A Platform for Real-Time Control Education with LEGO MINDSTORMS

Peter J. Bradley [*] **Juan A. de la Puente** [*] **Juan Zamorano** [*]
Daniel Brosnan [*]

[*] *Universidad Politécnica de Madrid (UPM), Spain.*
e-mail: {pbradley, jzamora, dbrosnan}@datsi.fi.upm.es,
jpuente@dit.upm.es

Abstract: A set of software development tools for building real-time control systems on a simple robotics platform is described in the paper. The tools are being used in a real-time systems course as a basis for student projects. The development platform is a low-cost PC running GNU/Linux, and the target system is LEGO MINDSTORMS NXT, thus keeping the cost of the laboratory low. Real-time control software is developed using a mixed paradigm. Functional code for control algorithms is automatically generated in C from Simulink models. This code is then integrated into a concurrent, real-time software architecture based on a set of components written in Ada. This approach enables the students to take advantage of the high-level, model-oriented features that Simulink offers for designing control algorithms, and the comprehensive support for concurrency and real-time constructs provided by Ada.

Keywords: Control education, real-time systems, embedded systems, LEGO MINDSTORMS, Simulink, robot programming, Ada tasking programs.

1. INTRODUCTION

Acquiring skills in real-time embedded systems programming is a fundamental component in the education of control engineers. Indeed, most control systems are implemented as real-time embedded computer systems. Progress in microprocessor technology has made it possible to embed real-time control systems into all kinds of systems, from industrial equipment to cars, trains, aircraft, and consumer electronics.

Model-based software development, e.g. as implemented by the Matlab®/Simulink®code generation facilities, have made it easier to develop program code for control algorithms, and are now commonplace in all control engineering curricula. Modelling tools allow control engineers to design and test sophisticated control algorithms using a high-level, simulation-based approach, and then generate implementation code in some programming language (C is most common) that can run on an embedded computer platform. However, control algorithms are only a part of a complete embedded software system. Even comparatively simple embedded systems require some kind of operating system support for device input/output, concurrency and real-time control, and user interface, among others. Moreover, the interaction between real-time behaviour and control performance may give rise to unexpected problems when the final system is implemented (Crespo et al., 2006).

Therefore, we believe that a course in real-time programming, including such classical topics as concurrency, real-time scheduling and schedulability analysis, and input/output device programming (see e.g. Burns and Wellings, 2009) is still needed as part of the control curriculum. We have been teaching such a course at UPM for many years, using Ada (ALRM05) as the main programming language. We chose Ada because of the comprehensive support for reliable software engineering and native concurrency available.

An important aspect of real-time programming courses is laboratory work. Finding real-time control applications that are at the same realistic enough and easy to use by students is not a simple task. Recently, LEGO®MINDSTORMS®has gained wide acceptance in different kinds of control-related laboratories (see e.g. Grega and Pilat, 2008; Kim, 2011), due to its versatility and availability as a low-cost robotics platform. In this paper we describe a programming environment for this platform that can be used to support a real-time embedded control systems course, including all critical concurrency and real-time scheduling aspects, while still being compatible with high-level control algorithm design using Matlab and Simulink. The rest of the paper is organised as follows: section 2 introduces the LEGO MINDSTORMS NXT architecture. Section 3 describes how to combine Ada and Simulink to develop real-time systems lab projects. The embedded software development toolchain is described in section 4. Section 5 describes an example of a student project for controlling the speed of a toy vehicle. Finally, some conclusions are drawn and plans for future work are sketched in section 6.

2. LEGO MINDSTORMS NXT

The LEGO MINDSTORMS NXT (from now on NXT) brings together the LEGO world with the digital world. It offers all kinds of LEGO bricks, gears and various other parts together with a wide spectrum of sensors, motors, linear actuators, etc. The brain of the NXT is an embedded system, also called "Intelligent Brick". Figure 1 shows its block diagram.

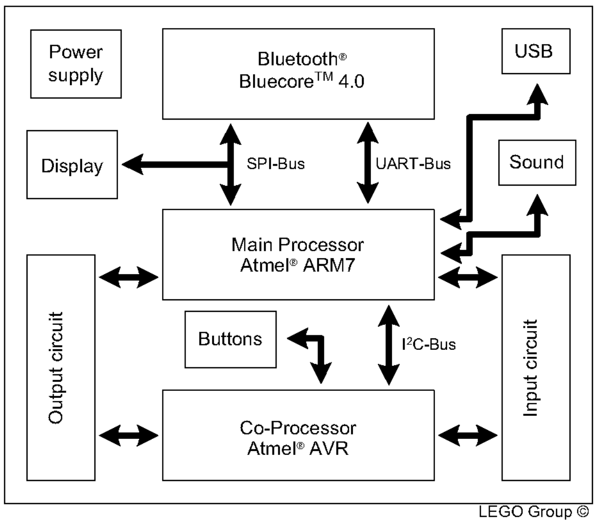

Fig. 1. NXT block diagram.

The Intelligent Brick is made up of a 32-bit ARM main processor (AT91SAM7S256) with 64 KB of RAM and 256 KB Flash memory that runs at 48 MHz. The ARM processor benefits from an 8-bit AVR co-processor (ATmega48) that handles low-level tasks like A/D conversion of the inputs and PWM generation for the output ports. The communication between the two microcontrollers is set up as two memory regions that are updated by both microcontrollers periodically. Notice that this communication period determines to a large extent the behaviour of the NXT.

As for the software part of the Intelligent Brick, LEGO has released the NXT firmware as open source. Availability of the firmware code and hardware schematics has encouraged developers to release their own programming environments. Therefore, the NXT can be programmed from a host computer using languages such as C/C++, Java, Python, Ada, MATLAB, and many more. Also, development environments like LabVIEW, Simulink and Microsoft Robotics Studio offer a model-based approach to program the NXT.

3. ADA, SIMULINK AND THE NXT

The purpose of this paper is to demonstrate how to fully develop the software of an embedded system so that students have a chance to experience the whole development process. This is, a feedback control system design, tuning and programming together with a real-time software architecture that will support the concurrent execution of the controller on the NXT.

As for the first part of the process, the feedback control system, the MATLAB/Simulink environment is used. There are many advantages when using this programming environment in control theory. A given controller can be modelled using the extensive block library and then tuned and tested. Furthermore, Simulink can discretise the continuous controller and generate its C code algorithm for embedded environments.

For the second part of the process, the real-time software architecture, Ada is used instead of C.

Although C is commonly used in embedded systems programming, we consider that is not a good programming language to teach concurrency concepts. C relies on external system calls to a real-time operating system to provide concurrency and real-time features. The low-level nature of these operating system primitives diminishes concurrency concepts. There is also a large list of known C vulnerabilities (see WG23-N0304) that may be critical in high-integrity systems. Finally, C syntax can be confusing and error prone.

On the other hand, the Ada programming language presents ideal features for embedded real-time programming. Concurrency and real-time are part of the language, no external calls are required. These features enable using high-level concurrency abstractions like tasks, protected objects and synchronous message passing that ease programming and avoid common pitfalls. It offers mechanisms to handle interrupts in a simple manner and also provides a clean and efficient way to interact with I/O registers and memory. Also, Ada's strong typing and run-time checking significantly reduces programming errors.

Ada offers mechanisms to interface with other languages. The controller C code generated by Simulink can be included in the Ada software architecture using standard compiler directives and libraries. This is shown in section 5.5.

Predictable behaviour of systems is one of the key goals of real-time development. This property is binding to ensure a hard real-time system meets its deadlines. By enforcing the Ravenscar Profile (Burns et al., 2004) on NXT applications, we can be certain that the outcome of any NXT application is predictable and a schedulability analysis can be performed.

The Ravenscar profile is a collection of restrictions to the Ada tasking model. This profile is restricted to a fixed priority and pre-emptive scheduling, i.e. the run-time scheduler ensures that at any given time, the processor executes the highest priority task of all those tasks that are currently ready to be executed. Also, the Immediate Ceiling Priority Protocol (ICPP) (Sha et al., 1990) is enforced by the Ravenscar profile. This means that when a task locks a resource, its priority is temporarily raised so that no task that may lock the resource can be scheduled. Thus, priority inversion due to task communication is minimized and bounded. More information can be found in ALRM05, Annex D.

4. ENVIRONMENT DESCRIPTION

4.1 Overview

The embedded software for the NXT is developed on a GNU/Linux x86 host computer. The set of tools required and the process to build an NXT application on the host computer is explained in this section.

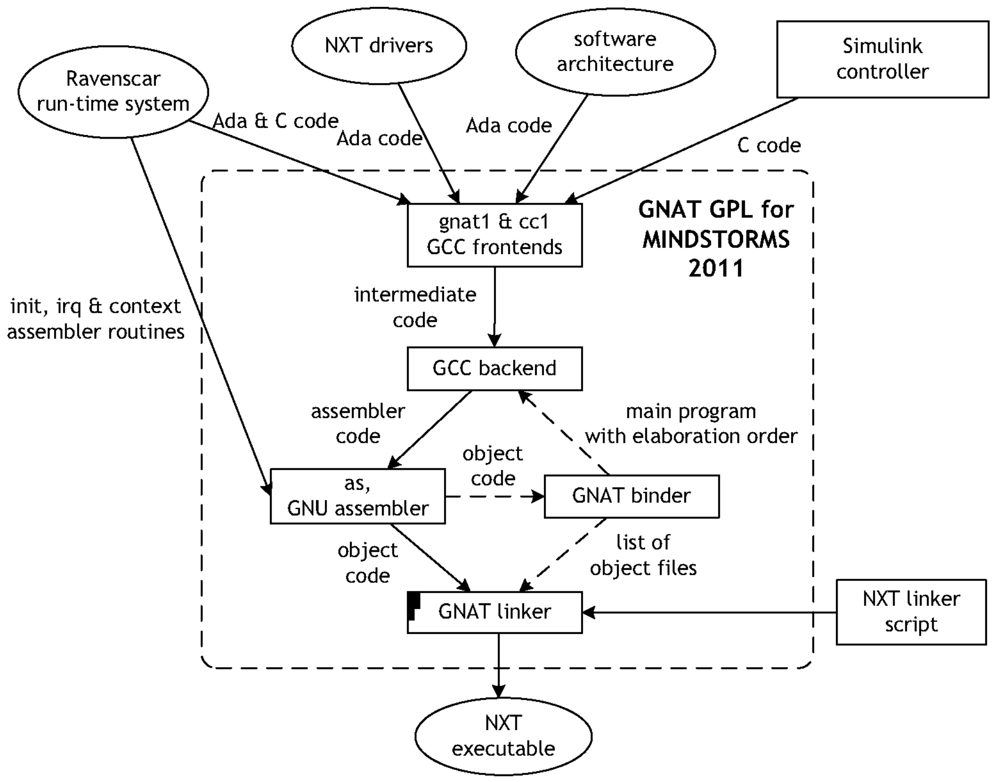

Fig. 2. Executable generation process for NXT.

4.2 Controller Design

As established previously, Simulink is used to design a feedback controller for the NXT. Besides the Simulink environment, a couple of its toolboxes are used. In order to translate the controller to C code for the ARM main processor of the NXT the Embedded Coder is needed.

4.3 Cross-compiler toolchain

A cross-compiler toolchain is a set of tools (essentially a compiler, an assembler and a linker) that generate executable code for a platform, in this case the NXT, other than the one on which the tools run, in this case GNU/Linux x86. Cross-compiler toolchains are used to compile code for a platform upon which it is not feasible to do such compiling.

AdaCore has ported the GNAT compiler toolchain to the ARM architecture by adapting part of the LEON-based Open Ravenscar Real-Time Kernel, ORK+ (de la Puente et al., 2000). The result is the GNAT GPL for MINDSTORMS cross-compiler toolchain which is available, at the time of writing, only for Windows platforms. The 2011 version that has been used for this project has been ported to a GNU/Linux x86 host by the STRAST research group from Universidad Politécnica de Madrid.[2]

4.4 Run-time system and Drivers

The GNAT GPL for LEGO MINDSTORMS NXT 2011 cross-compiler toolchain relies on a Ravenscar small footprint run-time system (Ravenscar SFP). This means that Ada applications for the NXT should comply with the Ravenscar profile for tasking purposes.

The NXT drivers developed by AdaCore are completely coded in Ada. These drivers allow Ada applications to manage the NXT peripherals.

4.5 Compiling a program

To generate an executable NXT file from an Ada application the GNAT cross-toolchain needs first to compile and then link to RAM the resulting object code using a linker script. The code that needs to be compiled is the Simulink controller C code, the real-time software architecture Ada code, the Ada NXT required drivers, the run-time system which includes Ada, C and assembly code and the elaboration code generated by the GNAT binder, see Fig. 2.

To accomplish all of this, a GNU `make` script is in charge of calling the different tools of the GNAT GPL for MINDSTORMS 2011 toolchain.

4.6 Debugging a program

A remote debugger is an extremely useful tool for an embedded system developer. It can drastically decrease development time. There is a way, described in Bradley et al. (2011), to remotely debug Ada/C programs for the NXT using the GNU debugger (GDB) and the ARM EmbeddedICE (In-circuit Emulator) technology. The ARM EmbeddedICE is a JTAG-based debugging channel available on the NXT.

[2] Available from `dit.upm.es/rts/projects/mindstorms`.

5. VEHICLE PROTOTYPE

As proof of concept, a wired controlled vehicle based on Bradley et al. (2010) with added speed control is designed.

5.1 Vehicle overview

The first task is to establish the functionality of the model, in this case, a wired control vehicle with speed control. This vehicle prototype has a front castor wheel to ease turns and provide stability. It also has two back wheels, each driven by an independent motor. Speed control of these two motors guarantees the adequate motion of the vehicle. To control the vehicle, a hardwired joystick, made with a touch sensor to start/stop motion and a motor to use its encoder to control turns, is used. Depending on the angle of the joystick encoder, different speed commands are sent to the back motors, thus controlling vehicle motion, see Fig. 3.

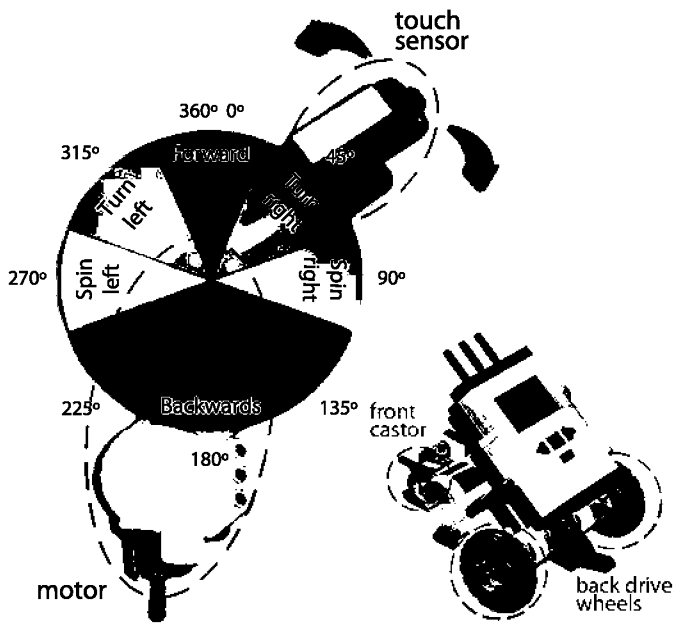

Fig. 3. Wired controlled vehicle prototype.

5.2 Controller design

The purpose of adding a speed control feedback system is to make the NXT able to follow a straight path. To do so, a PID is designed and implemented for both of the back motors. The NXT motor needs first to be characterised in order to study the PID's behaviour on Simulink.

Transfer function According to Franklin et al. (2002) a DC motor's transfer function between the voltage input and the output speed, considering the relative effect of the inductance negligible to the mechanical motion, has the following equation:

$$\frac{\Omega(s)}{V(s)} = \frac{K}{\tau s + 1} \tag{1}$$

where K equals (2) and τ equals (3). Being K_t the torque constant, K_e the electric constant, R the electric resistance, b the rotor's viscous friction coefficient and J the rotor's inertia.

$$K = \frac{K_t}{bR + K_t K_e} \tag{2}$$

$$\tau = \frac{RJ}{bR + K_t K_e} \tag{3}$$

The values used in this work (see Table 1) have been taken from Ryo Watanabe & Philippe E. Hurbain (`http://www.philohome.com/nxtmotor/nxtmotor.htm`) except for the rotor's inertia that has been taken from Sánchez et al. (2009).

Table 1. NXT motor values

$K_t = 0.31739$	$\mathrm{N \cdot m/A}$
$K_e = 0.46839$	$\mathrm{V \cdot s/rad}$
$R = 6.8562$	Ω
$b = 1.1278 \times 10^{-3}$	$\mathrm{N \cdot m \cdot s/rad}$
$J = 0.842 \times 10^{-3}$	$\mathrm{Kg \cdot m^2}$

Note that, although the LEGO motors exhibit a slight overshoot (Kim, 2011), for the purpose of this work it will not be considered.

Simulink model A PID, actually a PI, control feedback mechanism is used to control the motors speed. This closed loop is implemented in Simulink to tune the PI and later generate the functional C code for each of the back motors. Figure 4 shows the Simulink model used. The PI receives the difference between the desired voltage (represented by a step function) and the actual voltage that is calculated by multiplying the motor speed in rad/s (output of the transfer function) by 57.2958 to obtain degrees/s and then dividing the result by 116. This value, 116, has proven to be the best approximation to obtain volts from degrees/s on the NXT. The way the NXT determines the speed of a motor is by dividing the difference between two consecutive readings from the motor encoder by the elapsed time between these consecutive readings. The elapsed time will be close to the period of the task in charge of executing the PI as long as deadlines are generally met.

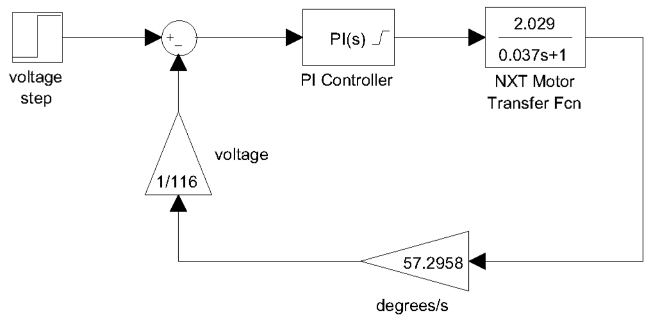

Fig. 4. Speed control loop for NXT motor.

5.3 Real-time software architecture

The vehicle application needs to address the following:

- Control the speed of the two back motors using the designed controller. Note that two controllers are needed, one for each motor.

– Periodically read the encoder value of each back motor to calculate the speed of the motors to feed it back to the controllers.
– Periodically read the battery status to determine the maximum speed in order to feed the controller the correct input voltage.
– Periodically check the status of the joystick motor encoder and touch sensor to determine the commands to the motors.
– Periodically display updated information of the vehicle's status on the LCD screen.
– Manage the communications between processor and co-processor.
– Periodically check the status of the power button to power-off the system.

These activities can be distributed in the following tasks:

– *Control Task*: This task is in charge of the speed controllers and the vehicle commands given by the joystick. It will basically start/stop motion, change motor speeds to achieve turns and control the speed. This task has to be executed periodically. To determine the period of the task some issues need to be taken into consideration. For example, the period has to be short enough to detect a human activated joystick. If its too long not all start/stop drive orders will be captured. At the same time, the period must be long enough so that task deadlines are generally met. Also, since the PI controller needs to be discretised to work in an embedded environment it is important to sample the motor speed quick enough to have a correct response. A tenth of the transfer function's time constant is used. Since $\tau = 0.037$, 4 ms are initially used. This task will have a high priority.
– *Co-processor Task*: The co-processor task handles the communication scheme between the ARM and the AVR. This task must have, at the most, the period of *Control Task* so that the speed controllers receive feedback data on time (2 ms are initially used). This task will have the highest priority.
– *Status Task*: This task gathers data to monitor the controllers and displays it on screen every 500 ms. This task has a low priority.

5.4 PI code generation

Once the controller has been discretised, tuned and stabilised with the desired settling time (approx. 400 ms is used with almost no overshoot) a new model with only the discretised tuned PI, the voltage inputs and voltage output of the PI is created (see Fig. 5). The feedback process is handled outside this algorithm because the Ada drivers API needs to be used to read the motors encoders and the battery voltage. The C code is generated using this model.

Both input ports (*input_voltage* and *real_voltage*) and the output port (*corrected voltage*) are declared as C external variables. The way of integrating the C functional code with the Ada applications is by means of these variables. Therefore, they are defined in Ada and exported to C.

There are several issues when directing the Simulink environment for code generation that are out of the scope of this paper. However, it must be noticed that only the functional code of the controller is needed. The periodic execution of this code is performed by an Ada task. No tasking or sampling should happen inside the generated code.

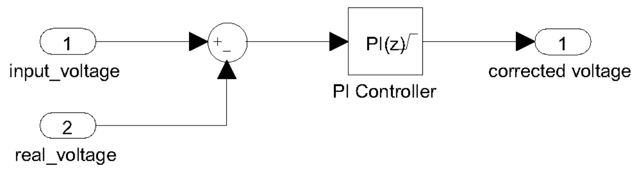

Fig. 5. Simulink model for code generation.

Note also that the NXT hardware does not include a Floating-Point Unit (FPU). As a result, floating-point operations are software emulated which is much slower than integer operations. The use of floating-point numbers should be kept to a minimum when designing Simulink models for the NXT.

5.5 Software implementation

The Simulink code generation process outputs three files: `rtwtypes.h` which include specific ARM7 type definitions and generic type definitions, `functional.c` which includes the model step function functional_step (), and the associated header file `functional.h`.

To implement the real-time architecture proposed in 5.3 three Ada compilation units are used: The main procedure (`vehicle.adb`), a package declaration (`tasks.ads`), and its body (`tasks.adb`). The Tasks package includes two tasks (Control_Task, which calls functional_step , and Display_Task), as well as some auxiliary functions. The periodic task in charge of the co-processor communication is part of the Ada NXT drivers and is declared in the NXT. AVR package. This package must always be imported even if its functions are not required. Therefore, an Ada with clause to import NXT.AVR is added to the main program to guarantee the execution of this task.

Listing 1 shows a fragment of `tasks.ads` containing the declaration of tasks. When declaring a task, besides using pragma Priority to establish the static priority, pragma Storage_Size is used. The NXT has only 64 KB of RAM memory and this pragma allows the programmer to allocate the memory needed for the task stacks.

Listing 1. Specification of tasks.

```
task Control_Task is
   pragma Priority
      (System. Priority ' First + 2);
   pragma Storage_Size (4096);
end Control_Task;

task Display_Task is
   pragma Priority
      (System. Priority ' First + 1);
   pragma Storage_Size (4096);
end Display_Task;
```

As stated in 5.4, *input_voltage*, *real_voltage* and *corrected voltage* are declared as external C variables. They must be defined as Ada variables and made visible for the C code. This is achieved using the compiler directive pragma Export. Also, the C functional_step function must be made visible so that Control_Task is able to call it periodically. This is achieved using the compiler directive pragma Import. Listing 2 shows how the above is accomplished.

Listing 2. Interface between Ada and C code.

```
-- Import functional_step from Simulink generated code

 procedure PID;
 pragma Import (C, PID, "functional_step");

  -- Export inputs to Simulink generated code

 type ExternalInputs_functional  is
   record
       Input_Voltage : Float;
       Real_Voltage : Float;
  end record;

 Functional_U : ExternalInputs_functional ;
 pragma Export (C, Functional_U, "functional_U");

  -- Export output to Simulink generated code

 type ExternalOutputs_functional  is
  record
     correctedvoltage : Float;
  end record;

 Functional_Y : ExternalOutputs_functional ;
 pragma Export (C, Functional_Y, "functional_Y");
```

6. CONCLUSION & FUTURE WORK

The real-time control platform described in the paper has proved to be very useful for real-time control students. The possibility of separating control design using Simulink and concurrency and real-time aspects of a system, which are best described using Ada, makes it simpler to carry out laboratory projects, and has resulted in a shorter learning time for the students. The platform is made of comparatively low-cost components, which makes it affordable for the academic environment.

Planned future developments include integrating schedulability analysis tools, and providing feedback to Simulink via Bluetooth in order to be able to analyse on-line the performance of the designed control system. The MATLAB/Simulink environment was used because our University has a corporative license but we also plan to study other solutions with code generation capabilities.

The entire environment is intended to be use in the matter of Real Time Systems of the new Computer Science Master in compliance with the European Higher Education Area. The students have not previous acknowledge in control theory and we think that the use of this environment may contribute to the integration of both fields in order to properly build embedded systems.

REFERENCES

ALRM05 (2007). *Ada Reference Manual ISO/IEC 8652:1995(E)/TC1(2000)/AMD1(2007).* Available on `http://www.adaic.com/standards/ada05.html`.

Bradley, P.J., de la Puente, J.A., and Zamorano, J. (2010). Real-time system development in Ada using LEGO MINDSTORMS NXT. *Ada Letters*, XXX(3), 37–40. doi:http://doi.acm.org/10.1145/1879063.1879077. URL `http://doi.acm.org/10.1145/1879063.1879077`.

Bradley, P.J., de la Puente, J.A., and Zamorano, J. (2011). Ada user guide for LEGO MINDSTORMS NXT. *Ada User Journal*, 32(3).

Burns, A., Dobbing, B., and Vardanega, T. (2004). Guide for the use of the Ada Ravenscar profile in high integrity systems. *Ada Letters*, XXIV, 1–74. doi:http://doi.acm.org/10.1145/997119.997120. URL `http://doi.acm.org/10.1145/997119.997120`.

Burns, A. and Wellings, A. (2009). *Real-Time Systems and Programming Languages.* Addison-Wesley, 4th edition.

Crespo, A., Albertos, P., Balbastre, P., Valles, M., Lluesma, M., and Simo, J. (2006). Schedulability issues in complex embedded control systems. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, 1200 –1205. doi:10.1109/CACSD-CCA-ISIC.2006.4776813.

de la Puente, J.A., Ruiz, J.F., and Zamorano, J. (2000). An open Ravenscar real-time kernel for GNAT. In H.B. Keller and E. Plödereder (eds.), *Reliable Software Technologies — Ada-Europe 2000*, number 1845 in LNCS, 5–15. Springer-Verlag.

Franklin, G.F., Powell, J.D., and Emani-Naeini, A. (2002). *Feedback Control of Dynamic Systems.* Prentice Hall, Upper Saddle River, NJ, fourth edition.

Grega, W. and Pilat, A. (2008). Real-time control teaching using LEGO MINDSTORMS NXT robot. In *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*, 625 –628. doi:10.1109/IMCSIT.2008.4747308.

Kim, Y. (2011). Control systems lab using a LEGO Mindstorms NXT motor system. *IEEE Tr. Education*, 54(3), 452 –461. doi:10.1109/TE.2010.2076284.

Sha, L., Rajkumar, R., and Lehoczky, J.P. (1990). Priority inheritance protocols: An approach to real-time synchronization. *IEEE Tr. on Computers*, 39(9).

Sánchez, S., Rodríguez, O., and Arribas, T. (2009). Utilización de LEGO NXT en docencia universitaria. Technical report, Universidad de Alcalá. [in Spanish].

WG23-N0304 (2011). *Vulnerability descriptions for the language C.* ISO/IEC JTC 1/SC 22/WG 23 Programming Language Vulnerabilities.