE 802.11 [27]). Transmission power control [28], [29] is also useful for saving energy while guaranteeing network connectivity, managing nodes density and allowing spatial reuse of radio channels. Moreover, minimizing transmission power can also indirectly reduce energy consumption by reducing the channel contention and collision between transmitting nodes.

C. Kernel support

The power management system, which is responsible for selecting the most appropriate power states both for the CPU and the peripheral devices, aims at providing a uniform programming interface, while achieving high efficiency to reduce the system overhead. Its main features allow:

- to get the maximum power level allowed by the system for a specific device;
- to get the minimum power level required for the operating system consistency;
- to get the power level actually used by a device;
- to get the power level required by a running, ready or idle task;
- to set the power level for a device while the task is running; the actual device power level will be decided by the power manager based on all task requirements;
- to set/get the power level required by the task while it is not running;

V. CONCLUSIONS AND FUTURE WORK

This paper describes a real-time control system for the autonomous flight of model aircrafts. The description focuses on all the relevant architecture components. The hardware platform includes the microcontroller, communication system, sensors for navigation and environment monitoring, and power management circuits. The control program running on the microcontroller is implemented on top of a real-time kernel to guarantee the timing constraints of application tasks. The software also controls the power consumption of peripheral devices and the microcontroller itself, implements the communication protocol and drives the motors to follow the desired trajectory.

The proposed architecture has been installed on a aircraft model for fire monitoring. The specific application justifies the installation of special sensors for fire detection and monitoring: infrared camera and temperature sensors are added to the sensors required by the navigation system. The platform is flexible enough to allow changing the application specific sensors to adapt the aircraft to other purposes, such as surveillance or different monitoring activities.

REFERENCES

[1] J. Turley, “Embedded processors,” *ExtremeTech, http://www.extremetech.com*, January 2002.

[2] H. Gill, "Challenges for critical embedded systems," in *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*, February 2005, pp. 7–9.

[3] R. Melhem, N. AbouGhazaleh, H. Aydin, and D. Mossé, *Power Management Points in Power-Aware Real-Time Systems*. R. Graybill and R. Melhem (editors), Plenum/Kluwer Publishers, 2002.

[4] G. C. Buttazzo, *Hard Real-time Computing Systems: Predictable Scheduling Algorithms amd Applications*, 2nd ed. Springer, 2005.

[5] "VxWorks Real-Time OS," wind River Corp., URL: http://www.windriver.com/vxworks.

[6] "QNX Neutrino RTOS," qNX Software Systems, URL: http://www.qnx.com.

[7] J. J. Labrosse, *Micro C/OS-II: The Real-Time Kernel*. CMP Books, 2002.

[8] "RTLinux RTOS," fSMLabs Inc., URL: http://www.fsmlabs.com.

[9] E. Bianchi, L. Dozio, G. L. Ghiringhelli, and P. Mantegazza, "Complex control systems, applications of DIAPM-RTAI at DIAPM," in *Realtime Linux Workshop*, 1999.

[10] S. Oikawa and R. Rajkumar, "Portable RK: A portable resource kernel for guaranteed and enforced timing behavior," in *Proc. of the IEEE Real-Time Technology and Applications Symp.*, June 1999.

[11] M. A. Rivas and M. G. Harbour, "MaRTE OS: An ada kernel for real-time embedded applications," in *Proc. of the 6th International Conference on Reliable Software Technologies (Ada-Europe'2001)*, May 2001.

[12] P. Gai and G. Buttazzo, "An open source real-time kernel for control applications," in *Proceedings of the 47th Italian Conference of Factory Automation (ANIPLA 2003)*, November 2003, pp. 21–22.

[13] D. Gay, P. Levis, and D. Culler, "oftware design patterns for tinyos," in *Proc. of the ACM SIGPLAN/SIGBED 2005 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'05)*, June 2005.

[14] "Mp202^g autopilot," micropilot Corp., URL: http://www.micropilot.com.

[15] "ERIKA Enterprise RTOS," evidence Srl, URL: http://www.evidence.eu.com.

[16] *dsPIC30F family reference manual (DS70046C)*, 2004, microchip Technology Inc., URL: http://www.microchip.com.

[17] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, pp. 40–61, 1973.

[18] T. P. Baker, "A stack-based resource allocation policy for real-time processes," in *Proc. of the IEEE Real-Time Systems Symp.*, December 1990, pp. 191–200.

[19] D. Gebre-Egziabher, R. C. Hayward, and J. D. Powell, "A low cost gps/inertial attitude heading reference system (ahrs) for general aviation applications," in *Proceedings of the IEEE Position Location and Navigation Symposium (PLANS)*, April 1998.

[20] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. on Computers*, vol. 39, no. 9, pp. 1175–1185, September 1990.

[21] M. Marinoni, G. Buttazzo, T. Facchinetti, and G. Franchino, "Kernel support for energy management in wireless mobile ad-hoc networks," in *Proc. of the Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT'05)*, July 2005.

[22] P. Pillai and K. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Operating Systems Review (ACM), 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2002, pp. 89–102.

[23] E. Bini, G. C. Buttazzo, and G. Lipari, "Speed modulation in energy-aware real-time systems," in *IEEE Proceedings of the Euromicro Conference on Real-Time Systems*, July 2005.

[24] P. M. Alvarez, E. Levner, and D. Mossé, "Adaptive scheduling server for power-aware real-time tasks," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 2, pp. 284–306, May 2004.

[25] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584–600, May 2004.

[26] W. Kim, D. Shin, H. Yun, J. Kim, and S. Min, "Performance comparision of dynamic voltage scaling algorithms for hard real-time systems," in *Proc. 8th IEEE Real-Time and Embedded Technology and Applications Symp.*, San Jose, California, September 2002, pp. 219–228.

[27] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, iEEE 1999.

[28] J. Heidemann and W. Ye, *Energy Conservation in Sensor Networks at the Link and Network Layers*. Nirupama Bulusu and Sanjay Jha (editors), 2005, technical Report ISI-TR-2004-599, USC/Information Sciences Institute, 2004.

[29] R. Ramanathan and R. Rosales-Hain, "Topology control of multihop wireless networks using transmit power adjustment," in *Proc. of the IEEE Infocom*, March 2000.