Embedded Implementation of a Real-time Switching Controller on a Robotic Arm

Giuseppe Ferrò, André C. Neto, Filippo Sartori, Luca Boncagni, Daniele Carnevale, Mateusz Gospodarczyk, Andrea Monti, Alessio Moretti, Riccardo Vitelli, Llorenç Capellà, Ivan Herrero

***Abstract*—The very high availability of low-cost embedded hardware development kits has enabled the fast prototyping of real-time control architectures and algorithms. The software development environments are usually very specific to the target platform, so that is very challenging to develop code that is portable between architectures (e.g. between an ARM and an ATMEL processor).**

The MARTe real-time software is a multi-platform C++ real-time framework which allows the execution of control algorithms, interfaces and services in different operating systems and platforms. A new version of this framework has been developed with a software architecture aiming at enabling the execution of the same code across different bare-metal systems.

This paper presents a project where the new version of the MARTe framework is used for the real-time control of a robotic arm using low-cost embedded technologies. The controllers are achieved by implementing a real time thread with maximum priority that communicates with the motors power amplifier setting the motor voltages and reading the angular position of the joints by mean of optical encoders.

The control algorithm to drive the DC motors is based on a new switching PID theory. Thus, given two different PIDs, the control algorithm can switch from one PID to the other in order to minimize overshoots and oscillations and increase the convergence speed of the angular position to the desired reference.

This work presents and compares the performance of the control algorithm implementation on a bare-metal and on a FreeRTOS deployment. Finally, it discusses the switching controller design improvements.

I. Introduction

A NEW version of the MARTe C++ real-time control framework (known as MARTe1) [1] has been developed with a software architecture that aims at enabling the execution of the same code across different *bare-metal* systems (i.e. a processor without operating system). In particular, this new version of the framework (named MARTe2) is being developed under a fairly strict quality assurance system [2] aiming at demonstrating compliance with the MISRA-C++:2008 [3] standard.

At the core of any MARTe application are the Generic Application Modules (GAMs). Each GAM is a runtime configurable software module with inputs and outputs, implementing a given function. By assembling and connecting a set of GAMs together, different systems can be deployed in distinct hardware architectures without having to change the source-code of these modules.

One of the limitations of MARTe1 was that it required an operating system to execute. Taking advantage of the fact that there was a new MARTe version being developed and of the high availability of low-cost embedded hardware development solutions, it was decided to impose on the MARTe2 design the requirement of being able to execute a MARTe1 equivalent application in a *bare-metal* system. Given that MARTe2 is aiming at critical systems, it was also decided to support the FreeRTOS [4] operating system (which also has a quality certifiable version).

The STM32F4-Discovery board from STMicroelectronics®, a low price ecosystem with an ARM® Cortex-M4® 32-bit processor with a FPU core, 1 MB of flash memory and 192 MB of RAM memory was selected as the environment to prototype the controller implementation. The system to be controlled is an anthropomorphic five degrees robotic arm manipulator called *Scortec-ER* [5].

This paper details the full system implementation, starting from the architecture design, followed by the high-level implementation details and concluded with the system performance measurements.

II. Architecture

As shown in Fig. 1 the system architecture is divided in four macro blocks: an external PC, a STM32F4-Discovery board, a power unit and the robotic arm. The control software is embedded in the STM32F4-Discovery board, while the PC can be used to monitor and control the embedded system . Each of these components is detailed in the following paragraphs.

A. External PC

This component is only required when the reference signals to the motors have to be provided to the embedded STM32F4-Discovery board from an external source. For example, using

Manuscript received May 29, 2016. The work leading to this publication has been funded partially by Fusion for Energy under the Contracts F4E-OFC-361-06 and F4E-OFC-620-01. This publication reflects the views only of the author, and Fusion for Energy cannot be held responsible for any use which may be made of the information contained therein.

G. Ferrò is with with CREATE Consortium Via Claudio, 21 80125 Napoli, Italy and with Dipartimento di Ingegneria Civile e Ingeneria Informatica, Università di Roma Tor Vergata, Via del Politecnico 1, 00133 Rome, Italy

A. C. Neto, F. Sartori, R. Vitelli are with Fusion for Energy, Josep Pla 2 - 08019 Barcelona, Spain

L. Boncagni is with ENEA Unità Tecnica Fusione, C.R. Frascati, Via E.Fermi 45, 00044-Frascati, Rome, Italy

D. Carnevale, M. Gospodarczyk, A. Monti, A. Moretti are with Dipartimento di Ingegneria Civile e Ingeneria Informatica, Università di Roma Tor Vergata, Via del Politecnico 1, 00133 Rome, Italy

L. Capellà is with Vitrociset Belgium, Rue Devant les Hetres 2 B-6890 Transinne. Belgiumw

I. Herrero is with GTD Sistemas de Información, Paseo García Faria 17 E-08005 Barcelona

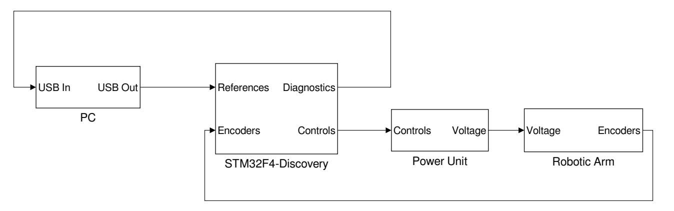

Fig. 1. The system architecture is composed of four main components. The external PC is optional and allows to monitor and configure the embedded system.

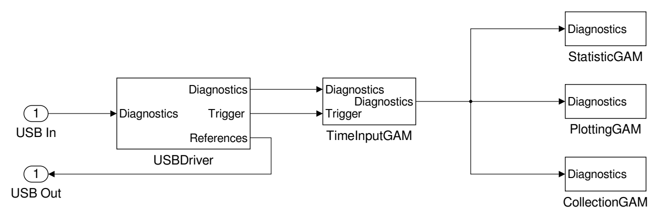

Fig. 3. MARTe1 execution sequence. The *USBDriver* triggers the start of a new cycle and transfers the data to the TimeInputGAM, which shares it with all the real-time processing GAMs.

the PC, the end-user is allowed to set in runtime the desired angle position (in encoder steps) of each motor from an user friendly interface. The PC also enables the access to all the diagnostics data sent from the STM32F4 such as: cycle time, amount of stack memory used, control signal values and encoder position.

The PC software has been deployed employing the MARTe1 framework which already provides an HTTP based user interface that can be customized for specific applications. Moreover, the MARTe1 framework also provides a large number of GAMs and tools for diagnostics. Examples are the *StatisticGAM* (providing real-time statistic data of each signal), the *PlottingGAM* (showing live real-time plotting of signals) and the *CollectionGAM* (data storage and offline retrieval of acquired data).

A MARTe1 hardware interface driver, known as Generic Acquisition Module (GAcqM) in the MARTe1 vocabulary, had to be developed in order to allow the communication between the PC and the MARTe2 running in the STM32F4. The developed GAcqM, named *USBDriver*, allows to synchronously receive in input any number of signals from the USB port. Simultaneously, the *USBDriver* HTTP interface also provides the possibility to asynchronously send the desired encoder reference to any of the motors through the same USB port. The PC block architecture with all the signals that are sent and received during the communication with the embedded board is shown in Fig. 2. The MARTe1 execution sequence is shown in Fig. 3.

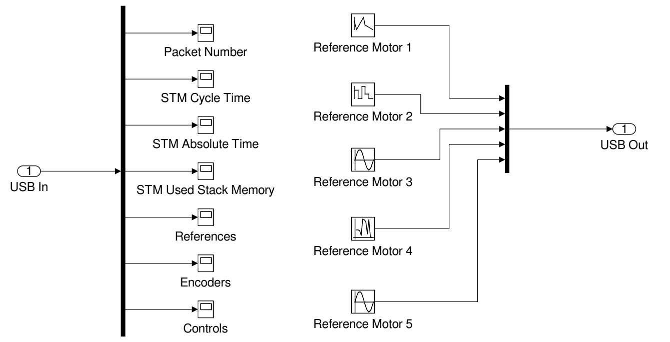

Fig. 2. PC block architecture. Components deployed inside a MARTe1 instance enable the communication with the embedded system. In particular the references to the motors can be asynchronously set using the MARTe1 HTTP user-interface.

B. STM32F4-Discovery

As discussed before, the control board selected for this project is the STM32F4-Discovery from STMicroelectronics®. This board provides many configurable peripherals and internal hardware timers that must be adjusted for any given project. The selected configuration includes the following components:

- **USB OTG FS** with a micro A-B connector to communicate with the PC. The baud rate was set to 115200 with 8 data bits and 1 stop bit (115200 8N1). The interrupt for this peripheral was set with priority 6.
- **UART2** using pins D5=Tx, D6=Rx. This port was used as a debug serial stream to receive error messages from the embedded board. The baud rate was set to 9600 with 8 data bits and 1 stop bit (9600 8N1).
- **16 digital GPIO** pins to read the number of encoders of the motors from the power unit.
- **5 PWM channels** (one for each motor). Since each STM32F4-Discovery timer can drive at most four PWM channels it is necessary to use two timers in order to provide the PWM signals to the five motors of the robot.
- **Timer 2** to trigger an interrupt every 2.5 ms and post an event semaphore. This can be used as a 400 Hz synchronous control cycle mechanism. This interrupt was set with priority 5 (the maximum value when using FreeRTOS).
- **Timer 5** to provide the high resolution timer. It triggers an input every 100 ms and increments a counter variable. Combining the value of this variable with the value of the internal counter value it allows to timestamp with a resolution of 1 μs. This interrupt also has maximum priority.
- **User-button** connected to the pin A0. The pressing of the button triggers an interrupt and allows the application to react to the event. This interrupt has a priority equal to 7.

Regarding the MARTe2 software deployed in the STM32F4 processor, when the application starts, the board waits to receive its configuration parameters from the USB port. This method allows the user to run different MARTe2 applications and to change the configuration, without needing to recompile and reload the code running on the board.

A palette of MARTe2 GAMs have been developed specifically for this project. The *ReferenceSignalGAM* is the signal generator and is used to provide the controller reference signals. It can generate sine waveforms with configurable frequency, offset, phase and amplitude or, alternatively, it can interpolate a pre-configured array of time-value pairs. The *ScortecControlGAM* implements the control system. It takes the reference signals and the encoder values in input and executes the desired control procedure. Given that the deployed applications implement PID-like controllers, all the controller parameters, such as the proportional, integral and derivative gains (K_p, K_i, K_d), saturations and dead zones can be set by configuration.

In order to be able to test the system without having the actual robot connected, a *ModelGAM* simulates a group of SISO or MIMO plants. It is a container of customizable *Plant* classes which can be implemented accordingly to the system that is to be simulated. In order to simulate the dynamics of the DC motor, we have implemented a SISO LTI plant, where by configuration it is possible to define the linear matrices A, the vectors B and C and the scalar D.

Concerning the hardware interface, a *ScortecEncoderModule* reads the encoder values of the motors from the power unit, a *ScortecPWMModule* the manages the PWM signals that are used to drive the motors, a *StmUSBModule* handles the interface with the USB port and a *StmTimeGeneratorModule* generates the absolute time in seconds from the beginning of the application.

The communication protocol with the power unit requires two ports of 8 digital pins, one used as a command line and the other as a data line. For each motor it is necessary to send a read request, sleep for 100 μs and only then read the encoder value. This procedure, which has to be repeated for each motor, is the major limiting factor on the cycle time of the system. The *ScortecPWMModule* allows to configure the PWM timers, frequency, duty-cycle resolution and the pins layout. It computes the duty cycle related to each control signal in input (provided by the *ScortecControlGAM*) and sets it to the PWM peripheral. The *StmUSBModule* manages the interface with the USB port. It can be configured to be a receiver or a sender in blocking or non-blocking mode. In the deployed applications we have defined two of these modules, one is a sender which writes the diagnostics data at the beginning of each cycle, and another as a receiver in non-blocking mode which allows to read asynchronously the reference signals from the outside. The program functional behavior is shown in Fig. 4.

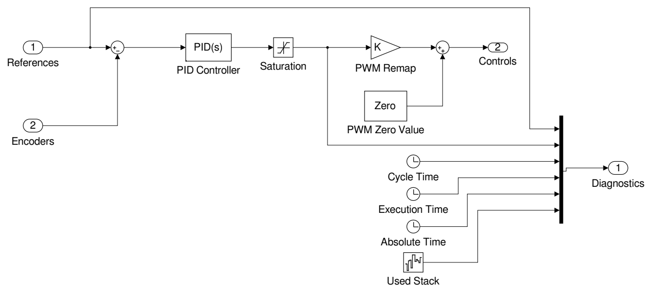

Fig. 4. STM32F4-Discovery block functional behavior. The controller is driven by a reference set by the user and a measurement given by the encoders. The outputs are the internal variables values (for debug) and the control voltage.

As depicted in Fig. 5 and Fig. 6, two applications, using the modules described above, have been developed. Both applications work in *bare-metal* mode and with the FreeRTOS operating system. The first application (Application-1) is to be used with the Scortec robot physically connected to the board. In this case the only GAM required is the *ScortecControlGAM* which computes the control signals from the *StmUSBModule* (the references) and from the *ScortecEncoderModule*(the encoders) and writes directly the output (control signal) on the *ScortecPWMModule*. The *ScortecControlGAM* also needs to know the cycle time value, in order to be able to compute derivatives and integrals of the signals. Given that, for performance statistics, the MARTe2 GAM scheduler generates and sets, for each GAM, the time elapsed from the begin to the end of the GAM execution (known as *RelativeUsecTime*) and the time elapsed from the begin of the last cycle to the end of the GAM execution (named as *AbsoluteUsecTime*), the *ScortecControlGAM* can use its *AbsoluteUsecTime* to compute the sample time.

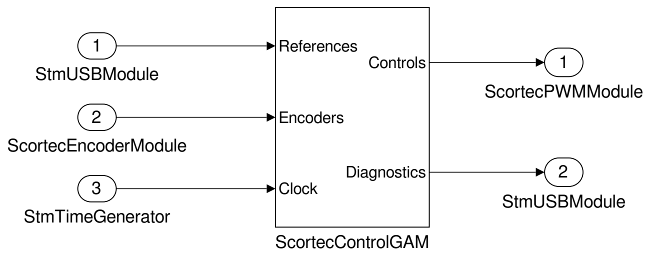

Fig. 5. Execution sequence of the MARTe2 application that is physically connected to the robot (Application-1).

The second application (Application-2) has the purpose of performing a stand-alone simulation of a closed-loop control system. The reference signals are generated from the *ReferenceSignalGAM* and the encoders are the output of the *ModelGAM* which can be used to simulate the behavior of generic plants. These three GAMs connected together as shown in Fig. 6 can simulate any type of closed-loop system.

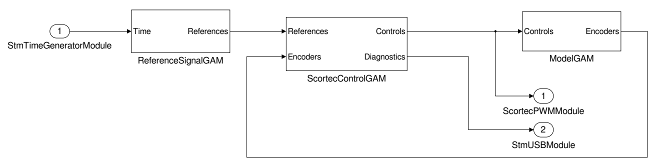

Fig. 6. Execution sequence of the MARTe2 application that is used to simulate the connection to the robot (Application-2).

C. Power Unit

The input port of the power unit is a 37-pin D female connector (ITT/Cannon DC-37S) with 6 analog input pins connected the motor power supply drivers and three blocks of 8 digital pins. Thus two ports of 8 digital pins are connected to the STM32F4-Discovery pins and they are used to read the encoder values. The power unit needs, for each motor, an analog signal between +5 and -5 Volts to drive the power supply. As a consequence, a stage between the board and the power unit to convert the PWM signal to an analog input is needed. In order to achieve this goal a second order low pass filter has been implemented and the poles of its transfer function have been set to $p_1 = p_2 = 666.67$. Since the PWM signal value can vary from 0 V to 3.3 V it is also necessary to amplify the signal by a factor $k = 3$ and then to add a constant -5V value to obtain an output in the [-5 V, +5 V] range. Fig. 7 depicts the power unit block scheme, Fig. 8 shows the electronic circuit scheme of the filter and Fig. 9 depicts the output amplitude in the frequency domain. Fig. 10 shows the output in the time domain in three cases: 100%, 50% and 0% duty cycle with a 1 kHz PWM frequency. From Fig. 9 it can be seen that in order to achieve a substantial attenuation of the PWM signals undesired frequencies , it is advisable to use a PWM frequency of at least 1 kHz. The frequency of the PWM signals can be configured in the configuration file, but in our applications we have always set it to 10 kHz. It is important to note that with this output stage a 50% PWM duty cycle is required to achieve a 0 V signal in the output. As a consequence the STM32F4 board must set the PWM duty cycles to 50% before initializing the power unit (otherwise with the PWMs off, -5 V will be provided as input voltage to the motors). The outputs of these stages (one for each motor) are connected to the analog pins of the power unit connector completing the interconnection with the STM32F4-Discovery board.

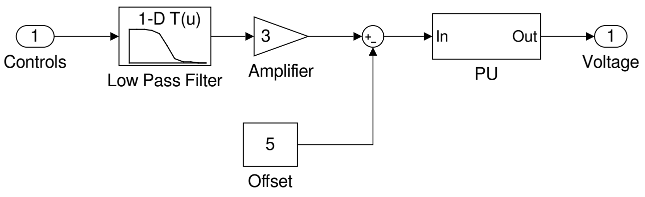

Fig. 7. Power unit interface to the control output. A low-pass filter translate the PWM voltage to the required analog range.

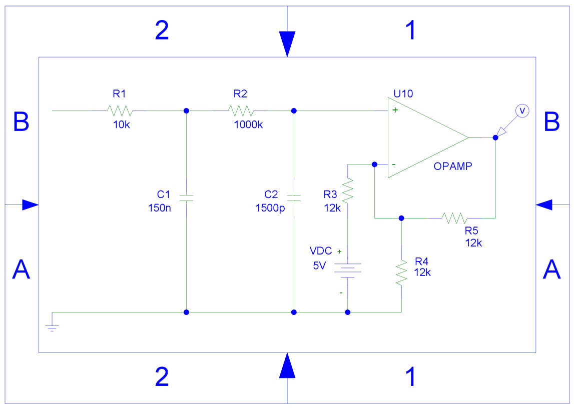

Fig. 8. Low pass filter scheme (with amplification and offset) for the translation between the PWM and the power supply analog input.

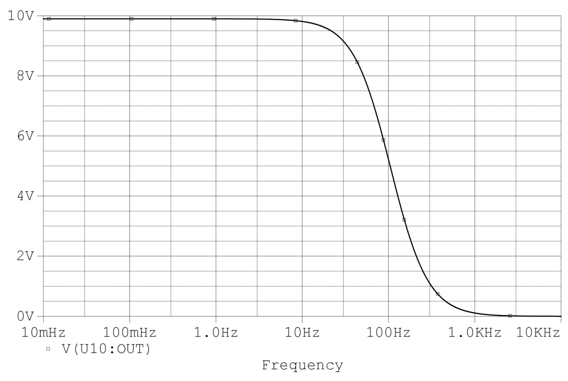

Fig. 9. Frequency response of the filtering stage electronic circuit.

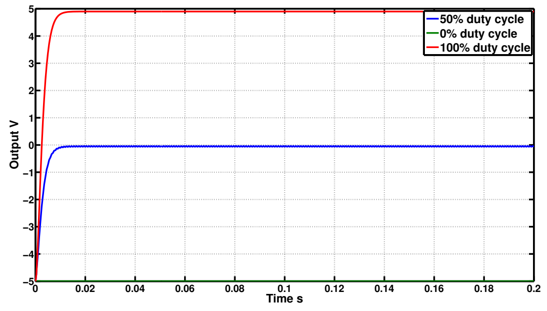

Fig. 10. Output of the filter with 50% (blue), 0% (green), 100% (red) PWM duty cycles at 1 kHz frequency

D. *Robotic Arm*

The robotic arm is fully driven by its power unit, with the encoder values read from a PIC microprocessor, while the control signals must be delivered to the power unit internal power supply driver. Each motor can move a single joint, one on the base, one on the shoulder, one on the elbow and the last two manage the wrist movements, commonly called pitch and roll (see Fig. 11).

The model of an anthropomorphic five-degrees manipulator, like the *Scortec*, is a MIMO non-linear plant (products of sine and cosine functions of the angular positions appear in the model equations of motion). Given that the analysis of the model and the project of a ad-hoc controller for the entire system are out of the scope of this paper, we have considered each motor as if it was decoupled from the others, implementing a separate controller for each one of them.

III. System Performance

One of the main goals of this paper was the comparison between the performance measured using a *bare-metal* GAM scheduler and the performance achieved using the standard GAM scheduler executed in the FreeRTOS operating system. It should be noted that using an operating system like FreeRTOS offers significant advantages, such as the possibility to have multi-threading in the application.

The applications have been tested in both cases with a synchronizing cycle time of 2.5 ms. In *bare-metal* the system waits for the start of the next cycle on a spin-lock semaphore while in FreeRTOS it waits in an event semaphore. The measured cycle times are shown in Fig. 12 and Fig. 13, while Fig. 14 and Fig. 15 show the total execution time, namely the amount of time in which the *ScortecControlGAM* is executing. The following results have been obtained from the two simulations in Bare-Metal and FreeRTOS modes respectively, where CT denotes the cycle time, ET the execution time, ρ the mean and σ the standard deviation:

- **Bare-Metal**: $\rho(CT)$ = 2.472 ms; $\sigma(CT)$ = 28 μs; $\rho(ET)$ = 1.789 ms; $\sigma(ET)$ = 19 μs
- **FreeRTOS**: $\rho(CT)$ = 2.473 ms; $\sigma(CT)$ = 28 μs; $\rho(ET)$ = 1.788 ms; $\sigma(ET)$ = 19 μs

As discussed above, one of the advantages of FreeRTOS over *bare-metal* applications is the possibility of running several threads in parallel. In a real-time application this can be used to asynchronously monitor the application without interfering with the real-time application. In order to demonstrate this feature, we have implemented on Application-1 a second thread which executes concurrently (but with lower-priority) to the GAM scheduler thread. Using this thread and the MARTe1 HTTP server, the user can query the properties of any the objects deployed in the MARTe2 embedded instance. Namely, this thread implements a connection to the UART port and upon request introspects and prints, also in the UART port, the properties of the object. Fig. 16 and Fig. 17 show the performance of this two-threading application when continuously sending asynchronous print requests. The measured results in this case are $\rho(CT)$ = 2.484 ms, $\sigma(CT)$ = 29 μs, $\rho(ET)$ = 1.785 ms, $\sigma(ET)$ = 19 μs, , where it can be seen that the system cycle-time is not affected by a second monitoring thread, thus demonstrating the robustness of the design and of FreeRTOS. Moreover, these performance figures are also similar to the ones obtained with MARTe1 [6], [7].

IV. System Simulation

One of the main features of MARTe, which is also retained in MARTe2, is the possibility to build applications by assembling different functional blocks, without the need to change or recompile the user-code. This feature was used as the basis for the design of the second application (Application-2), where an hardware-in-the-loop approach was used to simulate and test the behavior of the embedded program without physically connecting it to the board of the device to be controlled.

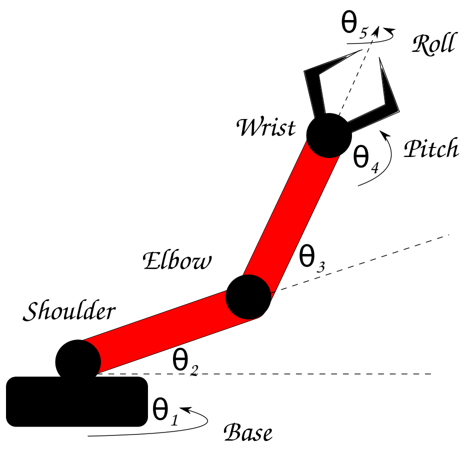

Fig. 11. The *Scortec* robot is an anthropomorphic five degrees manipulator. Each of its degrees of freedom is controlled by an independent motor.

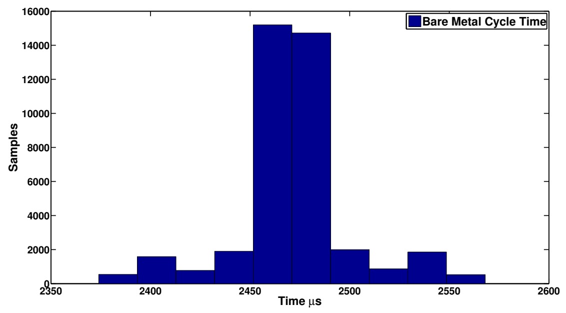

Fig. 12. *Bare-metal* application cycle time computed over 40000 samples

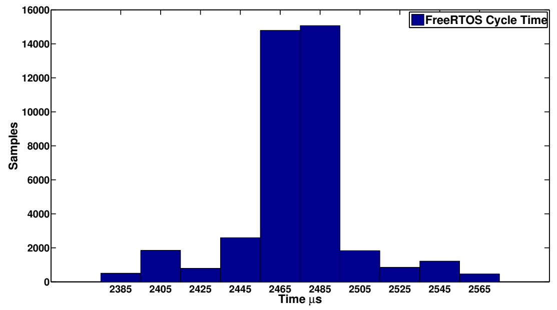

Fig. 13. FreeRTOS application cycle time computed over 40000 samples. The outliers are possibly driven by the USB communication but do not greatly affect the cycle time, i.e. the application is capable of restarting a new control cycle with-in an acceptable jitter.

In order to test the correctness of the *ScortecControlGAM* which currently provides a controller for each motor, we have identified the transfer function of one of the *Scortec* DC motors and we have simulated the entire closed loop system inside the embedded STM32F4 board. The first step was to define a standard position for the robot in which all the encoder values had to be reset to zero. This position is commonly called the robot *home* position. A common practice is to define the *home* position as the position in which all the motors stand at their limit switch. Unfortunately the *Scortec* robot does not provide hardware switch sensors, so that it was necessary to implement a software procedure to check if each motor has reached its limit switch. The adopted solution consists in assuming that the motor has reached its limit switch if the control output is different from zero (or, more realistically, inside a configurable dead zone) and the encoder value is kept constant for more than a configurable number of application cycles. The MARTe2 application is divided in two different states: the first (State-1) is only devoted to bring the robot back into the *home* position, the second state (State-2) allows to control the robot by imposing the desired encoder positions for each motor.

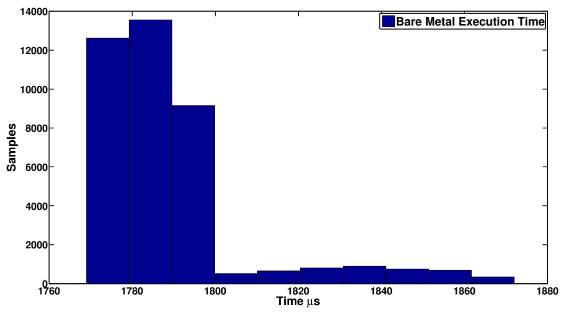

Fig. 14. *Bare-Metal* application execution time computed over 40000 samples.

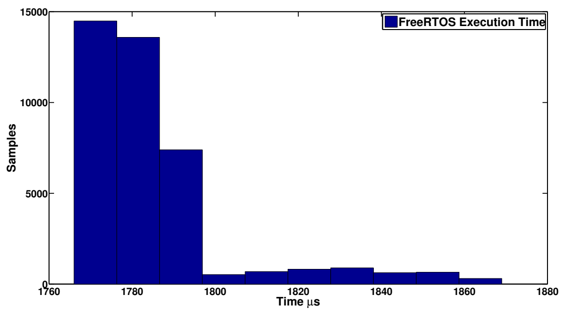

Fig. 15. FreeRTOS application execution time computed over 40000 samples. As in the *bare-metal* case the outliers are driven by a jitter in the USB interface.

During State-2 the *ScortecControlGAM* assumes that:

$$r_i = y_i + k_r$$

where r is the reference motor position, y the encoder value and k_r a configurable constant value for $i = 1 \dots n_{motors}$. In this way, using the implemented controllers, the motors will keep moving in the same direction until they will reach their limit switch, where the control and encoder values will be set to zero. Once the robot has reached its home position the user, pressing the user button of the STM32F4-Discovery board, triggers a state change inside MARTe2 and changes the application to the normal operation State-2.

The model of the controller shown in Fig. 18 was implemented using the *ModelGAM* and executed within Application-2. The parameters were identified by providing input voltages with different frequencies to the power unit and reading the encoder values in return. Defining:

- L_a, R_a the motor armature inductance and resistance.
- J the momentum of inertia.
- B the viscous friction.
- τ_d the disturbance torque.
- K_e the motor speed constant.
- K_a the motor torque constant.
- K_m the motor mechanical constant.

the transfer function between the input voltage v (measured in mV) and the angular speed ω (the derivative of the encoder

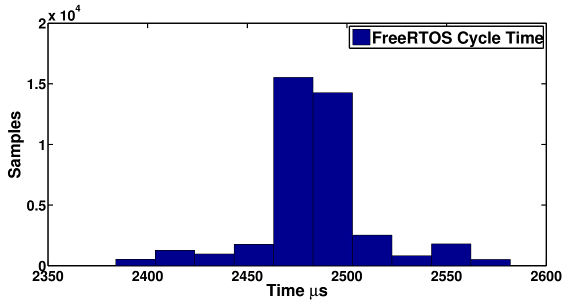

Fig. 16. FreeRTOS two-threading application cycle time computed over 40000 samples, where it can be seen that the system cycle-time is not affected by a second monitoring thread.

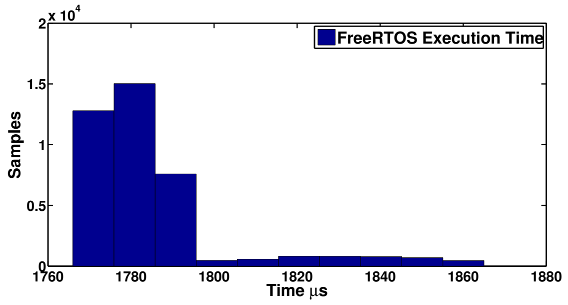

Fig. 17. FreeRTOS two-threading application execution time computed over 40000 samples.

position) is:

$$P_{v\omega}(s) = \frac{F(s)}{1 + K_e F(s)} - \frac{\tau_d K_m}{(Js + B)(1 + K_e F(s))}$$

with:

$$F(s) = \frac{K_m K_a}{(L_a s + R_a)(Js + B)}$$

For the identification we have ignored the filtering stage and the motor electrical dynamics, because their time-constants are much faster than the mechanical dynamics of the motor:

$$F(s) = \frac{K_m K_a}{Js + B}$$

Moreover assuming the disturbance torque $\tau_d = 0$ and defining $K_T = K_m K_a$ we obtain:

$$P_{v\omega}(s) = \frac{K_T}{Js + B + K_e K_T}$$

and accordingly, adding the integrator, we obtain the second-order transfer function from input voltage to the angular encoder position:

$$P_{v\theta}(s) = \frac{K_T}{s(Js + B + K_e K_T)}$$

The identified parameters for the motor are: $\frac{K_T}{J} = 0.96$, $\frac{B+K_e K_T}{J} = 6.05$. Note that since the plant transfer function already has a pole in the origin, we achieve zero steady-state tracking error for constant references by just employing a simple proportional gain as controller. As described previously, the control signal, namely the input voltage to the motor, has been saturated between [-5 V, +5 V]. As consequence, also imposing an higher value of the proportional value K_p,

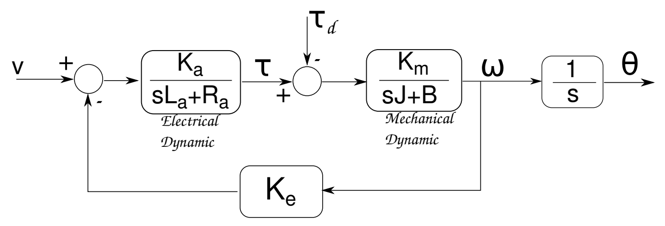

Fig. 18. The DC motor model implemented in the *ModelGAM*

the saturation attenuates the control strength. Nevertheless, a high value of K_p could still lead to undesired overshoots, thus we have implemented a switching PID controller which can change the value of its gains depending on the value of the tracking error. In this case, attenuating the proportional gain when the module of the error is sufficiently small, we can achieve a good convergence speed to the reference signal decreasing or avoiding the overshoot.

Fig. 19 and Fig. 20 show the results of the hardware-in-the-loop (i.e. executed in the STM32F4) simulations in which we have used a simple proportional controller with $K_p = 25$ and after a switching proportional controller beginning with $K_p = 25$ but switching to $K_p = 8$ when the module of the error became less than 200. Fig. 19 compares the outputs of the closed-loop system and we can note that the overshoot employing the classic proportional controller does not affect the output response if we employ the switching proportional instead. Fig. 20 shows the control signal for both controllers and we can observe the discontinuity appearing in the input voltage provided by the switching controller in the switching time instant.

This approach can be considered as a simplified version of the controller described in [8] and, with a minimum of effort required in adjusting the controller coefficients and gains, it can assure very good performance results. The system behavior during State-1, using the switching proportional controller, is shown in Fig. 21, where the value of k_r has been set to 100 and the limit switch to 3000 encoders. Note that, since the error value in this phase is constant and equal to $e = k_r = 100$, which is less than the imposed switching threshold ($\sigma(e) = 200$), the gain remains $K_p = 8$, thus, before the motor reaches its limit switch, the control action is always equal to $K_p \cdot e = 800$. Once it is detected that the input voltage is different than zero but the encoder read value remains constant along a configurable number of cycles (in this case set to $500 \simeq 1.25$ s), it is assumed that the motor has reached the limit switch (the joint movement is blocked) and the input voltage is set to zero.

V. Conclusions

In this paper we have presented an embedded implementation of a control project using the MARTe2 framework. In particular, the framework performance was measured in two use-cases that are of potential interest for the development of embedded (i.e. systems with limited amount of memory, number of cores and clock speed) real-time applications: *bare-metal* and FreeRTOS. Given that MARTe2 is being developed under a fairly strict quality assurance process this gives the potentiality to deploy critical systems without incurring in large effort penalties (most of the effort was already put into the framework development itself). It was also shown that FreeRTOS is also a viable solution with performance figures similar to *bare-metal*. Given that this project was developed in parallel with the development of the MARTe2 framework, it was also of significant importance to make sure that the framework design did not prevent the development of this type of applications.

Taking advantage of the developed application, and considering the model of a *Scortec* DC motor, we have deployed a switching proportional controller showing the advantages in terms of performance during the transient of the system

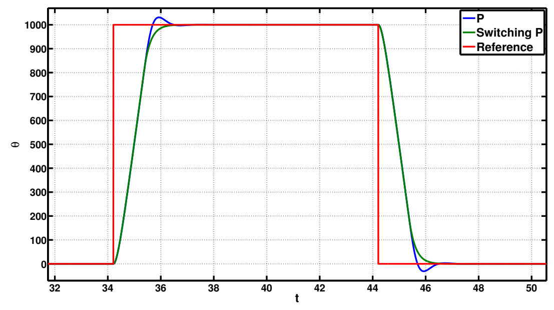

Fig. 19. Closed-loop system output with fixed proportional controller (blue), and with switching proportional controller (green).

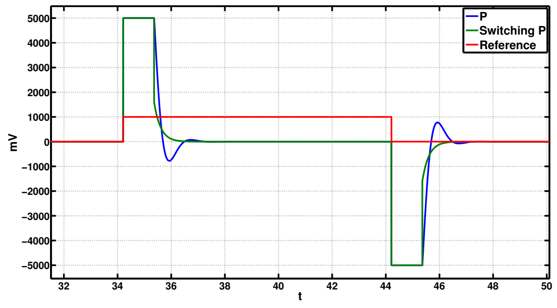

Fig. 20. Input voltages to the motors with fixed proportional controller (blue), and with switching proportional controller (green).

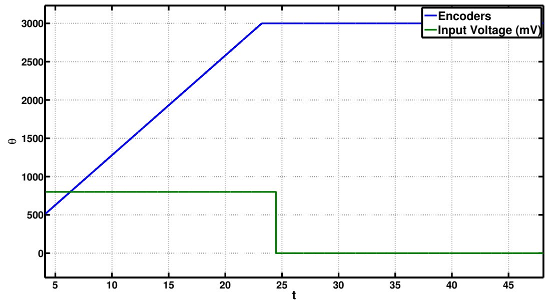

Fig. 21. State-1 Home position *Scortec* routine for one of the joints. In blue the number of encoders, in green the input voltage. The joint home position (limit switch) has been set to 3000 encoders.

response (overshoot and convergence speed) with respect to fixed-gain proportional controller. In the future, the MARTe2 block implementing the switching controller can be configured for instance to adapt the PID gains depending on the torque load acting on the robotic arm joints, when considering the entire robot model.

It should be noted that this same infrastructure can be used to develop other real-time applications in any similar ecosystem.

REFERENCES

[1] A. C. Neto, D. Alves, L. Boncagni, P. J. Carvalho, D. F. Valcarcel, A. Barbalace, G. De Tommasi, H. Fernandes, F. Sartori, E. Vitale, R. Vitelli, and L. Zabeo, "A survey of recent marte based systems," *IEEE Transactions on Nuclear Science*, vol. 58, no. 4, pp. 1482–1489, 2011.

[2] A. C. Neto, F. Sartori, R. Vitelli, L. Capellà, G. Ferrò, and I. H. and Héctor Novella, "An agile quality assurance framework for the development of fusion real-time applications," in *Proc. 20th IEEE-NPSS Real Time Conf. (RT)*, 2016.

[3] *MISRA C++:2008 Guidelines for the Use of the C++ Language in Critical Systems*, 2008, ISBN 978-906400-03-3 (paperback), ISBN 978-906400-04-0 (PDF).

[4] FreeRTOS, "The freertos project," 2016, [Online; accessed 27-may-2016]. [Online]. Available: http://www.freertos.org/index.html

[5] A. Gasparri, S. Panzieri, F. Pascucci, and G. Ulivi, "Pose recovery for a mobile manipulator using a particle filter," in *2006 14th Mediterranean Conference on Control and Automation*, June 2006, pp. 1–6.

[6] A. C. Neto, D. Alves, L. Boncagni, P. J. Carvalho, D. F. Valcarcel, A. Barbalace, G. De Tommasi, H. Fernandes, F. Sartori, E. Vitale, R. Vitelli, and L. Zabeo, "A survey of recent marte based systems," in *Proc. 17th IEEE-NPSS Real Time Conf. (RT)*, 2010, pp. 1–8.

[7] D. Alves, A. C. Neto, D. F. Valcárcel, R. Felton, J. M. López, A. Barbalace, L. Boncagni, P. Card, G. D. Tommasi, A. Goodyear, S. Jachmich, P. J. Lomas, F. Maviglia, P. McCullen, A. Murari, M. Rainford, C. Reux, F. Rimini, F. Sartori, A. V. Stephen, J. Vega, R. Vitelli, L. Zabeo, and K. D. Zastrow, "A new generation of real-time systems in the jet tokamak," in *Real Time Conference (RT), 2012 18th IEEE-NPSS*, June 2012, pp. 1–9.

[8] L. Boncagni, D. Carnevale, G. Ferro, S. Galeani, M. Gospodarczyk, and M. Sassano, "Performance-based controller switching: An application to plasma current control at ftu," in *2015 54th IEEE Conference on Decision and Control (CDC)*, Dec 2015, pp. 2319–2324.