UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

Doctor of Philosophy in Computer Science, Control and Geoinformation

Ph.D coordinator: Prof, Giuseppe F. Italiano

Academic Year: 2016/2017

Ph.D Thesis, XXX cycle

Switching Controllers applications to plasma control and embedded projects using MARTe framework

Supervisor

Ph.D Daniele Carnevale

Author

Giuseppe Ferrò

Co-Supervisors

Ph.D Andrè Cabrita Neto
Ph.D Mario Sassano

To my family

Contents

Acknowledgements **1**

Introduction **2**

1 Switching Controllers **5**
1.1 Introduction 5
1.2 Problem Formulation 6
1.3 Off-line Optimization 8
1.3.1 Numerical approximation of the optimal switching time 9
1.3.2 Controller Resets 10
1.3.3 Feedforward action 13
1.4 On-line Optimization 17
1.5 Conclusions 20

2 FTU Control System **22**
2.1 Introduction 22
2.2 MARTe 22
2.3 Acquisition 24
2.4 Plasma Current Control 27
2.5 Plasma Position Control 29
2.5.1 Future implementations: adaptive strategies 33
2.6 Runaway Control 35
2.6.1 Introduction 35
2.6.2 Control strategies 37
2.6.3 Shattered Pellet Injection 39
2.7 Simulation on FTU cluster 41
2.7.1 Compile new MARTe code 43
2.7.2 JTLogger (Error diagnostics) 43
2.7.3 Cint 44
2.7.4 See results on local machine 44

3 TCV Control System **48**
3.1 Introduction . . . 48
3.2 TCV actuators overview . . . 48
3.3 Control System Overview . . . 49
3.4 Plasma Current Control . . . 51
3.5 Plasma position control . . . 52
3.6 Soft Discharge . . . 55

4 MARTe2 for robotics **59**
4.1 MARTe2 Quality Process . . . 59
4.1.1 Project Organization . . . 60
4.1.2 Quality assurance . . . 62
4.1.3 Software lifecycle implementation . . . 64
4.1.4 Development tools . . . 66
4.1.5 Lesson learned . . . 68
4.1.6 Conclusions . . . 69
4.2 MARTe2 on STM32F4-Discovery . . . 70
4.2.1 Board Configuration . . . 73
4.2.2 Build the Project . . . 76
4.2.3 Embedded tool-kit . . . 79
4.3 Scortec Robotic Arm Controller v1 . . . 82
4.3.1 Scortec Robotic Arm . . . 82
4.3.2 Power Unit . . . 82
4.3.3 STM32F4-Discovery . . . 84
4.3.4 System Performance . . . 88
4.3.5 System Simulation . . . 90
4.3.6 Conclusions . . . 93
4.4 Scortec (Scorbot) Robotic Arm Controller v2 . . . 94
4.4.1 Hardware . . . 94
4.5 CoolCar . . . 103
4.5.1 Remote Control . . . 104
4.5.2 Line Follower . . . 106

5 Conclusions and Recommendations for Further Work **109**

Elenco delle figure **111**

Bibliografia **115**

Acknowledgements

I would like to thank my advisors Ph.D Daniele Carnevale, Ph.D André Neto, Ph.D Mario Sassano and Ph.D Luca Boncagni for all the support received during these three years. I would like to express my gratitude also to the fellows and colleagues of Tor Vergata University, FTU, TCV and F4E for their precious support and from whom I have learned so much. Finally I would to thank my family and my friends for supporting me always.

Introduction

The work described in this thesis has been carried forward within different research groups. The more theoretical part of the work, concerning the study of the switching controllers, has been mostly developed with the help the University of Tor Vergata Automation group under the supervision of Daniele Carnevale and Mario Sassano. In particular we focus on finding a solution of an optimization problem, having a set of given controllers and the possibility to perform switches between each other. The found switching policies, being results of an optimization problem, have the disadvantage to start from the assumption that the plant to be controlled has to be known. Nevertheless, the switching area can be tuned properly on the plant to be controlled without having its perfect knowledge but performing heuristic procedures. Actually, a consistent part of this work is dedicated to the description of switching and hybrid controllers implemented for plasma control. This has been carried forward mostly at the Frascati Tokamak Upgrade (FTU) with the collaboration of the Ph.D Luca Boncagni, the colleague Mateusz Gospodarczyk and the FTU group, under the supervision of Daniele Carnevale. A tokamak consists basically in a toroidal chamber surrounded by magnetic coils used to confine the plasma. In order to trigger the fusion process, the plasma has to be brought at high temperature and pressure then, being strongly influenced by magnetic field, it can be confined thanks to the magnetic coils. The configuration can be different tokamak by tokamak, but usually there is a central solenoid that is used to control the plasma current inducing it magnetically just as it happens in a standard electrical transformer.
During the last years we focus on the improvement of the control performance regarding plasma current and plasma position performing switching and hybrid policies to be together with the standard PID controller actions. Moreover, such strategies have been implemented to improve the runaway beam confinement in suppression experiments aiming at reducing the runaways energy avoiding/reducing the collision with the plasma facing components. The Runaway Electrons (REs), within a tokamak plasma, accelerates to velocities close to the speed of light. They can be very dangerous if uncontrolled because, if the beam is immediately lost, it results in a significant local energy deposition and a consequent critical damage of the plasma facing components (PFC). It is crucial, for the success of tokamaks in general (and in particular for the ITER project), to find a way to suppress the beam and to mitigate

its impact on the machine operation. My support consisted mostly in helping with the implementation of a suppression strategy that consists in a first detection of runaway activity, followed by a slow plasma current shutdown. In many cases we have ascertained the effectiveness of this strategy looking at the diagnostics, that confirms a significant reduction of runaway activity during the current shutdown. Great part of the work developed for FTU has been ported on the control system of the Tokamak a Configuration Variable (TCV) at the EPFL of Lausanne continuing and improving the work performed initially on FTU and TCV by Mateusz Gospodarczyk. The TCV tokamak has a large number of poloidal coils (16 PC) that make TCV very reliable to perform plasma shaping control. This particular configuration has been considered in the implementation of the plasma position control where it could be possible to change dynamically the gain of each poloidal coil improving the effectiveness of the control system.
The implementation of new control system features for FTU leads the side effect of learning how the MARTe framework works. MARTe stands for Multi-threaded Application Real-Time executor and it is a framework for the development of real-time control applications. It is currently employed by several international fusion devices like JET (the current bigger existent tokamak), FTU, COMPASS, etc. A great part of this doctoral period has been spent working to the development of the new version of MARTe (MARTe2) together with the CODAC group of F4E (Fusion For Energy) at Barcelona. A huge revision of the MARTe framework has been carried forward following a specific quality process. The MARTe framework C++ code has been significantly optimized and documented to be compliant with the quality rules and, at the same time, new interesting features have been added to the framework. One of the most remarkable is the decoupling of the framework from any operating system, allowing the possibility of implementing bare-metal MARTe applications. This opened the doors to the possibility to use the high level features of MARTe on embedded architectures. Following this path, some robotic educational projects have been developed using MARTe2 on the STM32F4-Discovery micro-controller implementing and testing at the same time new configurable blocks for the different peripherals.
This thesis is organized as follows:

- The chapter 1 describes the work on the switching control systems. A description of the optimization problems faced has been provided together with the relative optimal and sub-optimal switching policies found.

- The chapter 2 describes the work carried forward to the FTU control system. A description of the new implementations for plasma current and position control has been provided, together with a description of the runaway beam suppression strategy implemented.

- The chapter 3 describes the work implemented on the TCV control system.

- The chapter 4 describes the quality process followed to develop the new version of the MARTe framework. A description of the new main features of MARTe2 has been provided together with its porting on embedded architectures. Follows a description of the educational robotic projects developed using MARTe2 on the STM32F4-Discovery embedded micro-controller.

Chapter 1

Switching Controllers

In this chapter we propose procedures to evaluate switching times, controller resets and vanishing feedforward terms when two or more linear controllers can be switched in order to optimize a quadratic cost function of the closed-loop system response in case of linear plants. Numerical and closed-form solutions are provided. Simulation results are presented to show the effectiveness of the proposed strategies.

1.1 Introduction

The analysis of the properties of switching controllers for the stabilization of linear and nonlinear plants has attracted the attention of many researchers in the last years [4, 6, 8, 10]. This interest is essentially motivated by the suitability of such controllers that can, for instance, adjust themselves depending on external inputs or modifications in the desired operating conditions [1], [2].

Interesting studies on switched integrator control schemes have been performed in [11] and [13], where it is shown how switching policies allow to circumvent limitations of linear controllers, yielding smaller overshoots and consequently improving transient response, while other works have provided switching strategies based on the optimization of certain cost functionals [3]. Stability properties must also be considered in order to ensure asymptotic stability of the resulting closed loop system. Towards this end, intensive research effort has been devoted to analyze and characterize the improvement due to the reset of integrators and of first-order resets element such as in [12] and references therein, hence designing reset strategies that enforce, in addition, an improved $\mathcal{L}_2$ gain.

In this section we recall the approach proposed in [13] but considering different solutions to the problem of finding the optimal switching times for linear controllers

within a feedback interconnection of linear plants, to obtain zero tracking error at steady-state. Furthermore, a numerical (off-line) approach is provided to allow a multi-switches optimal solution unlike the single switche in [13]. Moreover, herein we show that the transient performances can be largely improved allowing a controller state optimal reset at the switching time instants. We also discuss and show the results achieved adding a vanishing feedforward signal to the control input with optimal initial conditions in order to improve the performance or to be compliant with bump-less type requirements.
In section 1.2 we nucleate the problem formulation where general LTI controllers and input references that can be generated by LTI systems (polynomial, exponential and sinusoidal functions) are considered under certain assumptions [14, 15, 9].

In section 1.3 we discuss an *off-line* multi-switch optimization strategy aiming at computing the optimal switching times between two controllers in order to minimize a quadratic cost functional, which is related to the $\mathcal{L}_2$ gain of the tracking error. The optimization is then improved by allowing optimal reset of the controller state at the switching times and, subsequently, by introducing and optimizing the reset values of a vanishing feedforward signal $w(t)$.

In section 1.3 we improve the *on-line* single-switch optimization approach discussed in [13] by introducing the possibility of optimizing the internal state of the controller and to suitably design the feedforward signal $w(t)$ at switching times.

1.2 Problem Formulation

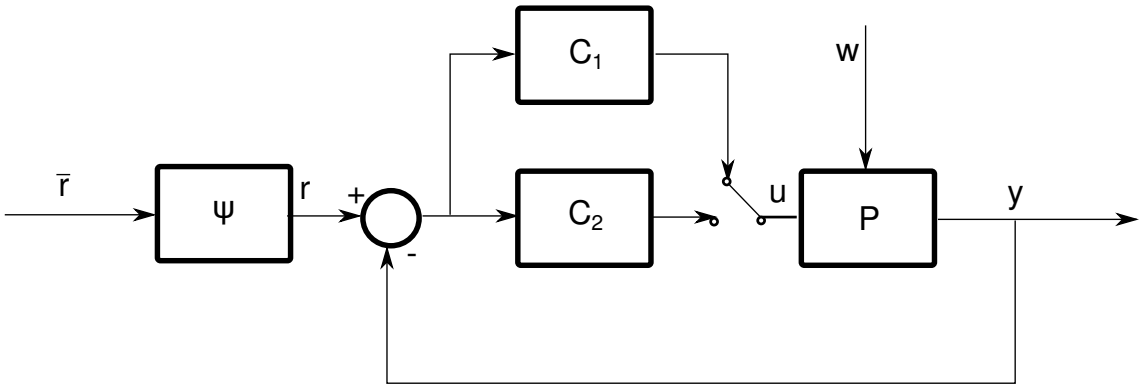

Figure 1.1: The considered system block scheme. It is possible to switch a pre-defined number of time instants N across two pre-defined controllers which ensure asymptotic stability of the closed-loop system.

We consider the scenario in which a plant $\mathcal{P}$ is interconnected, according to the scheme in Figure (1.1), to a controller $\mathcal{C}$ that consists of a set of controllers, all of them ensuring asymptotic stability of the closed-loop system in the presence of an

external reference input trajectory. Before the formal definition of the problem, the task may be informally stated as follows: determine the time switching policy of the controllers that minimizes a cost functional depending on the tracking error with respect a pre-designed desired output trajectory. The optimal switching strategy is sought for within the framework defined by the following standing assumptions.

Denoting with $\overline{r}(t)$ the desired steady state output response, we let the control scheme be driven by a reference signal $r(t)$ in order to provide a smoothed desired reference. In other words, we define $r(t) = \overline{r}(t) + \psi(t)$, with $\psi : \mathbb{R} \to \mathbb{R}$ any (Laplace transformable) smooth function chosen such that

$$\lim_{t\to\infty} \psi(t) = 0$$

. Note that the definition of the function ψ ensures that the desired response r asymptotically converges to $\overline{r}$, thus recovering asymptotically the requirements for the closed-loop system. In addition, we limit the set of admissible control strategies to those satisfying the following assumption.

Assumption 1. *The maximum number of switches among the controllers is defined* a priori *and denoted by* N. ◦

Assumption 2. *The controllers ensure asymptotic tracking of the reference signal* $r(t)$. ◦

Assumption 3. *The closed-loop system is reachable and observable with any of the considered controllers.* ◦

Let the plant and the controller be described as

$$\mathcal{P} : \begin{cases} \dot{x}_p &= A_p\, x_p + B_{p,u}\, u + B_{p,w}\, w \\ y &= C_p\, x_p + D_{p,u}\, u + D_{p,w}\, w \end{cases} \tag{1.2.1}$$

$$\mathcal{C}_q : \begin{cases} \dot{x}_c &= A_{c,q}\, x_c + B_{c,q}\, \overline{e} \\ u &= C_{c,q}\, x_c + D_{c,q}\, \overline{e} \end{cases} \tag{1.2.2}$$

with $q = 1, 2, \ldots N$, where $\overline{e} = \overline{r} - y$ is the tracking error, $x_p \in \mathbb{R}^{n_p}$, $x_c \in \mathbb{R}^{n_c}$. The dynamics of the closed loop system $\mathcal{W}$ with state $x_{cl} = [x_c', \quad x_p']' \in \mathbb{R}^n$ are

$$\mathcal{W} : \begin{cases} \dot{x}_{cl} &= A_{cl,q}\, x_{cl} + B_{cl,r,q}\, r + B_{cl,w,q}\, w \\ y &= C_{cl,q}\, x_{cl} + D_{cl,r,q}\, r + D_{cl,w,q}\, w \end{cases}$$

Consider the following state space realization of the system that generates the smoothed tracking error $r(t)$,

$$\dot{x}_r = A_r x_r, \quad r = C_r x_r, \tag{1.2.3}$$

with $x_r \in \mathbb{R}^{n_r}$. Note that $e = r - y$ is the smoothed tracking error given by

$$e = C_r x_r - C_{cl} x_{cl},$$

we can define a state-space realization of the smoothed tracking error dynamics with state $x_e = T\,[x'_{cl}, x'_r]'$, for a given map T specified in the following, with flow dynamics

$$\mathcal{E}_{\text{flow}} : \begin{cases} \dot{x}_e &= A_{e,q} x_e, \\ \dot{q} &= 0, \\ \dot{\tau} &= 1, \end{cases} \tag{1.2.4}$$

where $A_{e,q}$ is a state-space realization of the tracking error $e(t)$ when the system is being controlled by C_q. Note that by Assumption 2 the non-resonance Francis condition [5] is satisfied. Assuming that N switches are allowed among two controllers at selectable switching times $[t_1,\, t_2, \ldots, t_N]$, the jump map can be described by

$$\mathcal{E}_{\text{jump}} : \begin{cases} x_e^+ &= G(x_e) \\ q^+ &= \text{mod}(q, 3) + 1 \\ \tau^+ &= \tau \end{cases} \tag{1.2.5}$$

where the hybrid are $[x_e(t,k),\, q(t,k),\, \tau] \in \mathbb{R}^{n+m} \times \{1,\, 2,\, \ldots N\} \times \mathbb{R}_{\geq 0}$ and assuming $q(0) = 1$. The jump map $G(\cdot)$ is designed next as part of the proposed solution. The flow and jump sets are defined as

$$\begin{aligned} \mathcal{C} = \{(x_e,\, q,\, \tau) \,\in\, & \mathbb{R}^{n+m} \,\times \{1, \ldots N\} \,\times\, \mathbb{R}_{\geq 0} \,: \\ & (q = 1 \wedge \tau \in [t_{2r}, t_{2r+1}],\, r \in \mathbb{N} : 2r + 1 \leq N) \quad \vee \\ & (q = 2 \wedge \tau \in [t_{2r+1}, t_{2r+2}],\, r \in \mathbb{N} : 2r + 2 \leq N)\} \end{aligned}$$

and

$$\begin{aligned} \mathcal{D} = \{(x_e,\, q,\, \tau) \,\in\, & \mathbb{R}^{n+m} \,\times \{1,\, 2\} \,\times\, \mathbb{R}_{\geq 0} \,: \\ & (q = 1 \wedge \tau \notin [t_{2r}, t_{2r+1}],\, r \in \mathbb{N} : 2r + 1 \leq N) \quad \vee \\ & (q = 2 \wedge \tau \notin [t_{2r+1}, t_{2r+2}],\, r \in \mathbb{N} : 2r + 2 \leq N)\} \end{aligned}$$

respectively. Note that by Assumption 2 the flow dynamics are such that $x_e(t)$ converges asymptotically to zero for $t \longrightarrow \infty$, in other words the matrices $A_{e,q}$ are Hurwitz. The jumps, being limited to N by Assumption 1, do not hinder closed-loop asymptotic stability. In the following sections a series of different problems are discussed and solutions proposed.

1.3 Off-line Optimization

In this section we focus on the offline optimization in order to calculate a priori the set of switching time instants to minimize a given cost functional. Towards this end consider the following problem.

Problem 1 Consider the hybrid system (1.2.4)-(1.2.5) and the scheme in Figure 1.1 with $w(t) = 0$ and suppose that Assumption 1, 2 and 3 hold. The problem of designing the optimal multiple-switch controller consists in determining the switching times $t_k^\star \in \mathbb{R}$ with $k = 1, 2, \dots N$ that minimize

$$J(t) = \int_0^\infty x_e(\tau)' Q x_e(\tau)\ d\tau = \sum_{k=0}^{N} \int_{t_k}^{t_{k+1}} x_e(\tau)' Q x_e(\tau)\ d\tau \tag{1.3.1}$$

where $t_0 = 0$ and $t_{N+1} = \infty$, with $Q = Q',\ Q \succeq 0$.

Remark 1. *In the simulations of this paper we select the change of coordinate matrix T defining x_e such that its first two components are e and $\dot{e}$: selecting Q in (1.3.1) as $Q = diag[\gamma_1, \gamma_2, 0, \dots, 0]$, with $\gamma_1 > 0$ and $\gamma_2 > 0$, J is the $\mathcal{L}_2$-norm of the smoothed error and its first derivative in such a way that both the tracking error and its oscillations (overshoot), respectively, are weighted.*

1.3.1 Numerical approximation of the optimal switching time

As derived in [13], since $A_{e,q}$ are Hurwitz and by Assumption 3 the closed-loop system is observable, by LaSalle's theorem there exist symmetric and positive definite matrices $P_q = P_q'$, $P_q \succ 0$ such that

$$V_q = x_e' P_q x_e, \tag{1.3.2a}$$

$$A_{e,q}' P_q + P_q A_{e,q} = -Q, \tag{1.3.2b}$$

with $Q = Q'$, $Q \succeq 0$ selected according to the optimization objectives discussed above.

Integrating both sides of the latter equations between the switching time instants we obtain

$$V_{q_k}(t_k) - V_{q_{k+1}}(t_k) = \int_{t_k}^{t_{k+1}} x_e'(\tau)\, Q\, x_e(\tau)\, d\tau \tag{1.3.3}$$

and, due to Assumption 2, $V_{q_{N+1}}(t_{N+1}) = V_{q_{N+1}}(\infty) = 0$, we can rewrite the cost functional (1.3.1) as:

$$J(t) = \sum_{k=0}^{N} x_e(t_k)'(P_{q_{k+1}} - P_{q_k}) x_e(t_k), \tag{1.3.4}$$

letting $P_{q_0} = 0$, $t_0 = 0$. Considering that $x_e(t_k) = (\prod_{i=1}^{k} e^{A_{e,q_i}(t_i - t_{i-1})}) x_0$, it is evident that J is a transcendental function of the switching times t_i highly increasing the difficulties in finding a closed-form solution for Problem 1 even for very simple systems. Nevertheless, the analytical expression of the cost J given in (1.3.4) can be easily

computed in closed-form and evaluated on a grid of switching times. Certainly, this numerical approach suffers from the *curse of dimensionality* with respect to N and the quantization of the switching times. However, note that the time interval to be considered depends on the time constant of the closed-loop systems, so basically the discretized time range is proportional to the largest time constant for the maximum number of switches, e..g $N\,4/\lambda_{\min}$ where $\lambda_{\min}$ is the eigenvalue belonging to the spectrum of $A_{e,q}$ closer to the imaginary axis. The additional constraint $t_k > t_{k-1}$ reduces the points in the grid.

In the following simulation we consider the scheme in Figure 1.1 with a standard second order plant

$$P(s) = k\frac{\omega_n^2}{s^2 + 2\delta\omega_n s + \omega_n^2} \tag{1.3.5}$$

with $k = 36$, $w_n = 2\pi/0.3$, $\delta = 0.8$. The signal to be tracked is $r(t) = 1 - e^{-\sigma t}$ with $\sigma = 500$. The plant is assumed to be controlled by two PI (proportional-integral controllers) named C_1 and C_2, both ensuring asymptotic tracking of constant references which have the same proportional gain $K_p = 0.1$ but different integral gains $K_{i,1} = 0.2$, $K_{i,2} = 3$. In this simulation we have assumed two possible switches, from C_1 to C_2 and then to C_1 again. The simulation has been performed on a grid of switching times t_1, t_2 with $t_2 > t_1$ with a resolution of $10^{-2}s$. Figure 1.2 shows the tracking performances using the controller C_1 (cyan) and C_2 (magenta) without switches, with a single (green) and double (blue) switch. Figure 1.3 shows the value of the cost functional J in case of two switches. The two (approximated) optimal switching times yielding the lowest value of J are $t_1 = 0.1s$ and $t_2 = 0.14s$. For the single switch case the (approximated) optimal switching time is $t_1 = 0.2s$.

1.3.2 Controller Resets

Consider now the possibility of resetting the state of the controller at switching times.

Problem 2 Under the hypotheses of Problem 1, design the optimal switching times $t_k^\star \in \mathbb{R}$ with $k = 1, 2, \ldots N$ and the optimal reset values $x_c^\star(t_k^\star)$ that minimize the cost functional (1.3.1).
A solution to the this problem is proposed in the following proposition.

Proposition 1 A numerical approximated solution to Problem 2 can be obtained by evaluating on a time grid the cost functional (1.3.4) rewritten substituting $x_e = Tx_a$, $x_a = [x_c'\ x_p'\ x_r']$ with $\dot{x}_a = A_{a,q}x_a = TA_{e,q}T^{-1}x_a$, and replacing $x_c(t_k)$ with $x_c^\star(t_k)$ where

$$\vec{x_c^\star} = M_o^{-1}B_o, \tag{1.3.6}$$

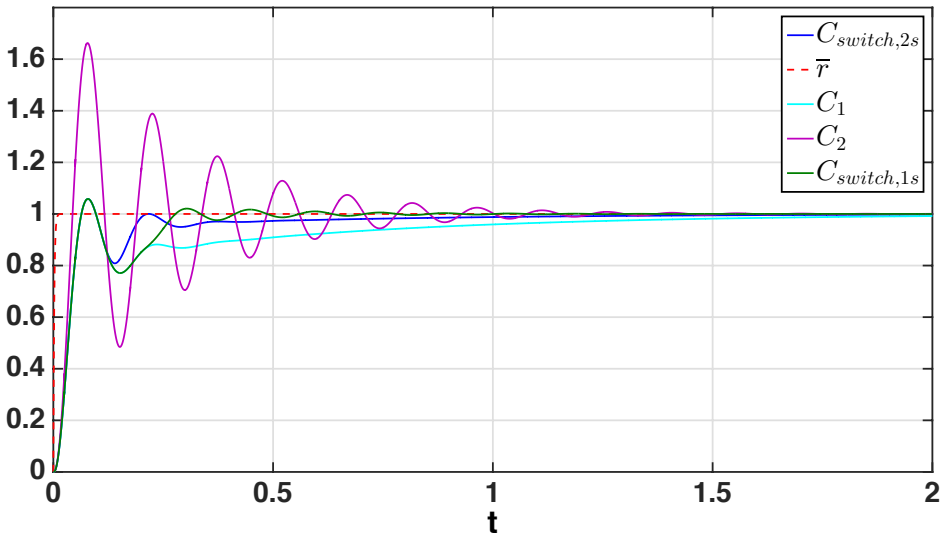

Figure 1.2: Tracking performances using the controller C_1 (cyan) and C_2 (magenta) without switches, with a single (green) and double (blue) switch.

with $\vec{x}_c^{\star} = [x_c(0)'^{\star}, x_c(t_1)'^{\star}, \dots x_c(t_N)'^{\star}]'$. The matrix M_o is constant, symmetric, and matrix B_o is a linear combination[1] of $[x_p(0), x_r(0), x_p(t_1), x_r(t_1), \dots, x_r(t_N)]$.

Proof. We can rewrite the functional cost defined in (1.3.4) as follows

$$
\begin{aligned}
J(t) &= \sum_{k=0}^{N} x_e^+(t_k)' P_{q_{k+1}} x_e'^+(t_k) - \sum_{k=0}^{N-1} x_e(t_{k+1})' P_{q_{k+1}} x_e(t_{k+1}) \\
J(t) &= \sum_{k=0}^{N} x_a^+(t_k)' T_{q_{k+1}}' P_{q_{k+1}} T_{q_{k+1}} x_a'^+(t_k) - \\
&\sum_{k=0}^{N-1} x_a^+(t_k)' e'^{A_{q_k}(t_{k+1}-t_k)} T_{q_{k+1}}' P_{q_{k+1}} T_{q_{k+1}} e^{A_{q_k}(t_{k+1}-t_k)} x_a(t_k)^+ \\
J(t) &= \sum_{k=0}^{N} x_a^+(t_k)' \tilde{P}_{q_{k+1}} x_a^+(t_k)
\end{aligned}
\tag{1.3.7}
$$

with $x_a^+ = [x_c'^+ \; x_p' \; x_r']$ namely the reset state and

$$
\begin{aligned}
\tilde{P}_{q_{k+1}} = {} & T_{q_{k+1}}' P_{q_{k+1}} T_{q_{k+1}} - \\
& (T_{q_{k+1}} e^{A_{q_k}(t_{k+1}-t_k)})' P_{q_{k+1}} T_{q_{k+1}} e^{A_{q_k}(t_{k+1}-t_k)}
\end{aligned}
$$

for $k = 0 \dots N-1$. If $k = N$: $\tilde{P}_{q_{N+1}} = T_{q_{N+1}}' P_{q_{N+1}} T_{q_{N+1}}$. We now rewrite the $k-$th

[1] The algorithm to evaluate such matrices can be downloaded in [16].

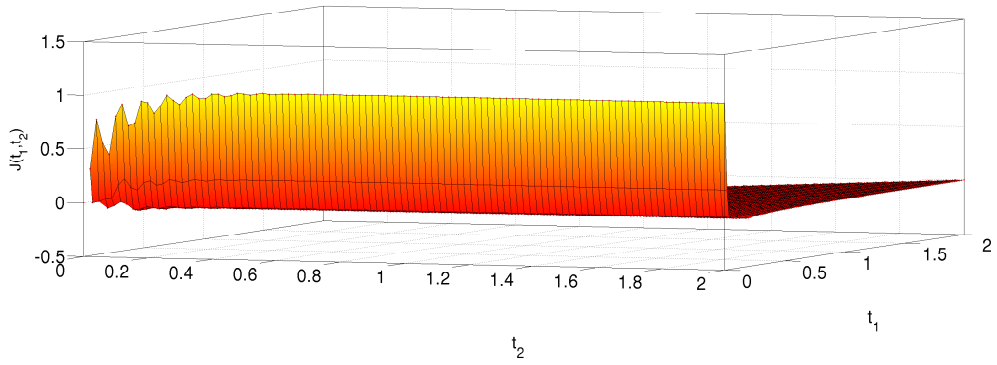

Figure 1.3: The cost functional (1.3.1) in case of two switching times with a resolution of $10^{-2}s$.

therm of (1.3.7) as

$$J_k = x_a^+(t_k)' \tilde{P}_{q_{k+1}} x_a^+(t_k) = \tag{1.3.8}$$

$$[x_c^+(t_k)'\, x_p(t_k)'\, x_r(t_k)'] \begin{bmatrix} \tilde{P}_{cc,q_{k+1}} & \tilde{P}_{cp,q_{k+1}} & \tilde{P}_{cr,q_{k+1}} \\ \tilde{P}'_{cp,q_{k+1}} & \tilde{P}_{pp,q_{k+1}} & \tilde{P}_{pr,q_{k+1}} \\ \tilde{P}'_{cr,q_{k+1}} & \tilde{P}'_{pr,q_{k+1}} & \tilde{P}_{rr,q_{k+1}} \end{bmatrix} \begin{bmatrix} x_c^+(t_k) \\ x_p(t_k) \\ x_r(t_k) \end{bmatrix} \tag{1.3.9}$$

Note that the term $x_r(t_k)'\tilde{P}_{rr,q_k} x_r(t_k)$ can be neglected because it will disappear in the partial derivative of J_k with respect x_c^+. Thus we obtain

$$\begin{aligned} J_{k,x_c^+} = & x_c^+(t_k)'\tilde{P}_{cc,q_{k+1}} x_c^+(t_k) + 2x_c^+(t_k)'\tilde{P}_{cp,q_{k+1}} x_p(t_k) + \\ & 2x_c^+(t_k)'\tilde{P}_{cr,q_{k+1}} x_r(t_k) + x_p(t_k)'\tilde{P}_{pp,q_{k+1}} x_p(t_k) + \\ & 2x_p(t_k)'\tilde{P}_{pr,q_{k+1}} x_r(t_k), \end{aligned} \tag{1.3.10}$$

whereas the term $x_p(t_k)$ is given by

$$x_p(t_k) = e_{pc,k} x_c^+(t_{k-1}) + e_{pp,k} x_p(t_{k-1}) + e_{pr,k} x_r(t_{k-1}), \tag{1.3.11}$$

where the vectors $e_{p\chi,k}$ are the blocks of the one-switch flow monodromy matrix to be multiplied for x_p.

$$e^{A_{a,q_k}(t_k - t_{k-1})} = \begin{bmatrix} e_{cc,k} & e_{cp,k} & e_{cr,k} \\ e_{pc,k} & e_{pp,k} & e_{pr,k} \\ e_{rc,k} & e_{rp,k} & e_{rr,k} \end{bmatrix}.$$

Based on (1.3.11), $x_p(t_k)$ can be written as a linear combination of $x_r(t_{k-1})$ and $x_p(t_{k-1})$, assumed to be known. To conclude, the minimization of (1.3.10) with respect to x_c^+ is trivially obtained solving the linear system $\partial J_k / \partial x_c^+ = 0$ and the same holds true for (1.3.7). □

Follows a deepening on the method to compute the matrices Mo and Bo.
From 1.3.11 we can write $x_p(t_k)$ in the following form:

$$\begin{aligned} x_p(t_k) = &\begin{bmatrix} e_{pp,k}e_{pp,k-1}\dots e_{pp,2}e_{pc,1} & e_{pp,k}e_{pp,k-1}\dots e_{pp,3}e_{pc,2} & \dots & e_{pc,k} & 0 & \dots & 0 \end{bmatrix} \vec{x}_c^{+} + \\ &\begin{bmatrix} e_{pp,k}e_{pp,k-1}\dots e_{pp,2}e_{pr,1} & e_{pp,k}e_{pp,k-1}\dots e_{pp,3}e_{pr,2} & \dots & e_{pr,k} & 0 & \dots & 0 \end{bmatrix} \vec{x}_r \\ =&M_{e,k}\vec{x}_c^{+} + M_{r,k}\vec{x_r} \end{aligned}$$

with $\vec{x}_c^{+} = [x_c^{+}(0)\ x_c^{+}(t_1)\dots x_c^{+}(t_k)\dots x_c^{+}(t_N)]$ and $\vec{x}_r^{+} = [x_r(0)\ x_r(t_1)\dots x_r(t_k)\dots x_r(t_N)]$.
Moreover noting that:

$$\begin{aligned} x_c^{+}(t_k) &= \begin{bmatrix} 0_1 & \dots & I_k & \dots & 0_N \end{bmatrix} \vec{x}_c^{+} = I_{a,c,k}\vec{x}_c \\ x_r^{+}(t_k) &= \begin{bmatrix} 0_1 & \dots & I_r & \dots & 0_N \end{bmatrix} \vec{x}_r = I_{a,r,k}\vec{x}_r \end{aligned}$$

Using the latter definitions, we can rewrite 1.3.10 in function of $\vec{x}_c^{+}$:

$$\begin{aligned} J_{k,\vec{x}_c^{+}} =& \vec{x}_c^{+}(t_k)' I'_{a,c,k}\tilde{P}_{cc,q_{k+1}} I_{a,c,k}\vec{x}_c^{+}(t_k) + 2\vec{x}_c^{+}(t_k)' I'_{a,c,k}\tilde{P}_{cp,q_{k+1}}(M_{e,k}\vec{x}_c^{+} + M_{r,k}\vec{x_r}) + \\ & 2\vec{x}_c^{+}(t_k)' I'_{a,c,k}\tilde{P}_{cr,q_{k+1}} I_{a,r,k}\vec{x}_r + (M_{e,k}\vec{x}_c^{+} + M_{r,k}\vec{x_r})'\tilde{P}_{pp,q_{k+1}}(M_{e,k}\vec{x}_c^{+} + M_{r,k}\vec{x_r}) + \\ & 2(M_{e,k}\vec{x}_c^{+} + M_{r,k}\vec{x_r})'\tilde{P}_{pr,q_{k+1}} I_{a,r,k}\vec{x}_r(t_k), \end{aligned} \tag{1.3.12}$$

Now we can compute the partial derivative $\frac{\partial J_k}{\partial \vec{x}_c^{+}}$ obtaining:

$$\begin{aligned} \frac{\partial J_k}{\partial \vec{x}_c^{+}} = &[I'_{a,c,k}\tilde{P}_{cc,q_{k+1}} I_{a,c,k} + 2I'_{a,c,k}\tilde{P}_{cp,q_{k+1}} M_{e,k} + M'_{e,k}\tilde{P}_{pp,q_{k+1}} M_{e,k}]\vec{x}_c^{+} + \\ &[I'_{a,c,k}\tilde{P}_{cp,q_{k+1}} M_{r,k} + I'_{a,c,k}\tilde{P}_{cr,q_{k+1}} I_{a,r,k} + M'_{e,k}\tilde{P}_{pp,q_{k+1}} M_{r,k} + M'_{e,k}\tilde{P}_{pr,q_{k+1}} I_{a,r,k}]\vec{x_r} \end{aligned}$$

From 1.3.4, $\frac{\partial J}{\partial \vec{x}_c^{+}} = \sum_{k=0}^{N} \frac{\partial J_k}{\partial \vec{x}_c^{+}}$ holds, thus the matrices Mo and Bo will be:

$$Mo = \sum_{k=0}^{N} [I'_{a,c,k}\tilde{P}_{cc,q_{k+1}} I_{a,c,k} + 2I'_{a,c,k}\tilde{P}_{cp,q_{k+1}} M_{e,k} + M'_{e,k}\tilde{P}_{pp,q_{k+1}} M_{e,k}] \tag{1.3.13}$$

$$Bo = -\sum_{k=0}^{N} [I'_{a,c,k}\tilde{P}_{cp,q_{k+1}} M_{r,k} + I'_{a,c,k}\tilde{P}_{cr,q_{k+1}} I_{a,r,k} + M'_{e,k}\tilde{P}_{pp,q_{k+1}} M_{r,k} + M'_{e,k}\tilde{P}_{pr,q_{k+1}} I_{a,r,k}]\vec{x_r} \tag{1.3.14}$$

1.3.3 Feedforward action

Consider now the possibility of employing a feedforward signal $w(t)$ as a further input to the plant P (usually added to the controller output) as depicted in Figure 1.1. This term has been considered to improve the minimization of the cost functional and possibly to meet signal constraints requirements (input-output bumpless). The feedforward signal, since it should be used only to adjust transients induced by controller switching, satisfies the following assumption.

Assumption 4. *The feedforward signal $w(t)$ is generated by an asymptotically stable linear system, i.e. $w(t) = C_w x_w(t)$ with $\dot{x}_w(t) = A_w x_w(t)$, A_w a Hurwitz matrix of suitable dimensions.* ∘

With this assumption, and selecting *a priori* A_w and C_w according to the desired switching transients, we aim at finding the best (reset) state $x_w(t_k)$ that improves the transients and/or satisfy given constraints. A suggested selection of A_w would be a Jordan block of dimension m corresponding to a negative real eigenvalue λ with $C_w = [1, 0, \ldots, 0]$, yielding $w(t) = e^{\lambda\tau}[1, \tau, \tau^2, \ldots, \tau^{m-1}]x_w(t_k)$ with $\tau = t - t_k$. Specifically, in the following problem we consider the feedforward signal in order to improve performances and satisfy the plant output bumpless constraints that, in case of plants with instantaneous input-output relation (bi-proper transfer function), generalize the standard (plant input) bumpless concept [7].

Problem 3 Consider the hypotheses of Problem 1 and determine the switching times $t_k^\star \in \mathbb{R}$ with $k = 1, 2, \ldots N$ and the optimal reset values $[x_c^\star(t_k^\star), x_w^\star(t_k^\star)]$ that minimize the cost functional (1.3.4) and satisfy the following linear constraints

$$E(x_e^+(t_k) - x_e(t_k)) = 0,$$

where $E \in \mathbb{R}^{c \times N}$ is a constant matrix.

In order to appreciate the improvement of the optimal controller resets and the use of the feedforward signal, output tracking performances in case of single switch are shown in simulation of Figure 1.5 where no constraints have been considered ($E \equiv 0$) on the same plant, controllers and reference signal of the previous simulation. The feedforward signal has been added to the controller's output with $A_w = -100$ and $C_w = 1$. Note that if we consider $x_w(t)$ as a part of the aggregate state so that $x_a = [x_c' \; x_w' \; x_p' \; x_r']$.

The solution to Problem 3 trivially derives from Proposition 1 rewriting $x_{c,k}^+$ as $[x_c'^+ \; x_c'^+]'$ as the reset state to be optimized at switching times t_k. We can adopt the standard optimization procedures adding the linear constraints, thus we exploit the Lagrange multipliers expressing the cost functional as

$$\begin{aligned} J &= J_A + J_B, \\ J_B &= \sum_{k=1}^{N} \lambda_k' E(x_e^+(t_k) - x_e(t_k)), \end{aligned}$$

with J_A defined in (1.3.4). Note that λ_k is typically a vector with c_k elements for $k = 1 \ldots N$.

In order to visualize the dependencies of the cost functional from the reset states that we are going to optimize, we express J in function of the aggregate state $x_a =$

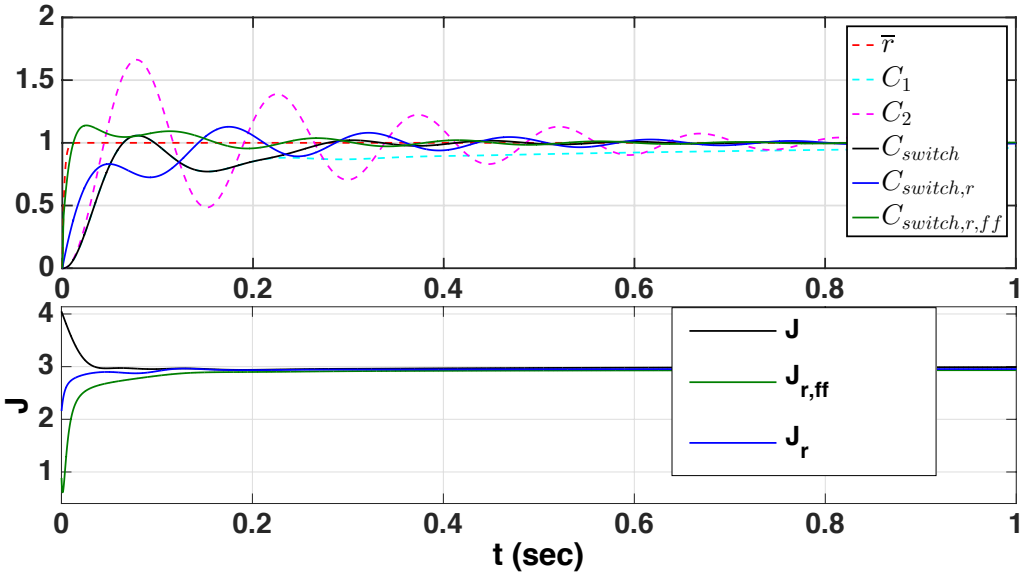

Figure 1.4: (Top) Tracking performances: (red dashed) reference signal $r(t)$, the C_1 (cyan) and C_2 (magenta) controllers (no switch). Single-switch (blue), two-switch (green). (Bottom) Cost functionals J: only the switching time optimization (black), adding the controller reset state (blue) together with the feedforward (green) signal. The time resolution is $10^{-5}s$.

$[x_c', x_w', x_p', x_r']'$. Thus J_A can be rewritten as (1.3.7), while J_B can be expressed as follows

$$J_B = \sum_{k=1}^{N} \lambda_k' E(T_{q_{k+1}} x_{a,k}^+(t_k) - T_{q_k} x_{a,k}(t_k)).$$

Consider now the part of the cost functional that depends on the states $x^+(t_k) = [x_c^+(t_k)' \; x_w^+(t_k)']'$ for each $k \in [0 \dots N]$.

$$\begin{aligned} J_{x^+(t_k)} = & (x^+(t_k)' \tilde{E}_{cw,k}' + x_p(t_k)' \tilde{E}_{p,k}') \lambda_k - \\ & (x^+(t_k)' \tilde{E}_{cw,k}'^+ + x_p(t_k)' \tilde{E}_{p,k}'^+) \lambda_{k+1} + J_{A,x^+(t_k)} \end{aligned} \tag{1.3.15}$$

where we have denoted with

$$\begin{aligned} \tilde{E}_k &= E T_{q_{k+1}} \\ \tilde{E}_k^+ &= E T_{q_{k+1}} e^{A_{a,q_{k+1}}(t_{q_{k+1}} - t_k)} \end{aligned}$$

and we have divided the rows of the latter matrices which will be post-multiplied by $x^+{}'$, x_p' and x_r' respectively such that $\tilde{E}_k = [\tilde{E}_{cw,k}' \; \tilde{E}_{p,k}' \; \tilde{E}_{r,k}']'$. $J_{A,x^+(t_k)}$ has already be defined in 1.4.2. For $k = 0$ and $k = N$ the formula differs as

$$\begin{aligned} J_{x^+(0)} &= J_{A,x(0)} - (x(0)' \tilde{E}_{cw,0}'^+ + x_p(0)' \tilde{E}_{p,0}'^+) \lambda_1 \\ J_{x^+(t_N)} &= J_{A,x^+(t_N)} + (x^+(t_N)' \tilde{E}_{cw,N} + x_p(t_N)' \tilde{E}_{p,N}) \lambda_N \end{aligned}$$

By extracting the part of J depending on the Lagrange multipliers, one has

$$J_{\lambda_k}=\lambda_k'[(\tilde{E}_{k,cw}x(t_k)^+ + \tilde{E}_{k,p}x_p(t_k) + \tilde{E}_{k,r}x_r(t_k))- (\tilde{E}^+_{k-1,cw}x(t_{k-1})^+ + \tilde{E}^+_{k-1,p}x_p(t_{k-1}) + \tilde{E}^+_{k-1,r}x_r(t_{k-1}))] \quad (1.3.16)$$

where we have divided the columns of $\tilde{E}_k$ and $\tilde{E}^+_k$ which will be post-multiplied by x^+, x_p and x_r respectively such that $\tilde{E}_k = [\tilde{E}_{cw,k}\ \tilde{E}_{p,k}\ \tilde{E}_{r,k}]$.
The optimal solution will be achieved imposing all the partial derivatives of $J_{x^+(t_k)}$ and J_{λ_k} with respect to x^+ and λ equal to zero. $x_p(t_k)$ can be expressed as follows

$$x_p(t_k) = e_{pc,k}x_c^+(t_{k-1}) + e_{pw,q}x_w^+(t_{k-1})+ \quad (1.3.17a)$$

$$e_{pp,k}x_p(t_{k-1}) + e_{pr,k}x_r(t_{k-1}). \quad (1.3.17b)$$

Then, $x_p(t_k)$ can be written as a linear combination of $x^+_{cw}(t_{k-1})$, $x_r(t_{k-1})$ and $x_p(t_{k-1})$ which can be recursively expressed as a combination of the states at the previous switch. The optimal values can then be obtained solving a set of linear equations since the partial derivatives of (1.3.15)-(1.3.16) and $x_p(t_k)$ with respect to x^+ and λ are linear.

The optimal controller and feedforward reset values can then be obtained solving a set of linear equations since the partial derivatives of J and $x_p(t_k)$ with respect to x^+ and λ are linear.

In order to appreciate the bumpless effect, a simulation has been performed comparing the optimization with and without solving the constraints in Problem 3. Consider the second order plant

$$P(s) = \frac{s^2 + 20s + 100}{s^2 + 2\delta\omega_n s + \omega_n^2}$$

with $\omega_n = 2\pi/5$, $\delta = 0.8$ and the scheme in Figure 1.1 with C_1 and C_2 chosen as PI controllers with different proportional gains $K_{p,1} = 0.1$, $K_{p,2} = 0.3$, yielding an input/output discontinuity, and different integral gains $K_{i,1} = 0.2$, $K_{i,2} = 3$. The feedforward signal has been added on the controller's output and chosen as $w(t) = x_w(t_k)e^{-250(t_k-t_{k-1})}$ (with $x_w(t_k)$ to be computed by the optimization algorithm) and the reference signal to be tracked has been selected as $r(t) = 1 - e^{-25t}$. Note that we have a plant with a non-zero instantaneous input-output map and the control input jumps at switching times, due to the different proportional gains of the controllers, in the absence of the feedforward signal. The simulated switching controller is single-switch and in the bumpless case we have imposed the constraint of a continuous output ($E = [1\ 0\ \ldots 0]$). Figure 1.5 shows the output with and without bumpless constraint optimization. Note that without the bumpless constraint the system output have is discontinuous at the switching time, while imposing the bumpless constraint the system output does not jump whereas a discontinuity continuous to appear in output derivate (which has not been constrained).

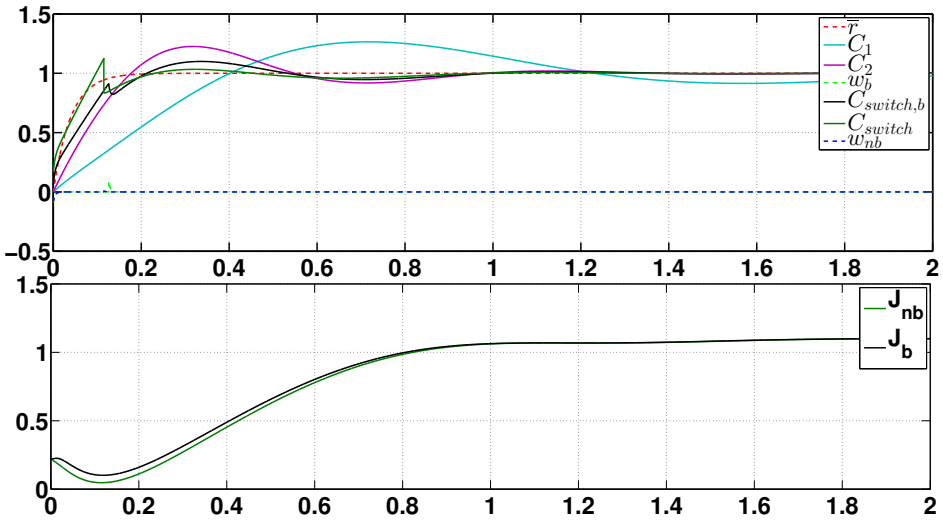

Figure 1.5: On top, a comparison of the system outputs obtained with and without bumpless constraint. In red is depicted the reference signal to be tracked, the light blue and purple trajectories are the output achieved employing separately C_1 and C_2 controllers respectively. The green line is the output achieved with a single-switch controller from C_1 to C_2 without bumpless constraint, the black one is the output achieved by the single-switch controllers with bumpless constraint. On bottom, the cost functionals for the two different cases: the green line obtained without bumpless constraint and the black one obtained introducing and optimizing with bumpless constraint $(e(t_1^+) - e(t_1) = 0$. The time resolution is $10^{-5}s$.

1.4 On-line Optimization

In the previous section we have discussed how to compute the optimal switching time instants between two controllers, their resets and the feedforward signal in order to minimize the functional cost J. The advantage of the former solutions is that multiple switches are consider at once, whereas the drawback is that only numerically approximation of the optimal solution, not in closed form, is provided. In the current section, we propose a sub-optimal single-switch closed-form solution extending the result proposed in [13] by adding the controller resets and the employing the feedforward signal $w(t)$.

Problem 4 Consider the hypotheses of Problem 1 assuming that a single switch is allowed. Design the optimal *single* switching time $t^\star \in \mathbb{R}$ and the controller reset $x_c^\star(t^\star)$ that minimize the cost functional J in (1.3.1).

Proposition 2 Consider Problem 4 and let the matrix $\overline{P}_q = T_q' P_q T_q$ be partitioned such that

$$J_k = x_a^+(t_k)' \tilde{P}_{q_{k+1}} x_a^+(t_k) = \tag{1.4.1}$$

$$[x_c^+(t_k)'\, x_p(t_k)'\, x_r(t_k)'] \begin{bmatrix} \tilde{P}_{cc,q_{k+1}} & \tilde{P}_{cp,q_{k+1}} & \tilde{P}_{cr,q_{k+1}} \\ \tilde{P}'_{cp,q_{k+1}} & \tilde{P}_{pp,q_{k+1}} & \tilde{P}_{pr,q_{k+1}} \\ \tilde{P}'_{cr,q_{k+1}} & \tilde{P}'_{pr,q_{k+1}} & \tilde{P}_{rr,q_{k+1}} \end{bmatrix} \begin{bmatrix} x_c^+(t_k) \\ x_p(t_k) \\ x_r(t_k) \end{bmatrix} \tag{1.4.2}$$

and pick the controller resets as

$$x_c^\star(t^\star) = -(\overline{P}_{cc,2})^{-1}(\overline{P}_{cp,2} x_p(t^\star) + \overline{P}_{cr,2} x_r(t^\star)), \tag{1.4.3}$$

with

$$\begin{aligned} t^\star = \arg\min_t \{ & x_a'(t) R' \overline{P}_2 x_a^+(t) + x_a'^+(t) \overline{P}_2 R x_a(t) \\ & - x_a'(t) T' Q T x_a(t) \geq 0 \}, \end{aligned} \tag{1.4.4a}$$

$$R = \begin{bmatrix} R_{cc} & R_{cp} & R_{cr} \\ R_{pc} & R_{pp} & R_{pr} \\ R_{rc} & R_{rp} & R_{rr} \end{bmatrix}, \tag{1.4.4b}$$

and

$$\begin{aligned} R_{cc} &= -\overline{P}_{cc,2}^{-1} \overline{P}_{cp,2} B_p (C_c - D_c L^{-1} D_p C_c), \\ R_{cp} &= -\overline{P}_{cc,2}^{-1} \overline{P}_{cp,2} (A_p - B_p D_c L^{-1} C_p), \\ R_{cr} &= -\overline{P}_{cc,2}^{-1} [\overline{P}_{cp,2} B_p D_c (C_r - L^{-1} D_p D_c C_r) + \overline{P}_{cr,2} A_r], \\ R_{pc} &= B_p (C_{c,1} - D_{c,1} L^{-1} D_p C_{c,1}), \\ R_{pp} &= A_p - B_p D_{c,1} L^{-1} C_p, \\ R_{pr} &= B_p D_{c,1} (C_r - L^{-1} D_p D_{c,1} C_r), \\ R_{rc} &= 0, \\ R_{rp} &= 0, \\ R_{rr} &= A_r, \end{aligned}$$

with $L = (1 + D_p D_{c,1})$. Then, the switching time $t^\star$ solves the Problem 4.

Proof. Following the arguments of Proposition 1, equation (1.3.4) still holds with $N = 1$, then we can rewrite the cost functional J substituting the optimal resets in (1.4.3) with t in place of $t^\star$ as follows

$$\begin{aligned} J_{x_c^\star} &= x_e(t, x_c^\star(t))' P_2 x_e(t, x_c^\star(t)) - x_e'(t) P_1 x_e(t) + V_1(0) \\ &= x_a'^+ T_2' P_2 T_2 x_a^+ - x_a' T_1' P_1 T_1 x_a + V_1(0) \\ &= x_a'^+ \overline{P}_2 x_a^+ - x_a' \overline{P}_1 x_a + V_1(0). \end{aligned} \tag{1.4.5}$$

Note that $V_1(0) = x_a(0)\overline{P}_1 x_a(0)$ and $x_a'\overline{P}_1 x_a$ do not depend by $x_c^+(t)$ whereas

$$x_a'^+\overline{P}_2 x_a^+ = [x_c'^+ \; x_p' \; x_r'] \begin{bmatrix} \overline{P}_{cc,2} & \overline{P}_{cp,2} & \overline{P}_{cr,2} \\ \overline{P}_{cp,2}' & \overline{P}_{pp,2} & \overline{P}_{pr,2} \\ \overline{P}_{cr,2}' & \overline{P}_{pr,2}' & \overline{P}_{rr,2} \end{bmatrix} \begin{bmatrix} x_c^+ \\ x_p \\ x_r \end{bmatrix}.$$

Also in this case we can neglect terms depending only on x_p and x_r obtaining

$$x_c'^+\overline{P}_{cc,2} x_c^+ + x_c'^+\overline{P}_{cp,2} x_p + x_c'^+\overline{P}_{cr,2} x_r,$$

whose partial derivative with respect to x_c^+ is rendered zero selecting the controller state reset as in (1.4.3).
We propose then to switch controller whenever the cost function J starts increasing. Exploiting the structure of J as in (1.4.5) and the definition of P_1 and P_2, the condition $\dot{J}(t, x_c^\star(t)) > 0$ rewrites as

$$\dot{x}_a'^+\overline{P}_2 x_a^+ + x_a'^+\overline{P}_2 \dot{x}_a^+ - \dot{x}_a'\overline{P}_1 x_a - x_a'\overline{P}_1 \dot{x}_a > 0. \tag{1.4.6}$$

Given the expression of $x_c^\star$ in (1.4.3) we can rewrite $\dot{x}_a^+ = R x_a$ using the matrix R defined in (1.4.4*b*) that allows to rewrite (1.4.6), nothing that $\dot{x}_{e,1} = A_{e,1}x_{e,1} = TA_{a,1}x_a$, as (1.4.4*a*).

□

Remark 2. *$V_1(0)$ is the value of the cost functional if the system never switches. Then we can easily know if there exist a time for which the functional $J(t)$ is less than $V_1(0)$, i.e. if a switch is going to occur. This transforms into the condition $J(t) < J(\infty)$, or*

$$\begin{aligned} &x_e(t, x_c^\star(t))' P_2 x_e(t, x_c^\star(t))) - x_e(t) P_1 x_e(t) + V_1(0) < V_1(0), \\ &x_e(t, x_c^\star(t))' P_2 x_e(t, x_c^\star(t))) - x_e(t) P_1 x_e(t) < 0. \end{aligned}$$

Rewriting the latter inequality with the aggregate state x_a, the condition for the existence of the switch is

$$x_a'^+\overline{P}_2 x_a^+ - x_a'\overline{P}_1 x_a < 0.$$

Remark 3. *The cost functional (*1.3.1*) may in general admit several (local) minimum points and we provided a solution that generally detects a local minimum. More precisely, we select $t^\star$ defined as* $\arg\min_{t\geq 0}\{\dot{J}(t, x_c^\star(t)) \geq 0\}$.

Remark 4. *Consider now a feedforward signal $w(t) \neq 0$ with a pre-defined dynamic supplying assumption* 4*. It might be a great improvement to compute also the optimal reset state of $w(t)$ with respect our cost functional. Thus with the same approach discussed in Proposition 2, considering the aggregate state $x_a = [x_c' \; x_w' \; x_p' \; x_r']$ and the state to be optimized $x^+ = [x_c'^+ \; x_w'^+]$, it is possible to consider the feedforward signal $w(t)$ to further minimize the cost functional J.*

Remark 5. *The solution introduced in Proposition 2 can be iterated recursively (for a finite number of time N) replacing C_1 with C_2 and vice-versa (or adding new controllers). The advantage is intuitively quite clear because after the first switch it is possible, if convenient, to switch again to the next controller and so on, minimizing the value of the cost functional.*
Note that this "myopic" strategy is quite far from providing the global optimum considering all the N possible switches. Nevertheless, allowing multiple switches using the single-switch procedure in Proposition 2, the cost functional J can be reduced.

Remark 6. *Assumption* 2*, for both Propositions, can be relaxed requiring only that the final controller ensures asymptotic tracking. In fact, if the controller q does not ensure null steady state error we can rewrite (*1.3.2 *) imposing the following*

$$\begin{aligned} V_q(t) &= x_e(t)' P_q(t) x_e(t), \\ \dot{V}_q(t) &= -x_e(t)' Q x_e(t), \\ A'_{e,q} P_q(t) + \dot{P}_q(t) + P_q(t) A_{e,q} &= -Q. \end{aligned}$$

Thus $P_q(t_k)$ can be computed integrating the following from t_{k-1} to t_k

$$\begin{aligned} \dot{P}_q(t) &= -A'_{e,q} P_q(t) - P_q(t) A_{e,q} - Q, \\ P_q(t_{k-1}) &= P_{q-1}(t_{k-1}), \end{aligned}$$

and the cost functional can be written again as in (1.3.4)*. Note that we need the last controller ensuring closed-loop asymptotic stability to keep the condition $V_{N+1}(\infty) = 0$. This enlarges the class of admissible controllers possibly leading to improved performances.*

In Figure 1.6 we show a simulation with the same plant (1.3.5) comparing the system output achieved performing on-line optimization of the switching time only, with the optimal controller reset and finally adding also the feedforward signal $w(t) = x_w(t^\star) e^{-100(t-t^\star)}$. The reference signal is $r(t) = 1 - e^{-20t}$.

1.5 Conclusions

Given two pre-defined controllers and a LTI plant, in this paper we provided some strategies aiming to compute the optimal switching times, controller resets and vanishing feedforward signal minimizing a quadratic cost functional of the closed-loop system state. Compared to [13], we consider multiple switching times, controller resets and the feedforward term that sensibly improve performances and allow to fulfill further requirements such for instance input/output signal smoothing properties. Simulation results highlight the effectiveness of the proposed solutions.

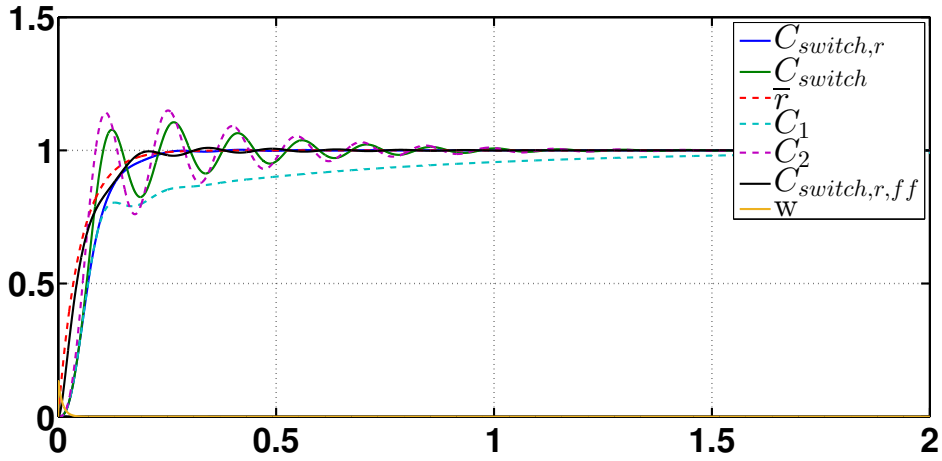

Figure 1.6: Comparison of the system outputs obtained in three different cases of on-line optimization. In red is depicted the reference signal to be tracked, the light blue and purple trajectories are the output achieved employing separately C_1 and C_2 controllers respectively. The green line is the output achieved with a single-switch controller optimizing only the switching time, the green one has been achieved optimizing also the controller reset state and the black one has been achieved optimizing also the initial condition of the feedforward signal $w(t) = x_w(t^\star)e^{-100(t-t^\star)}$. The time resolution is $10^{-4}s$

Chapter 2

FTU Control System

This chapter is dedicated to the description of the work carried out on the Frascati Tokamak Upgrade (FTU) control system

2.1 Introduction

The greater part of the universe is formed by plasma, a ionized gas composed by positive ions and electrons which forms the so-called fourth state of the matter.
In order to trigger the fusion process, it is necessary to heat the plasma to very high temperatures confining it in a limited space. A tokamak, is a machine that exploits the magnetic interaction properties of the plasma, to confine it in well-defined areas. Frascati Tokamak Upgrade (FTU) is a medium-size tokamak with a high toroidal magnetic field up to 8 Tesla. Figure 2.1 shows the position of the FTU coils. The plasma current inside the vacuum chamber is induced by the central solenoid T, which acts as the primary of a transformer. The coil V is used to provide the greater part of the vertical magnetic field needed to confine radially the plasma. The maximum current flowing in V is 25 kA, with a maximum variation of 54 kA/s. The feedback radial control actuator coil F, having a faster dynamics than V, can bring maximum 12.5 kA of current but with a maximum variation of 830 kA/s. The coil H generates a radial magnetic field and it is responsible of the vertical plasma displacement. The toroidal coil surrounding the vacuum vessel is called M and can generate up to 8 T magnetic field.

2.2 MARTe

The control system of FTU has been implemented within the MARTe framework. MARTe stands for Multi-Thread-Real-Time executor, it has been developed in Cul-

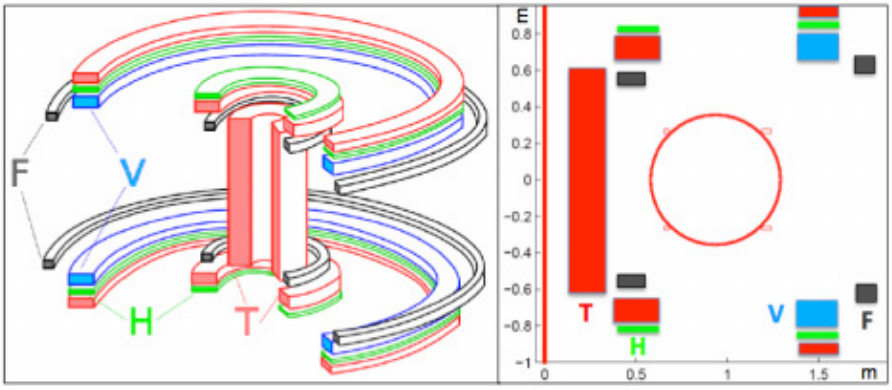

Figure 2.1: FTU coils layout. The central T coil is used for plasma current control, the F and V coils supply the plasma radial confinement, the coil H supplies the vertical plasma confinement.

ham since 1994 and today it is used on several international fusion devices. The core of MARTe is the BaseLib2 code written in C++, formed by several levels representing the abstraction layers from the operating system and architecture of the devices. Actually MARTe does not implements natively an embedded operating system, but is up to the Level 0 of BaseLib2 to wrap the APIs of the operating system of the device where the framework is intended to be used. Nowadays MARTe supports the most popular operating systems: Windows, Linux, RTAI, VxWorks.
The concept of MARTe meets the block schemes approach that should be familiar to control engineers. The final user of MARTe can connect together functional blocks called GAMs (Generic Application Modules) that share their signals through a common database collector called DDB (Dynamic Data Buffer). Special blocks called GACQMs (Generic Acquisition Modules), manage the interface with hardware devices like ADCs, DACs, etc. The MARTe user can employ the already developed available GAMs like the WebStatisticGAM that shows statistics for each defined signal (average, max and min value, etc.), the CollectionGAM that collects and stores a configurable amount of samples for each signal, the PlottingGam that shows in real-time plots of the signal values. MARTe implements internally a HTTP server that allows to the user to visualize graphical interfaces of the blocks that implement it, directly on a web browser.
The blocks configurations and interconnections can be built within a configuration file where the user can specify the configurable parameters exposed by the involved GAMs and the interconnections. Inside each thread block the user can define the set of GAMs to be executed by the scheduler together with other configurable options like the CPU mask where the thread should be executed or the thread priority. Errors in the configuration file are captured by the MARTe error manager and there exists

available tools like the MARTe JTLogger that shows all the captured diagnostic and error messages and streams them out to an UDP port allowing to the user the debugging operations.
The greater part of each configuration file is devoted to the configuration of the application state machine. Actually, for each different application state it is possible define a set of different threads executing a set of different GAMs.
The GACQMs in MARTe can be set as the synchronization point of a thread cycle. Among the GAMs there are two types which have a special purpose; the InputGAM generally acquires data from external devices and OutputGAM streams out data on external devices. They are usually linked to a GACQM for the synchronization and for the data acquisition (in case of the InputGAM) or storage/streaming-out (in case of the OutputGAM). Figure 2.2 shows on the left how a GAM interacts with others through the DDB, on the right an example of MARTe Real Time Thread (RTT) that schedules serially seven GAMs.
MARTe at FTU has been implemented on a RTAI distribution and the entire system has been deployed on a bootable USB key that must be re-generated after each software upgrade. The compiled code is packed in kernel modules that are inserted in the kernel and, after system boot, the kernel space implementation of MARTe realizes the real time threads using the RTAI kernel threads. All this system has been mainly implemented by the Ph.D Luca Boncagni in collaboration with other researchers.
Further explanations and descriptions on MARTe framework can be found at [17] [18].

2.3 Acquisition

The PPCDC (plasma position current density control) Driver Pool is composed by five GACQMs that drives an industrial controller based on VME bus consisting in a VME crate equipped with four 16-channels ADCs (analogic to digital converters with resolution of 12 bits), two 8-channels DACs (digital to analogic converters with resolution of 12 bits), a scaler for NIM (Nuclear Instrumentation Module) pulses counting and a CPU board. The latter consists in a Intel quad-core 32-bit processor, 4 GB RAM and a reflective memory installed on its PCM (Pulse-Code Modulation) interface.
Follows a list of the five GACQMs belonging to the DriverPool:

- VMEDrv, manages the VME devices (ADC, DAC and Scaler) and the real time thread synchronization with the experiment time.
- Rfm2gDrv manages the IO from and to the reflective memory,
- StreamingDrv, manages the UDP non real-time streaming for the Experiment monitor graphical user interface,

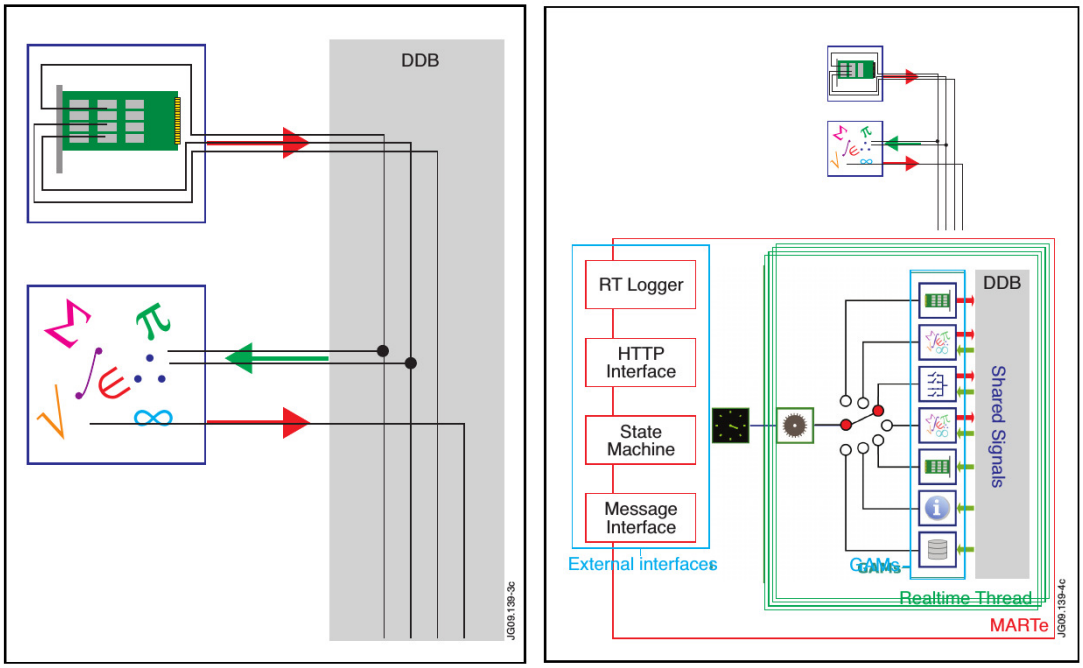

Figure 2.2: On the left is shown a GAM interacting with an hardware device across the shared DDB memory. The DDB is the only available way for GAMs to interchange signals. On the right is shown a group of seven GAMs scheduled within a RTT.

- Two SynchronizingDrv to manage a second slower real time thread synchronization and data exchange. The real time thread calculates in parallel the equilibrium reconstruction.

The VMEDrv GACQM synchronizes the main MARTe real time thread by polling on an external clock with 2KHz frequency. At each clock rising edge, each ADC channel stores the converted value to its corresponding memory register. Each memory address can be found on the VME data sheet and defined on the configuration file.
Concerning the DACs, the signals that control the coils and plasma presence for radio frequency injection acknowledgment are set up between -5V and 5V, while the gas valves signals between 0V and 10V. As for the ADCs, the ranges have to be selected by jumpers on the board.
The ADCs acquire the magnetic measurements from the 16 saddle loops and the 16 poloidal field pickup coils placed in each instrumented sector (sectors 4,7,10). From these measurements, it is possible to build the signals to be controlled; we mainly focus on the plasma current I_p, on the two radial plasma boundaries R_{in}, R_{ext} and on the two vertical plasma boundaries Z_{up}, Z_{down}. From the acquired magnetic fluxes, the MomentsGAM calculates the vacuum internal and externals moments (4 internal and 4 external moments) from those is possible to compute the plasma current and

to approximate the plasma shape. The moments go in input to the LcsmGAM (Last Closed Magnetic Surface GAM) that computes plasma current and shape. LcsmGAM contains Mesh configurable objects that must match the real positions of the poloidal and toroidal contact points in order to perform the plasma shape reconstruction. Moreover, LcsmGAM computes the DEP and DEZ errors, necessary to the plasma position control. The DEP error, used for radial plasma control, is:

$$DEP = (\psi_{ext,des} - \psi_{ext}) - (\psi_{int,des} - \psi_{int})$$

where $\psi_{ext,des}$, $\psi_{int,des}$ are the desired fluxes at (R_{ext}, 0) coordinates and (R_{int}, 0) coordinates, while ψ_{ext}, ψ_{int} are the fluxes computed from measurements at the same coordinates. Similarly, the DEZ error, used for vertical plasma control is:

$$DEZ = (\psi_{up,des} - \psi_{up}) - (\psi_{down,des} - \psi_{down})$$

where $\psi_{up,des}$, $\psi_{down,des}$ are the desired fluxes at (R_{tor}, Z_{up}) coordinates and (R_{tor}, Z_{down}) coordinates, while ψ_{up}, ψ_{down} are the fluxes computed from measurements at the same coordinates. R_{tor} is the radial center of a poloidal section of the vacuum vessel. Figure 2.3 shows the poloidal section of the chamber within Cartesian coordinates.

The reference signals are generated using a WaveformGenerator MARTe block that allows the user to define two arrays with the same length on the configuration file, containing a set of points and a set of related time instants respectively. These points will be first-order interpolated generating the desired waveform. Besides the reference signals, the WaveformGenerator has been used to generate the pre-programmed control signals (feedforward control) to be provided to the coils T, F, V in addiction to the feedback control.

The feedback control has been implemented providing the error signals to PID controllers. The MARTe block implementing the controller is FTUPID that takes in input the error $e(t)$ and calculated its derivative and integral providing in output $u(t) = K_p(t)\, e(t) + K_d(t)\, \dot{e}(t) + K_i(t) \int e(t)\, dt$. Obviously the derivative and the integral of the error are approximated over the sample time: the integral is trivially the sum of the error samples multiplied for the sample time, while the derivative is calculated by employing a MARTe block called PseudoDerivator. The FTUPID block contains three WaveformGenerators, one for each PID gain, allowing to change them during the experiments to adapt the controller action for the different experiment phases (ramp-up, flat-top, ramp-down). Again, the gains waveforms are fully configurable by configuration file. The PseudoDerivator block, largely used in the control system implementation, stores the last d samples of a signal x in input and computes the derivative approximation as follows:

$$\frac{\sum_{i=0}^{c} x(k-d+i+1) - x(k-i)}{(d-c)c\,T_s} \tag{2.3.1}$$

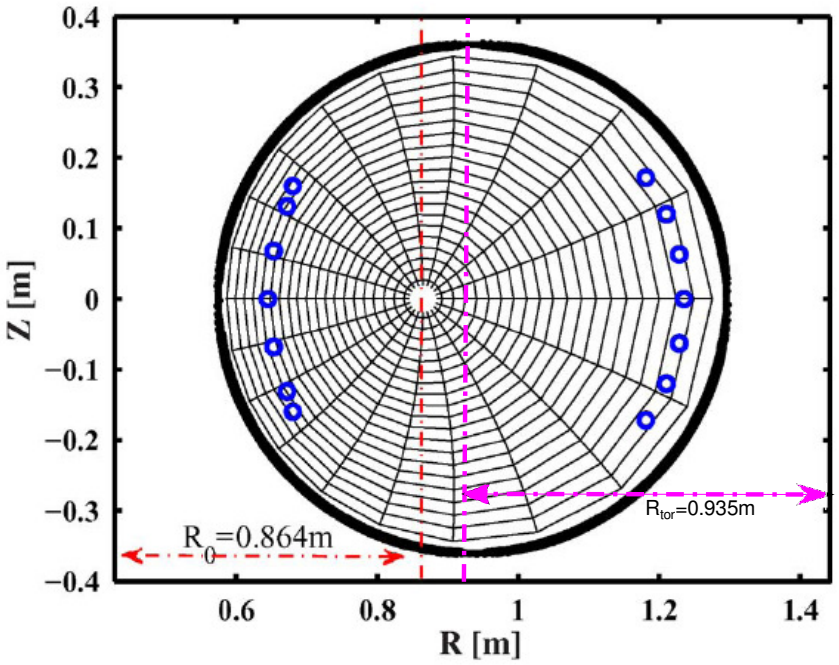

Figure 2.3: Poloidal section of FTU chamber in Cartesian coordinates. In blue the position of the meshes used to perform the plasma shape reconstruction.

where $c \leq d/2$ and T_s is the sample time. The constant values d and c can be configured as usual in the configuration file. By acting on the sliding window sizes d and c, this blocks have the effect of filtering the signal derivative (enlarge d have the effect of obtain a smoother derivative). Note that with $d = 2$ and $c = 1$ (2.3.1) is simply the incremental ratio, namely the simpler and basic derivative discrete approximation.
The milestones of the FTU MARTe control system are the SafetyGAM and the ControllerGAM. The SafetyGAM performs various checks and has the task to shutdown the discharge in case of situations that might cause damages to the machine. The ControllerGAM implements the control system.

2.4 Plasma Current Control

The central solenoid T acts as the primary of a transformer inducing the plasma current. A first approximation, considering the plasma as a simple solenoid, is the following:

$$R_p I_p + L_p \dot{I}_p + M_T \dot{I}_T + \ldots = 0 \tag{2.4.1}$$

and transforming it in Laplace domain we obtain:

$$I_p(s) = \frac{sM_T}{R_p + L_p s} I_T(s) + \dots \tag{2.4.2}$$

Note that the transfer function between the current flowing in T and plasma current have a zero in the origin. This means that, in order to achieve asymptotic tracking of constant references, we need to add two poles in zero to the controller. Towards this end, a switching double integrator block has been deployed to be added to the standard PID controller in order to eliminate the steady state error in flat-top phase. The double integrator acts only when the tracking error and its derivate evolve within predefined ranges; this switching policy tries to prevent large overshoots and to improve the convergence speed. Figure 2.4 shows the block scheme of the I_p control loop. The

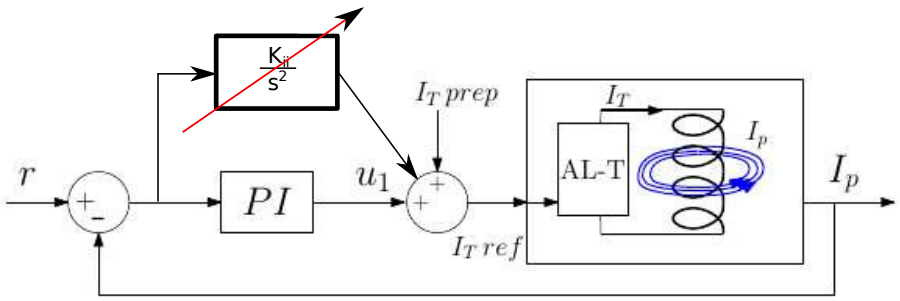

Figure 2.4: The FTU plasma current control loop.

used approach recalls the work done on switching controllers described in chapter 1 but, in this case, without the perfect knowledge of the plant, it has been decided to define a priori the switching state regions. The code has been implemented in the MARTe SwitchedDoubleIntegrator block and the user can define on configuration file the thresholds $\gamma_{e,1}$, $\gamma_{e,2}$, $\gamma_{\dot{e},1}$, $\gamma_{\dot{e},2}$ that bounds the tracking error $e_{I_p} \in [\gamma_{e,1}, \gamma_{e,2}]$ and its derivative $\dot{e_{I_p}} \in [\gamma_{\dot{e},1}, \gamma_{\dot{e},2}]$, the gains of the double integrator k_{12}, k_{2e}, and a discharge constant ϵ. Denoting the tracking error with e, the PID action with u_{PID} and the double integrator action with u_{DI}, follows the description of the procedure implemented in the SwitchedDoubleIntegrator block.

- If $e \notin [\gamma_{e,1}, \gamma_{e,2}]$ or $\dot{e} \notin [\gamma_{\dot{e},1}, \gamma_{\dot{e},2}]$ and $\dot{u}_{PID} u_{DI} < 0$ (this condition checks if the double integrator action contrasts the PID action), discharge exponentially the

double integrator by doing:

$$\begin{aligned} \dot{x}_1 &= -\frac{1}{\epsilon}x_1 \\ \dot{x}_2 &= -\frac{1}{\epsilon}x_2 \\ u_{DI} &= x_1 \end{aligned}$$

- If $e \in [\gamma_{e,1}, \gamma_{e,2}]$ and $\dot{e} \in [\gamma_{\dot{e},1}, \gamma_{\dot{e},2}]$, keep double integrating by doing:

$$\begin{aligned} \dot{x}_1 &= k_{12}x_2 \\ \dot{x}_2 &= k_{2e}e \\ u_{DI} &= x_1 \end{aligned}$$

- If $e \notin [\gamma_{e,1}, \gamma_{e,2}]$ or $\dot{e} \notin [\gamma_{\dot{e},1}, \gamma_{\dot{e},2}]$ and $\dot{u}_{PID}u_{DI} > 0$ (this condition checks if the double integrator action helps the PID action), freeze the action of the double integrator to the current value. The double integrator action will be freezed also when saturations occur for x_1 and x_2 states (saturation values can be defined in configuration file too).

The double integrator action (when active) together with the basic PID action implement the following controller's transfer function:

$$C(s) = \Big[k_p + \frac{k_i}{s} + \frac{k_{ii}}{s^2}\Big]e(s)$$

since the derivative part of the PID is generally never used imposing $k_d = 0$. The gain $k_ii = k_{12}k_{2e}$ is the gain of the double integrator. The block SwitchedDoubleIntegrator block execution code is called inside the ControllerGAM during the execution cycle adding its action to the FTUPID one. It has been largely tested during the last two years of experimental campaigns with optimal results. In the figure (2.5) we can see a remarkable reduction of the tracking error after the activation of the double integrator for the shot 38786. During the 35975 and the 35976 shots, when the double integrator has been disabled note that the tracking error is constant in flat-top phase.

2.5 Plasma Position Control

In this section a description on the control system implemented for plasma displacement is provided. As mentioned in the section 2.3, the *DEP* denotes the plasma radial position tracking error and the *DEZ* the vertical one. The radial and vertical controllers have to be designed to bring and keep both to zero as best as possible during the shot time.
The radial control actuators are the coils V and F (see figure 2.1). The feedback

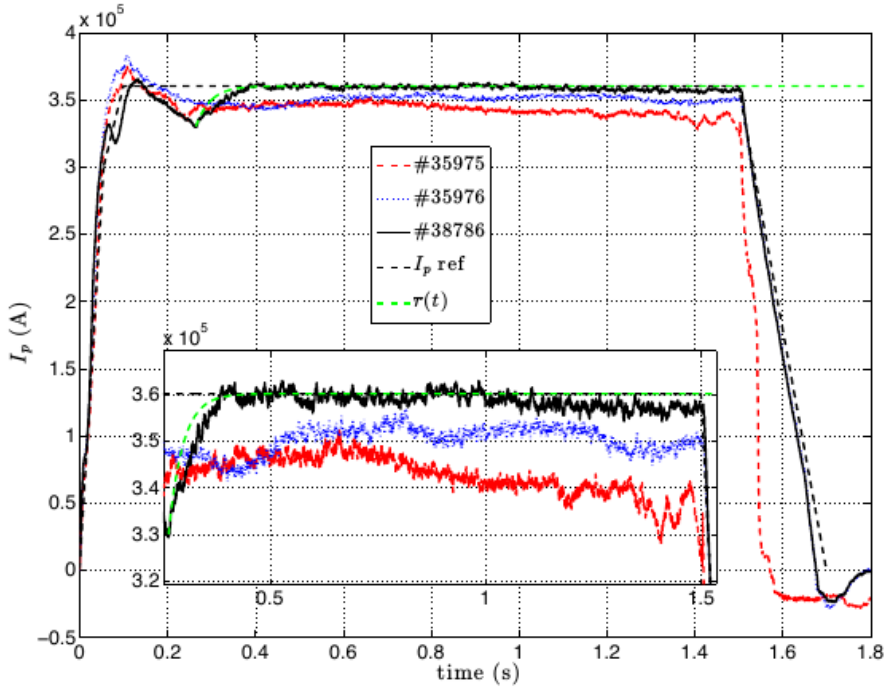

Figure 2.5: Comparison between the shots 35975 and 35976 without the double integrator and the shot 38786 when the double integrator has been activated from 0.25s

control has been implemented on the coil F, while the great part of the feed-forward action is demanded to the coil V. The coil F is much faster with respect the other coils (max 830 kA/s vs the 54 kA/s of V), but its maximum current range is ±12 kA versus ±25 kA on V, so it can happen that the F current saturates impacting on the control performance (radial shift of the plasma position). For this reason, a current allocator has been implemented, that transfers the control action from F to V when the F current is close to its limit bound. The basic controller implemented for radial plasma position control is also in this case a PID. Besides the PID gains have been well tuned, the DEP is usually affected by oscillations and instabilities, mostly in case of runaway activity. Moreover, during the elongated plasmas experiments, the PID action is not fast enough to suppress the well known vertical instabilities [19] thus, to improve the controller performance, a hybrid fast controller has been implemented for radial and vertical position, aiming to bring back the DEP and DEZ to zero in case of instabilities. In particular, it tries to recover the plasma position when tracking errors $e_x = DEP$ and $e_y = DEZ$ and their derivatives $\dot{e_x}$ and $\dot{e_y}$ evolve outside pre-defined ranges by adding a ramp signal with adaptive slope to the basic control. Actually, the instability is detected out when the norm of the tracking error and its derivative are large enough and have the same sign. In this case the

fast ramp controller adds its aggressive action to the standard PID control: a ramp with a slope that depends on the *speed* of the instability. The ramp action has to be discharged when the plasma position is going to return in the safe region. The discharge policy is crucial because if the fast ramp controller acts until the tracking error and its derivative have returned into their ranges, it can cause an oscillation in the other direction, inducing instabilities instead of suppressing them. This is also caused by the slow dynamics of the actuators that contributes to delay the control action. To anticipate the discharge time, it has been decided to start decreasing the ramp action when the norm of the second derivative of the error becomes minor than zero (or than a configurable threshold) because it means that the norm of the tracking error is starting to decrease. Using the double derivative to anticipate the discharge prevents that the ramp action itself contributes to more oscillations and instabilities. In case of consecutive activations of the fast ramp controller if the plasma is escaping in the same direction, the ramp inclination is increased. Conversely, if the controller is activated consecutively because the plasma is escaping alternatively in opposite directions, the ramp inclination is decreased assuming that the action is so aggressive that contributes to generate oscillations. In case of no activations during a configurable time period, the ramp inclination is reset to the initial one.
The ramp controller is a time-varying control that can be expressed with:

$$u_F(t) = \begin{cases} -sign(e)k(t-t_s), & \text{if } |e| \geq \sigma_1,\ |\dot{e}| \geq \sigma_2,\ e\dot{e} \geq 0 \text{ and } sign(e)\ddot{e} > \sigma_3 \\ 0, & \text{elsewhere} \end{cases} \tag{2.5.1}$$

where $\sigma_1 > 0$, $\sigma_2 > 0$ and σ_3 are respectively the thresholds on e, $\dot{e}$ and $\ddot{e}$, t_s is the time instant when the error goes beyond the threshold and k is the ramp slope. The second derivative is used to discharge the ramp in time to prevent oscillations. The transition between the ramp action (when the controller is active) to zero (when the control is inactive) is not immediate but first-order filtered to obtain an exponential discharge. To avoid the chattering phenomena it has been implemented an anti-chattering policy that prevents that the controller returns to the previous state consecutively. To prove the proprieties of this type of controllers, a simple case has been analyzed.
Consider a linear first ordered system with the following state-space model:

$$\dot{x}(t) = ax(t) + bu(t)$$

and suppose that it is not asymptotically stable ($a \geq 0$). Assume that $b = 1$ and the slope of the ramp controller k constant. Denote the constant $\gamma \geq 0$ such that the fast controller is not active if the state $x(t) \in [-\gamma, \gamma]$ and assume the initial state $x_0 \notin [-\gamma, \gamma]$ such that the controller is active from the beginning. Under these extremely simplified hypothesis, there exists a set

$$\Phi = \{x_0 \in \mathbb{R} : \gamma < |x_0| < \frac{k}{a^2})\} \tag{2.5.2}$$

such that if $x_0 \in \Phi$ then the control will bring in finite time the state in the set $x \in \epsilon$ with $\epsilon < \gamma < |x_0|$. Consider also ϵ as the switching threshold hysteresis when the ramp control is turned off, so consider the case in witch the ramp control is still active ($|x| > \epsilon$). Under these hypothesis, the control can be simplified as follows:

$$u_F(t) = -sign(x)kt$$

Let choose the Lyapunov function $V(x) = \frac{x^2}{2} \succ 0$, whose derivate is $\dot{V}(x) = ax^2 - |x|kt$ because $xsign(x) = |x|$. Let introduce the set Γ such that if $x(t) \in \Gamma$ then $\dot{V}(x,t) \prec 0$:

$$\Gamma = \{x \in \mathbb{R} : \epsilon < |x| \leq \frac{kt}{a}\}$$

If during its evolution $x(t)$ enters in the Γ set (which is strictly growing), it will be attracted to zero or, in this case, to the *safe region* $|x| \leq \epsilon$, where the ramp control will be disabled.
Consider the evolution of x if $x > \epsilon$ when the control is $-kt$:

$$x(t) = x_0 e^{at} - \int_0^t e^{a(t-\tau)} k\tau d\tau$$
$$x(t) = (x_0 - \frac{k}{a^2})e^{at} + \frac{kt}{a} + \frac{k}{a^2}$$
$$\dot{x}(t) = a(x_0 - \frac{k}{a^2})e^{at} + \frac{k}{a}$$

Following the same path if $|x| < -\epsilon$ we obtain:

$$\dot{x}(t) = a(x_0 - \frac{k}{a^2})e^{at} - \frac{k}{a}$$

The growing speed of the set Γ is $\frac{k}{a}$, so it is sufficient that the derivative of the state $x(t)$ is minor to ensure that there exists a T such that for $t > T$ the state $x(t) \in \Gamma$.

$$a(x_0 - \frac{k}{a^2})e^{at} + \frac{k}{a} < \frac{k}{a}$$
$$a(x_0 - \frac{k}{a^2})e^{at} - \frac{k}{a} > -\frac{k}{a}$$

obtaining the condition $|x_0| < \frac{k}{a^2}$ that defines the set 2.5.2.
The integration of the ramp controller on the vertical position controller, has improved very much the control performance in case of vertical instabilities for elongated plasmas and the achieved good results have encouraged the integration also to the radial position control. Figure 2.6 shows the ramp controller action for plasma radial confinement. In the left column the magenta rectangle highlights when the slope of the ramp is adaptively decreased because of the oscillations in the plasma position error while the green rectangle puts in evidence when the slope is increased because the plasma position error diverges to the same direction.

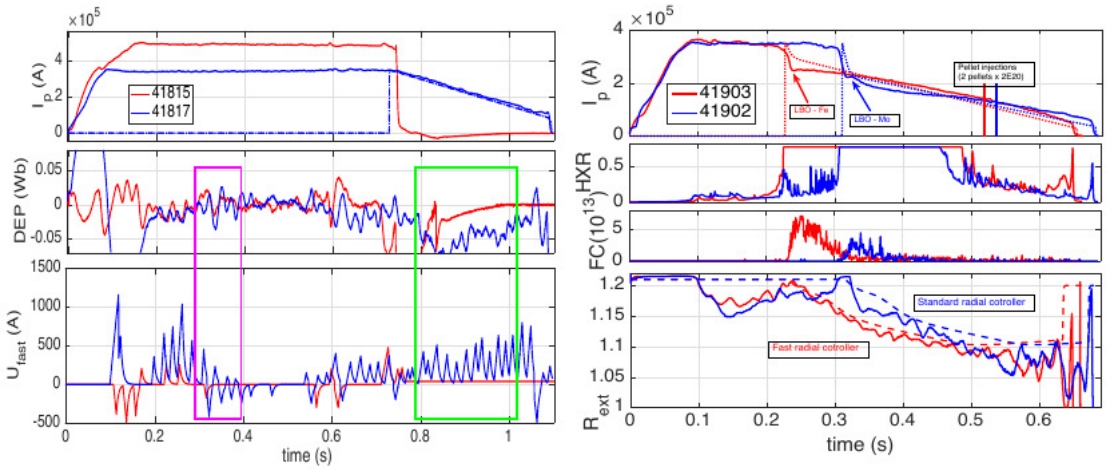

Figure 2.6: On the left column two shots: 41815 and 41817 are shown. On top the plasma current, in the middle the DEP radial displacement, on bottom the ramp controller action. On the right column a comparison between a shot without the ramp controller (41902) and one with the ramp controller (41903).

2.5.1 Future implementations: adaptive strategies

One of the problems of developing controllers more sophisticated than PIDs is that it is not easy to extract a suitable tokamak model with identification techniques. A good alternative to robust controllers like PIDs could be to approach to adaptive techniques, estimating the plant parameters during the experiments and changing accordingly the controller gains. A study on adaptive techniques for output regulation carried forward mainly by the professor of the Tor Vergata university Sergio Galeani, provided the occasion to consider this approach to plasma radial control.
The implemented *slow* controller has the task to provide the greater part of the control action computed thanks on a on-line estimation of the input-output gains of the plant. Its action will be further added to the PID and to the "fast" ramp controller that provide the smaller corrections to the control in order to achieve null error. A MARTe block that implements the slow adaptive radial controller has been coded and simulated in open loop but it has never been tested in feedback during the experimental campaign. Even if its correctness cannot be actually experimentally proved, a description to its behavior has been provided in this thesis.
First of all it is assumed that the DEP error depends on the plasma current I_p and on a combination of the currents flowing in the V and F coils. It is known that the current on F has an effect on the radial error about 4.5 times greater than the current on V (in plasma equilibrium conditions), thus these two inputs are assumed to be $u_r = I_F + 4.5I_V$ and I_p. Moreover the plant is supposed to be linear and asymptotically stable, assumption that reflects the experimental input-output behavior.
Exciting an asymptotically stable plant with a sinusoidal wave, the steady-state out-

put has the same frequency of the input but (in general) different amplitude and phase. For constant inputs (frequency equal to zero) it results in a change of the offset and for ramp inputs also in a change of the slope. In this case it has been decided to approximate the plant inputs I_p and u_r and output DEP to ramps during a configurable time interval:

$$\begin{aligned}\overline{I}_p(t) &= m_{ip}t + q_{ip}\\ \overline{u}_r(t) &= m_u t + q_u\\ \overline{e}(t) &= m_e t + q_e\end{aligned}$$

Providing a ramp in input to a linear asymptotically stable plant, the output after the transient time is a ramp with different slope and offset with respect the input. In particular, considering a generic input ramp in the Laplace domain $u_r(s) = \frac{m_u}{s^2} + \frac{q_u}{s}$, the output $y(s)$ of the plant $P(s)$ is:

$$y(s) = P(s)\Big[\frac{m_u}{s^2} + \frac{q_u}{s}\Big] = \frac{\beta_1 m_u}{s^2} + \frac{\beta_2 m_u}{s} + \frac{\beta_1 q_u}{s}$$

where $\beta_1 = P(s)|_{s=0}$ and $\beta_2 = \frac{\partial P(s)}{\partial s}|_{s=0}$. In the the time domain:

$$\begin{aligned}\overline{e}_{ip} &= \beta_1(m_{ip}t + q_{ip}) + \beta_3 m_{ip}\\ \overline{e}_u &= \beta_2(m_u t + q_u) + \beta_4 m_u\\ \overline{e} &= \overline{e}_{ip} + \overline{e}_u = m_e t + q_e\end{aligned}$$

The adaptive controller aims to estimate the gains β_1, β_2, β_3, β_4 once a time interval with configurable length T_s is elapsed. The algorithm stores the inputs u_r, I_p and the output $e = DEP$ samples belonging to the last sliding window computing step by step the offsets q_u, q_{i_p}, q_e and the slopes m_u, m_{i_p}, m_e of the ramps that best fits the signals. Lets denote with $\delta(x)$ the mean of the samples of the signal x stored during the elapsed time interval, the angular coefficients $m_x = \delta(\dot{x})$, the offsets $q_x = \delta(x) - m_x T_s$. The current approximation of each (input or output) signal is $\overline{x}(t) = m_x T_s + q_x$. Each time interval with duration $T_e \geq T_s$, the algorithm refreshes the estimation of the β gains using the slopes and offsets of the last time interval $[t - T_s, t]$ $(m_u, q_u, m_{i_p}, q_{i_p}, m_e, q_e)$ and the previous one $[t - T_e - T_s, t - T_e]$ $(m_{u,-1}, q_{u,-1}, m_{i_p,-1}, q_{i_p,-1}, m_{e,-1}, q_{e,-1})$. Once the new β gains have been estimated, they can be used for the next time interval T_e to generate a control ramp $u_{ad} = m_{ad}t + q_{ad}$ that reset to zero the DEP error ramp approximation $\overline{e}$ (in the last time interval T_s). The β gains are estimated as follows:

$$\begin{bmatrix} m_{e,-1} \\ q_{e,-1} \\ m_e \\ q_e \end{bmatrix} = \begin{bmatrix} m_{ip,-1} & m_{u,-1} & 0 & 0 \\ q_{ip,-1} & q_{u,-1} & m_{ip,-1} & m_{u,-1} \\ m_{ip} & m_u 0 & 0 & \\ q_{ip} & q_u & m_{ip} & m_u \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}$$

Note that, since our plant takes two input, at least the last two time intervals are necessary to compute the β gains. Using the informations of three or more intervals could improve the estimation of β gains pseudo-inverting the matrix.
Once the gains have been computed, for the next T_e time period, the u_{ad} control is added to u due to reset to zero $\overline{e}$:

$$\begin{aligned} m_e T_s + q_e =& \beta_1(m_{ip}T_s + q_{ip}) + \beta_2((m_u + m_{ad})T_s + q_u + q_{ad}) + \beta_3 m_{ip} + \beta_4(m_u + m_{ad}) \\ m_e =& 0 \implies m_{ad} = -\frac{1}{\beta_2}(\beta_1 m_{ip} + m_u) \\ q_e =& 0 \implies q_{ad} = -\frac{1}{\beta_2}(\beta_1 q_{ip} + q_u + \beta_3 m_{ip} + \beta_4(m_u + m_{ad})) \\ u_{ad} =& m_{ad}T_s + q_{ad} \end{aligned}$$

The latter computed u_{ad} is provided to the SlowFFControllerF MARTe block to be added to the F coil current.

2.6 Runaway Control

In this section I present some experimental results regarding the position and current control of disruption generated runaway electrons (RE) in FTU. A scanning interferometer diagnostic has been used to analyze the time evolution of the RE beam radial position and its instabilities. New RE control algorithms, which define in real-time updated plasma current and position references, have been tested in experimental scenarios featuring disruption generated RE plateaus: the experimental data confirm the effectiveness of the control strategies as the RE beam interaction with the plasma facing components is reduced while the current is ramped-down.

2.6.1 Introduction

A crucial challenge towards a safe and efficient operation of ITER consists in the need of reducing the dangerous effects of runaway electrons (RE) during disruptions [20]. RE are considered to be potentially intolerable for ITER when exhibiting currents larger than 2 MA. One of the most popular strategies to address this task is based mainly on RE suppression by means of Massive Gas Injection (MGI) of High-Z noble gas before the thermal quench (TQ), which possesses the additional advantage of reducing the localized heat load. However, MGI leads to long recovery time, requires effective disruption predictors, and may lead to hot tail RE generation [21] or high mechanical loads if the Current Quench (CQ) does not occur in a suitable time interval [20], [22], [23]. Nevertheless, in the circumstances in which such suppression strategy is not effective, for instance due to a delayed detection of the disruption and/or to a failure of the gas valves or of disruption avoidance techniques

(e.g. ECRH) [24], an alternative strategy consisting of the RE beam energy and population dissipation by means of a RE active beam control may be pursued, as noted in [25], [26], [27]. Alternative mitigation techniques exploit magnetic (resonant) fluctuations/perturbations to displace RE; their effect on RE beam dissipation have been studied in [28], [29], [30]. Resonant magnetic perturbation techniques can be also used at the CQ to prevent large avalanche effects. However they require specific active coils that are not available at FTU. The method proposed here achieves stabilization of a disruption generated RE beam by minimizing its interaction with the plasma facing components (PFC). The RE energy dissipation is obtained by reducing the RE beam current via the central solenoid (inductive effects). Similar techniques have been investigated in DIII-D [25]. In particular, the focus here is on those RE that survive the CQ. When the RE beam position is stabilized, further techniques, not studied in the present work, such as high-Z gas injection to increase RE beam radiative losses, could be exploited. In the last years, experiments on RE active control have been carried out in DIII-D, Tore Supra, FTU, JET, and initial studies have been carried out also at COMPASS [31]. In Tore Supra attempts of RE termalization via MGI (He) have been investigated [32]. In DIII-D disruptions have been induced by injecting either Argon pellets or MGI while the ohmic coil current feedback has been left active to maintain constant current levels or to follow the desired current ramp-down [25]. DIII-D also studied the current beam dissipation rate by means of MGI with a final termination at approximatively 100kA [33]. Similar results on MGI mitigation of RE have been obtained at JET [34]. The present work goes along similar lines but RE beam dissipation is obtained only by inductive effects, i.e. via central solenoid as in [25], combined with a new dedicated tool of the Plasma Control System (PCS). This scheme yields a RE beam current ramp-down and position control. Effectiveness of the novel approach is measured in term of reduced interaction of highly energetic runaways with the PFC. Furthermore, as in [33], we consider the RE beam radial position obtained by the CO_2/CO scanning interferometer, showing that is also in agreement with neutron diagnostics and the standard real-time algorithm based on magnetic measurements that estimate the plasma boundary. A brief list of FTU diagnostics correlated with this work is given below. Further details are given in [35].

- **Fission Chamber (FC)**: a low sensitivity ^{235}U fission chamber manufactured by Centronic, with a coating of $30\mu g/cm^2$ of ^{235}U operated in pulse mode at $1ms$ time resolution and calibrated with a 252Cf source. This diagnostic is essential in the analysis of the sequence of events occurring during the RE current plateau phase since the standard Hard X-ray (HXR) and neutron monitors are typically constantly saturated after the CQ. During the RE plateau phase this detector measures photoneutrons and photofissions induced by gamma rays with energy higher than $6\,MeV$ (produced by bremsstrahlung of the RE interacting with the metallic plasma facing components).

- **Soft-x (SXR)**: the multichannel bolometer detects X rays emitted at the magnetic center of the toroidal camera (major radius equal to $0.96\,m$) in the range of 5 eV to 10 keV. Within this range also RE collisions with plasma impurities can be detected.

- **Hard-x (HXR)**: the X-rays are monitored by two systems:

  1. NaI scintillator detector sensitive to hard-x rays with energy higher than 200 keV mainly emitted by RE hitting the vessel (labeled as HXR in the figures).
  2. The NE213 detector sensitive both to neutron and to gamma rays and cross calibrated with a BF3 neutron detector in discharges with no RE [28]. This detector (labeled as NEU213 in the figures) is used to monitor the formation of RE during the discharge, however at the disruption and during the RE plateau its signal is usually saturated and therefore the gamma monitoring is replaced by the fission chamber.

- **Interferometer**: the CO_2/CO scanning interferometer can provide the number of electrons measured on several plasma vertical chords (lines of sight, LOS) intercepting the equatorial plane at different radii ranging from 0.8965 m to 1.2297 m with a sampling time of 62.5 μs. Detailed information related to mounting position and specific features are given in [30].

- **MHD sensors**: the amplitude of the Mirnov coil signal [17] considered is directly related to helical deformations of the plasma resulting from MHD instabilities, having in most cases n=1 (m=2) toroidal (poloidal) periodicity.

- **Cherenkov probes**: three probes placed at the equatorial plane in the low-field side of the chamber and within 2.5 cm from the poloidal limiter. The three probes detect REs leaving the plasma with energy higher than 58, 189 and 359 KeV, respectively.

2.6.2 Control strategies

Dedicated FTU plasma discharges have been performed to test a novel real-time (RT) RE control algorithm. Such algorithm has been implemented within the framework of the FTU plasma control system (PCS) for position and I_p ramp-down control of disruption-generated RE. The PCS safety rules impose that whenever the HXR signal takes value above a given safety threshold (0.2) for more than 10 ms, indication that

harmful RE are present, the discharge has to be shut-down. In the previous shut-down policy the I_p reference was exponentially decreased down to zero. The new controller has been specifically designed for RE beam dissipation and comprises two different phases. In the first phase, specific algorithms are employed to detect the CQ and the RE beam plateau by processing the I_p and the HXR signal. At the same time, the Current Allocator steers the values of I_F away from saturation limits, to ensure that a larger excursion is available for the control of the RE beam position. In the second phase, once the RE beam event has been detected (CQ or HXR level), the I_p reference is ramped-down in order to dissipate the RE beam energy by means of the central solenoid. In particular, a scan of the initial values and slope of the updated I_p reference for RE suppression (current ramp-down), that substitutes the original I_p reference when the RE beam is detected, have been performed. At the same time the desired (reference) external radius R_{ext} is reduced linearly with different slopes down to predefined constant values. However, the updated R_{ext} reference is such that, below 1.1m, it is constrained to be not smaller than $R_{ext} - 0.03m$ to avoid large position errors that might induce harmful oscillations of the RE beam due to the action of the PID-F position controller. The R_{ext} has been reduced in order to compensate for a large outward shift of the RE beam, hence to preserve the low field side vessel from RE beam impacts, similarly to what has been proposed in [25]. The reduction of the external plasma radius reference can be considered the way of finding the RE beam radial position that provides minimal RE beam interaction with PFC, similar findings have been discussed in [25] and the RE beam position with minimal PFC interaction is called *safe zone*. In all the experiments, the internal radius R_{int} is not changed since we are operating in (internal) limiter configuration. Nevertheless, the control system has the objective to maintain the plasma within the desired horizontal and vertical radii, avoiding the plasma impact with the vessel.
The runaway control strategy has been managed inside the SafetyGAM by the interaction of different MARTe sub-blocks. The plasma current I_p is ramped-down if one or both these two conditions hold:

- The HRX signal is greater than a configurable threshold (usually set to 0.2) for more than a configurable time interval (usually 10 ms). This check is directly implemented in the SafetyGAM code.

- A CQ followed by a runaway plateau has been detected. The CQDetector MARTe block has been implemented for this check: if the plasma current derivative goes beyond a configurable negative threshold (current drop), this is recognized as a current quench; if successively the current returns above the threshold this is recognized as a runaway plateau. During experiment the threshold value for $\dot{I}_p$ is about $-6 \cdot 10^6$ MA/s.

The MARTe IpReferenceGenerator block receives in input the two assertions and if at least one is true, changes the plasma current reference to a descending ramp beginning

from the current value of I_p with a configurable slope. The CQDetector block can be configured to detect multiple runaway plateaus during the ramp discharge, thus the I_p reference can be adjusted multiple times.
During the ramp discharge, the descending ramp can be adjusted adding a offset if:

- The loop voltage (Vloop) is greater than a configurable threshold. This can be due to MHD instabilities, thus a negative offset is added to the descending ramp accelerating the shut down.
- The norm of the I_p tracking error is greater than a configurable threshold. To avoid to stress too much the central solenoid T during the shut down, a offset is added to the reference to approach the plasma current.

Similarly, the MARTe ReRefGenerator block has been implemented to change the radial external radius R_{ext} if runaway activity has been detected. The last implementation allows to define a waveform consisting in two vectors with the same length that define a set of points related to their time instants. These time instants are not absolute but relative to the plasma current ramp down time between 0 (the runaway plateau detection instant, namely the begin of the descending ramp) to 1 (when the ramp reference has reached zero). The most used configuration is to set a ramp-down reference also for R_{ext} to compensate the radial shift observed during the runaway plateau due to the high energy of the runaway electrons. Figure 2.7 shows the shots *38514, 38519, 39903, 40714* where the current reference hase been ramped down after a CQ detection. Note from the HXR and FC diagnostics that the runaway activity has been reduced significantly during the ramp down.

2.6.3 Shattered Pellet Injection

An alternative strategy to reduce the dangerous effects of the runaway beam that in recent times is under study is the *Shattered Pellet Injection* (SPI). This approach consists in firing a cryogenically freezing pellets of the desired species (deuterium or neon) from a specially designed *pipe gun*. The ionization of the pellet materials within the plasma can induce to a controlled termination of the plasma discharge. SPI showed significant improvements on DIII-D when compared to equivalent MGI in identical plasma targets [37].
On FTU, the pellet injector consists in four parallel guns that can shoot deuterium pellets at a speed between 1 and 2.5 km/s. The guns 3 and 4 shots the larger pellets with an injection quantity of $2 \cdot 10^{20}$, the guns 1 and 2 shots pellets with injection quantity equal to $1 \cdot 10^{20}$.
During the RE beam ramp-down we performed experiments in which these deuterium pellets have been injected in order to investigate their interaction with the RE beam. A MARTe block (PelletInjector) that manages the pellet shot system has been integrated within the SafetyGAM. The signals in output from this block go in input to

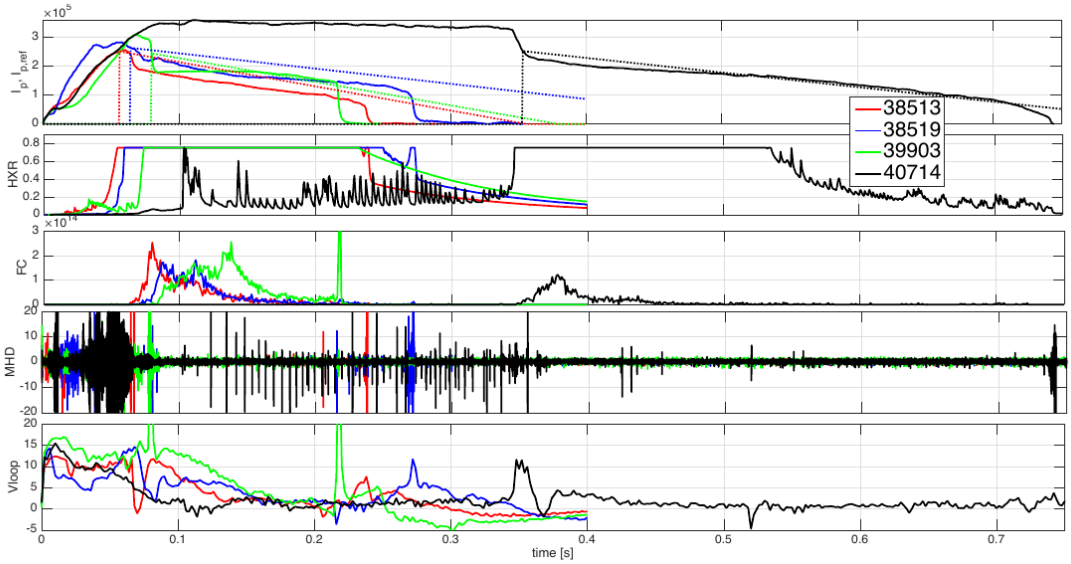

Figure 2.7: On top the I_p current for the shots: 38513, 18519, 39903 and 40714. Follow, from top to bottom, the HXR, FC, and Vloop diagnostics.

DACs to generate TTL signals to trigger the pellet shots. One of the requirements was to shot in feedback from the value of a chosen signal, so the block takes in input a vector containing a set of possible signals to be evaluated $Signal[]$. During experiments we shoot in feedback from plasma current, loop voltage, density and external radius. The configuration requests an array of values $Val[]$, an array of time instants $Time[]$ and two threshold delta values δ_{max} and δ_{min}. The system provides only four pellet guns, so both $Val[]$ and $Time[]$ arrays have four elements and the index selects the gun where the pellet will be shot. Depending on what specified by the user in the configuration file, it is possible select different modes:

- mode = 0: test mode, used to test the pellet shots in the absolute time instants specified in $Time[]$.
- mode = 1: pre-programmed mode, the pellets are shoot in the absolute time instants specified in $Time[]$.
- mode = i (≥ 2): feedback mode, the gun j shoots if:

$$(Val[j] - \delta_{min}) < Signal[i-2] < (Val[j] + \delta_{max}) \; and \; (t \geq Time[j])$$

  It is possible to shoot more than one pellet bullet at the same time.

In order to see the effects of pellets on the runaway beam, a flag in the configuration file specifies to shoot only after runaway activities detection, namely when the current will be ramped down to shut-down safely the discharge. If this flag is true in the

configuration file and we are in feedback mode (mode ≥ 2), the values in $Time[]$ have not absolute meaning, but they are relative to the runaway activity detection time instant. Follows an example of the *PelletInjector* MARTe block configuration:

```
+PelletShooter = {
    Class = PelletInjector
    Mode = 1  //pre-programmed mode
    Amplitude = 5.0  //5V trigger
    PreDisruptionIndex = -1  //don't shoot before disruption
    ShotTimes={0.15 0.25 0.27 0.28}
    ShotValues={0 1.06 1.18 1.8}
    TriggerDuration = 5000 //us
    Pellet_minIPL = 40E3 //don't shot if Ip is minor
    ValueRangeMin = 0.0
    ValueRangeMax = 10.0E10
    BypassCondition = 0
}
```

The signals of the CO_2 scanning interferometer, H-alpha, Soft/Hard-X and Mirnov coil reveal that the pellet is completely ablated by the RE beam but the deuterium ionization is small fraction with respect to the case of a hot plasma (or hot plasma with REs). This could be explained by the very low temperature of the beam and of the collisional rate of highly energetic REs even on a solid D pellet. Figure 2.8 shows some diagnostics of the shot 41899 where deuterium pellets have been injected. Note from the central cord density, that the pellets seems to be (at least partially) ablated but no remarkable effects have been observed regarding the mitigation of the runaway beam amd further studies are currently carried on.

2.7 Simulation on FTU cluster

Together with the researchers Luca Boncagni, Mateusz Gospodarczyk and Daniele Carnevale, a tool for testing and simulations that integrates MARTe and Matlab has been set up. A copy of the code that, after being compiled for RTAI, is executed during the shots, has been brought on a dedicated server to simulate in open loop, the developed MARTe environment for plasma control and diagnostics. In order to simulate a shot, it is necessary to select the shot number and to provide the correct configuration file. To perform a reasonable simulation, it is advisable to start from the configuration file used during the experiment, make the necessary changes (the configuration file for simulation is a little bit different from the real one because the shot signal inputs come from a data file built from the database and not from the hardware diagnostic boards) and the changes that has to be tested. The simulation is not in closed loop because of the absence of a plant and of a realistic model, but also

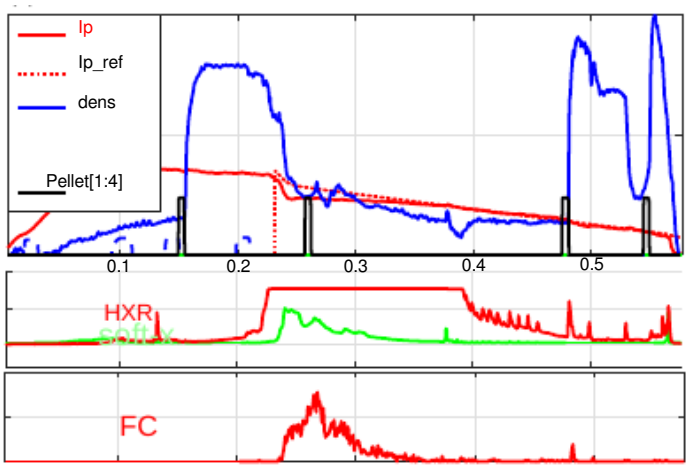

Figure 2.8: shot 41899: after the pellet injection the density value raises but no remarkable effects on FC and HXR diagnostics have been observed.

the open-loop simulation can be useful to test and debug the new developed code. It is possible to perform simulations remotely on the FTU servers. The data of the selected shot will be downloaded from the database and used by the *DataGAM*, which substitute the drivers within the simulations. Providing the custom configuration file, we can use the real shot inputs in order to simulate in open loop our improvements. Follows a short description of the behavior:

- the function *getftudata(shot, ...)* in matlab generates the file *data_actual.dat* with the shot data from database.
- *data_actual.dat* is copied from the FTU clusters to a dedicated server where the MARTe application will be launched taking in input the configuration file and the shot data file.
- at the end of the MARTe task, a matlab file containing the output signals that we have selected by the configuration file is generated and copied back to the FTU cluster, where it can be observed directly in matlab.
- the code must be changed on the dedicated server and compiled before launching a new simulation.

Follow the instruction on how to run a simulation on cluster ($-linux console, %-matlab console):

```
- $ssh -Y [user name]@[FTU cluster IP address]
- $matlab
- %cd [your project folder]
- %runSims_on_LucaAtBoncagni.m
```

2.7.1 Compile new MARTe code

How to change the MARTe code and run simulations manually (assuming you have the shot data file):

```
- $ssh -Y [user name]@[FTU cluster IP address]
- $ssh [dedicated server IP]
- $cd code/MARTe/GAMs/[GAM folder]
- edit and change the source code.
- $make -f Makefile.linux
- $cd ~/code/MARTe/TestArea
- $./MARTe-frascati.sh [cfg name]
- open a firefox on cluster and type [dedicated server IP]:[port specified
on cfg in Port param in HttpService class] as url
- click on BROWSE and roll out the StateMachine and run the simulation
step by step
- Press PULSE_SETUP_COMPLETED
- Press PRE
- Wait in PULSING state until the simulation is done (when
refreshing the http://[dedicated server IP]/BROWSE/MARTe/Thread_1/Collection/
the number of samples stored does not change)
- Press EJP and then COLLECTION_COMPLETED
- See output data in http://[dedicated server IP]/BROWSE/FlotPlot/
- Select the output data to be downloaded in mat format in
http://[dedicated server IP]/BROWSE/MatlabSignalServer/
```

2.7.2 JTLogger (Error diagnostics)

It is not unusual after a code change that MARTe crashes during the initialization phase due to configuration errors. In this case it could be very useful to see the log messages to debug the eventual errors or mistakes. A good practice is to put more assertions as possible when developing new code. To see the assertions generated from the MARTe *AssertErrorCondition(...)* function we should use the provided *JTLogger* tool. It has to be launched before the MARTe simulation:

```
- $ssh -Y [user name]@[FTU cluster IP address]
- $cd code/BaseLib2/BaseLibTools/JTLogger
- $./JTLogger.sh
- Click on the button with the two displays picture and set the
same address of LoggerAddress parameter of your cfg.
- The address is always 192.107.90.23x where x is the number of
the cluster you are connected with (fusc02 is 192.107.90.23x,
fusc05 is 192.107.90.235, ecc.). Check that the address set in the
```

```
logger is equal to the address in your cfg and to the number of the
cluster you are connected with, otherwise the logger will not work.
- Clean the logger console with the third button before running a new
simulation.
```

2.7.3 Cint

The cint tool must be used after changing the input or output signal structure of a GAM. This tool will refresh the structure meta-data which will be used from the MARTe framework to handle the signals connections between GAMs. The tool generates the file [GAM name]ClassInfo.sinfo.cpp taking in input [GAM name]OutputStructure.h [GAM name]InputStructure.h and [GAM name]ClassInfo.h. After an alteration in [GAM name]OutputStructure.h or [GAM name]InputStructure.h we can run a shell script in order to re-generate [GAM name]ClassInfo.sinfo.cpp.

```
- change your GAM input or output structure in
  [GAM name]InputStructure.h or [GAM name]OutputStructure.h
- $cd code/MARTe/GAMs
- $source cintScript.sh [GAM name]
- check if the new struct member appears in [GAM name]ClassInfo.sinfo.cpp
```

Follows the content of *cintscript.sh*

```
#!/bin/sh
export PATH=$PATH:/home/[user]/cint-5.18.00/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/[user]/cint-5.18.00/lib
export CINTSYSDIR=/home/[user]/cint-5.18.00
echo 'exported cint paths'
cd /home/luca/[user]/MARTe/GAMs/$1
rm $1ClassInfo.sinfo.cpp
make -f Makefile.linux clean
make -f Makefile.linux
cd -
```

2.7.4 See results on local machine

It is possible to launch and see results of a MARTe simulation (which will be performed on remote FTU server) on your local machine. First of all, in order to avoid to give the password every time we are connecting to FTU cluster, edit a script called *exp* and write inside the following code:

```
#!/usr/bin/expect
set timeout 20
```

```
set cmd [lrange $argv 1 end]
set password [lindex $argv 0]
eval spawn $cmd
expect "password:"
send "$password\r";
interact
```

After that, save it in *usr/bin* path or export the path in the environment variable PATH.
In your working directory create the script runSim.sh with the following code.

```
#!/bin/sh

CPPFILES=[local code to be copied and compiled on remote dedicated server]
SSHPASSW=[the FTU cluster password]
export REMOTEDIR=[the FTU cluster path where the scripts are]

#copy the cpp files in the cluster folder
exp $SSHPASSW scp $CPPFILES [user name]@[FTU cluster IP address]:
$REMOTEDIR/cppSourceMARTeFile
#copy the initialization parameters for matlab simulation
exp $SSHPASSW scp initParams.m [user name]@[FTU cluster IP address]
:$REMOTEDIR
#generate the remote matlab launcher
rm launch_remote_matlab.sh
echo -e "#!/bin/sh\nssh [user name]@[FTU cluster IP address]
<< 'ENDSSH'\nsource
$REMOTEDIR/launch_matlab.sh\nENDSSH" >> launch_remote_matlab.sh
chmod 777 launch_remote_matlab.sh
#run the matlab simulation
exp $SSHPASSW ./launch_remote_matlab.sh
#copy here the matlab data
exp $SSHPASSW scp [user name]@[FTU cluster IP address]:
$REMOTEDIR/signalsremote.mat .
```

setting the variable *CPPFILES* with the list of files you want to copy on remote server and *SSHPASSW* with the ssh password of the FTU cluster. The variable *REMOTEDIR* contains the path of the folder on FTU cluster where there are the script to launch the simulation on remote FTU server. In addiction to the source code file, the configuration file and an *initParams.m* file will be copied on cluster. Inside *initParams.m* the variable *cfg* must be set with the name of the configuration file and the variable *shot* with the shot number.

On the cluster folder there is the script *launch_matlab.sh* which basically executes the simulation on remote FTU server, same as explained at the begin of this section. The *launch_matlab.sh* script is the follow:

```
#!/bin/sh
[MATLAB folder]/bin/matlab -nodisplay
-nosplash -r "cd $REMOTEDIR; runSims_on_LucaAtBoncagni"
exit
```

The file *runSims_on_LucaAtBoncagni.m* launches the script *startSim_remote.sh*:

```
#!/bin/tcsh -f

set REMOTE_MARTE_DIR=[MARTe code path on dedicated server]
eval ‘ssh-agent -c‘

ssh-add .ssh/id_rsa

cd cppSourceMARTeFile
make_backups.sh $REMOTE_MARTE_DIR
cd -
scp  -i .ssh/id_rsa.pub   data_actual.dat [dedicated server IP]:
/home/luca/TestArea/
data_actual.dat
scp  -i .ssh/id_rsa.pub   zero_actual.dat
[dedicated server IP]:$REMOTE_MARTE_DIR/MARTe/TestArea/zero_actual.dat
scp  -i .ssh/id_rsa.pub   $1 [dedicated server IP]:
$REMOTE_MARTE_DIR
ssh  -i .ssh/id_rsa.pub   [dedicated server IP]
"cd $REMOTE_MARTE_DIR/MARTe/GAMs/SafetyGAM/;
make -f Makefile.linux"
ssh  -i .ssh/id_rsa.pub   [dedicated server IP]
"cd $REMOTE_MARTE_DIR/MARTe/GAMs/ControllerGAM/;
make -f Makefile.linux"
sleep 1
ssh  -i .ssh/id_rsa.pub   [dedicated server IP]  "sync;killall MARTe.ex;
cd $REMOTE_MARTE_DIR/MARTe/TestArea;  ./MARTe-frascati.sh ../../$1" &
./download_remote.sh
ssh  -i .ssh/id_rsa.pub   [dedicated server IP]  "killall MARTe.ex;"
```

As we can see, this script launches the *make_backups* script which copies the source code on remote FTU server after making a backup of that files appending the current timestamp.

```
for i in *.cpp; do scp -i .ssh/id_rsa.pub [dedicated server IP]:
$1/MARTe/GAMs/ControllerGAM/$i backup/$(date +%Y-%m-%d-%H-%M-%S)$i; done;
for i in *.h; do scp -i .ssh/id_rsa.pub [dedicated server IP]:
$1/MARTe/GAMs/ControllerGAM/$i backup/$(date +%Y-%m-%d-%H-%M-%S)$i; done;
scp -i .ssh/id_rsa.pub *.cpp *.h
[dedicated server IP]:$1/MARTe2/GAMs/ControllerGAM/
```

Note that source code files are copied to the *ControllerGAM* folder. Change it if we are working in another directory. The second script *download_remote.sh*, runs the simulation on remote FTU server and downloads a mat file with all the selected results:

```
#!/bin/sh
rm MatlabSignalServer*
rm *.html
cd $PWD

MARTESERVER=[dedicated server IP]:[dedicated server MARTe port]
sleep 8
wget --user-agent=Mozilla/5.0  http://$MARTESERVER/BROWSE
/StateMachine/?StatusChangeRequest=PULSE_SETUP_COMPLETED
sleep 1
wget --user-agent=Mozilla/5.0  http://$MARTESERVER/BROWSE
/StateMachine/?StatusChangeRequest=PRE
sleep 18
wget --user-agent=Mozilla/5.0  http://$MARTESERVER/BROWSE
/StateMachine/?StatusChangeRequest=EJP
sleep 1
wget --user-agent=Mozilla/5.0  http://$MARTESERVER/BROWSE
/StateMachine/?StatusChangeRequest=COLLECTION_COMPLETED
sleep 2
wget --user-agent=Mozilla/5.0  --post-data 'ALLSIGNALS=Yes&
FORMSENT=Yes' http://$MARTESERVER/BROWSE/MatlabSignalServer
mv MatlabSignalServer signalsremote.mat
rm index.html*
sync
```

Thus, after we run the *runSim.sh* on our local machine, at the end we have downloaded the *signalsremote.mat* file with all the simulation results, and we can use matlab to observe them accordingly.

Chapter 3

TCV Control System

This chapter is dedicated to the description of the work carried out on the tokamak á configuration variable (TCV) control system

3.1 Introduction

The tokamak á configuration variable (TCV) located at Ecole Polytechnique Fedéral de Lausanne is a medium size tokamak characterized by a highly elongated rectangular vacuum vessel. Its particular features allow to explore and analyze specialized plasma shaping and heating capabilities of the physics of magnetically confined plasmas, partly to support the ITER project. The work carried out together with Daniele Carnevale and Mateusz Gospodarczyck, with the support of the internal TCV team, consists in the development of the control system of the runaway beam. During experimental campaigns we tried to trigger runaway activity by well-known recipes (low pre-fill + Ne injection ...) and then to control the beam slowly discharging the plasma current due to avoid disruptive effects. Mainly because of the too small toroidal magnetic field (max 1.54 Tesla against the 8 Tesla of FTU), it is very rare to observe a runaway plateau effect and generally the runaway electrons are much less energetic with respect to FTU.

3.2 TCV actuators overview

The TCV tokamak actuators for plasma shape and position control consist in 16 poloidal coils. The external eight are called F-coils, the internal are the E-coils. By providing currents with the same direction to the F-coils and currents with opposite direction to the E-coils, we generate a vertical magnetic field which is the sum of E and F coils contributes. Moving the center of this vertical field, we are capable to control the radial plasma position.
Contrariwise, by applying currents with opposite direction to the F-coils in the upside

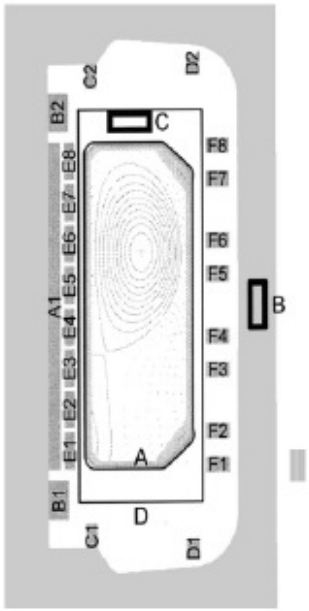

Figure 3.1: TCV poloidal cross section showing the ohmic transformer coils A, B, C, and D, the shaping coils E and F.

and the F-coils in the downside (it is recommended to choose the center as close as possible to the desired plasma vertical position reference because here the lines of the radial magnetic field are less distorted) and, respectively, currents with the same directions on the specular E-coils, we generate the radial magnetic field to control the plasma vertical position. Figure (3.2) shows the lines of the vertical and radial magnetic fields at TCV.
Referring to fig. (3.1), two ohmic coils circuits OH1 and OH2 are used for plasma current control, the first powers the central solenoid A, the second powers the coils B, C and D connected in series. A fast G coil controlled analogically is used as a fast actuator for the vertical plasma position and it is crucial during elongated plasmas when vertical instabilities occur.

3.3 Control System Overview

The Matlab-Simulink environment of the digital control system of TCV provides a flexible platform for implementing new algorithms. The new implementations can be previously simulated on a local machine or on the TCV clusters, acceding to a MDS plus data base that contains all the available diagnostics for each single shot. As for FTU, the simulations are in open-loop because of the lack of a valid model of the whole tokamak+plasma system. Once the Matlab-Simulink code have been

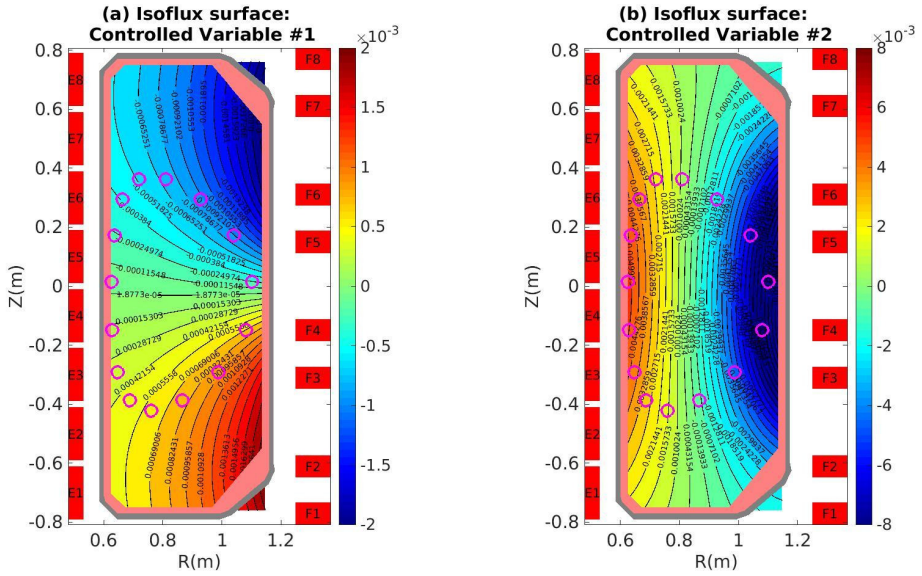

Figure 3.2: The isoflux surfaces of the radial magnetic field on the left (for vertical position control) and of the vertical magnetic field on the right (for radial position).

sufficiently tested, it will be traduced in real-time C++ code and compiled to build the control system for the experiment.
Figure (3.3) shows the simulink block scheme of the whole digital control system highlighting the new blocks added to the basic control system architecture for run-away control. Follows a rapid description of the blocks involved: for a more detailed overview on the basic TCV digital control system please refer to the online TCV documentation.

- The 128 measurements that include the currents on the poloidal coils, the flux loops and the current difference between OH1 and OH2, are given in input to the A matrix that builds by linear operations the 24 signals to be controlled (plasma current, radial displacement, vertical position, density, elongation, etc.). These signals are subtracted to their respective references providing the tracking errors to the controller block.

- The standard PID controller block is composed by three G matrices: G1 contains the gains of the integral, G2 the proportional, G3 the derivative. Consequently, the error signals are digitally integrated before going in input to G1 and derivate before G3.

- The control signals, sum of the outputs from the G matrices, cover the voltages to be provided to the 16 poloidal E-F coils and to the ohmic OH1 OH2 coils. The controls are given in input to the M matrix that corrects voltages for mutual inductances and resistive contributions.

- The M matrix outputs are summed to the respective pre-programmed control signals.

Follows a description of the blocks that have been introduced specifically for runaway control experiments:

- The *new_ref_gen* block takes in input the reference signals of the plasma current and radial and vertical positions that have to be changed in case of runaway activity detection.
- The *new_controls* block takes in input the plasma current and radial and vertical plasma positions with their respective references to implement alternative controllers to be employed in case of runaway activity to control the beam.

The same algorithms developed for FTU have been ported on TCV maintaining the same implementation, unless for the necessary changes to adapt them to a different tokamak architecture. A Simulink block library has been specifically developed to group the tools and algorithms to be used for runaway control on TCV.

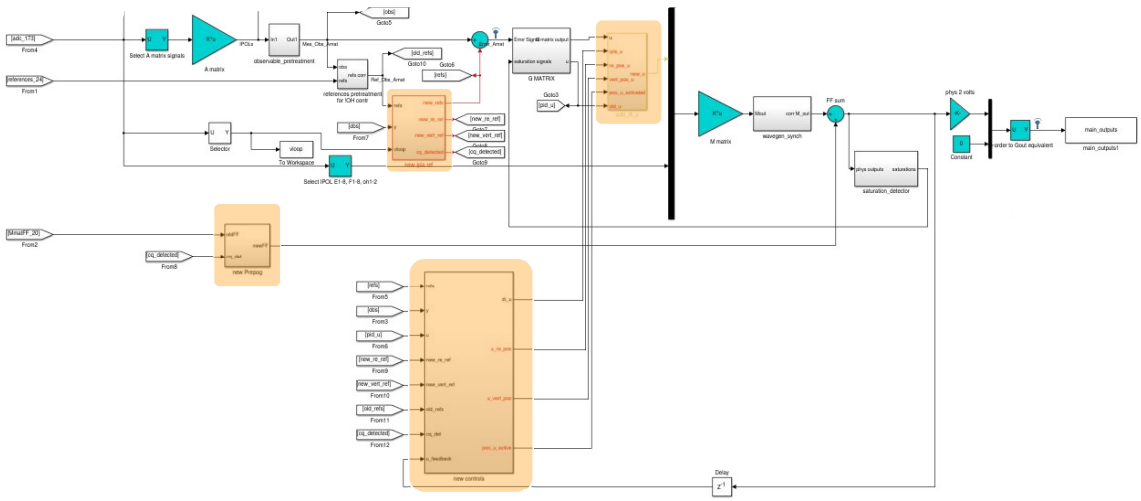

Figure 3.3: Simulink scheme of runaway control system at TCV

3.4 Plasma Current Control

The plasma current control actuators are the two ohmic coils called OH1 and OH2. Differently from FTU, the considered control signals are the coil voltages and not their currents thus, neglecting the low resistance of the coil, we can assume the coil voltage proportional to the derivative of the flowing current. For plasma current control on

FTU, where the control signal is the current flowing in the central solenoid T, we added double integrator; for TCV, being the control signal the voltage of the ohmic coils, a single switching integrator is sufficient to address null tracking error. The basic control system for plasma current control is a PID but, by default, only the proportional is activated leading constant plasma current tracking error. Imposing the integral gain different from zero to the already implemented PID controller might improve the tracking performance but a switching logic ensures less overshoot and oscillations. The switching controller has been implemented in a configurable mode in order to implement different types of linear controllers. In particular we can define a set of matrices A, B, C, D so that the controller output evolves as

$$\dot{x}_c = Ax_c + Be_{I_p}$$
$$u_s = Cx_c + De_{I_p}$$

with u_s to be added to the basic PID control u_{PID}, when the plasma current error e_{I_p} and its derivative $\dot{e}_{I_p}$ are within pre-fixed ranges. The discharge system has been defined trough the matrices $A_{dis}, B_{dis}, C_{dis}, D_{dis}$. The switching policy is the same of the switching double integrator developed on the FTU control system described in section (2.4). In the case of a switching sigle integrator, the block (fig. 3.4)has been configured with the following matrices: $A = 0$, $B = K_i$, $C = 1$, $D = 0$ and $A_{dis} = -1/\epsilon$, $B_{dis} = 0$, $C_{dis} = 1$, $D_{dis} = 0$.
Figure 3.5 shows the plasma current tracking and the switching integrator control action for the shot 57781. Note that after the switching controller activation at 0.3s, the plasma current tracking error is reduced thanks to the integral action.

3.5 Plasma position control

TCV has unique features that make it particularly suitable for plasma shaping tasks. This is possible thanks to the 16 poloidal coils that surrounds the vacuum chamber, F1 to F8 externally and E1 to E8 internally. By default only the coils F3, F4, F7, F8 are used to control the radial and vertical plasma position. Also in this case, the standard controllers are PIDs, in particular a simple proportional has been employed for radial position control and a PD for the vertical one. The computed control values have to be split wisely on the actuators, because each coil can have a part in both radial and vertical position control. For instance, the basic radial control has been divided equally on the four F activated coils with the same sign; the vertical control has been divided equally too but provided with positive sign to F7 and F8 and with negative sign to F3 and F4. The coordinates of the plasma position are rIp and zIp: in other words the plasma's radial and vertical displacements are current weighted and so it is the tracking position errors that are given in input to the standard controllers. An alternative implementation of the standard controllers has been developed to

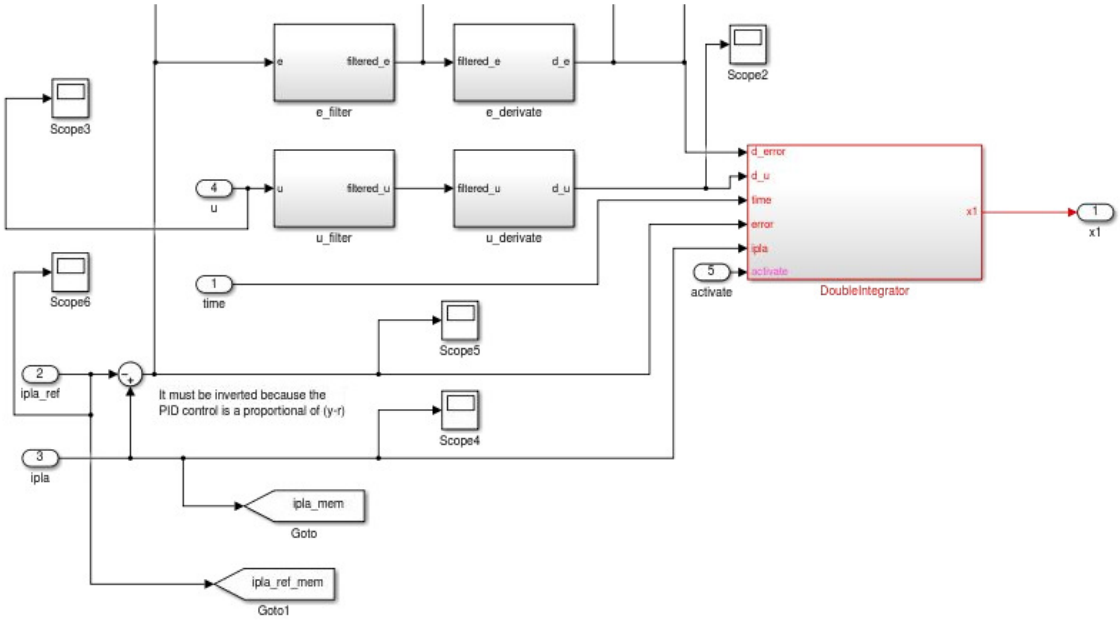

Figure 3.4: Simulink scheme of the switching integrator on plasma current: x_1 will be added to plasma current PID regulator

improve the plasma position control performance. It has been chosen to scale the original radial and vertical position coordinates by removing the current weighting. Figure (3.7) shows the block scheme that builds the radial or vertical plasma position that is given in input to the new controller. The new controllers replace the standard PIDs and include new hybrid algorithms. The implementation of the alternative radial controller has been replicated exactly for the vertical one, thus a single description is sufficient.

The new PID is composed by a proportional block, a derivative block and a switching integrator block (the same added to the plasma current controller, but the switching integrator is generally set-up with e and $\dot{e}$ ranges large enough to be always active as a normal integrator). A fast ramp controller adds its action to the PID's one.
The computed control signal has been split on the 16 poloidal coils by a configurable vector of 16 weights (one for each coil), allowing the possibility of involving a greater set of coils for plasma position control. By observing experiments and documentation and in some cases the RZIP model developed specially for TCV tokamak, the E coils have been half-weighted with respect the F coils, but these weights might be refined successively. Again, the gain vectors must take account of what explained in section (3.2) about the current versus of the poloidal coils for vertical and position control. For instance, if the coils E3, E4, E7, E8, F3, F4, F7, F8, have to be employed, the

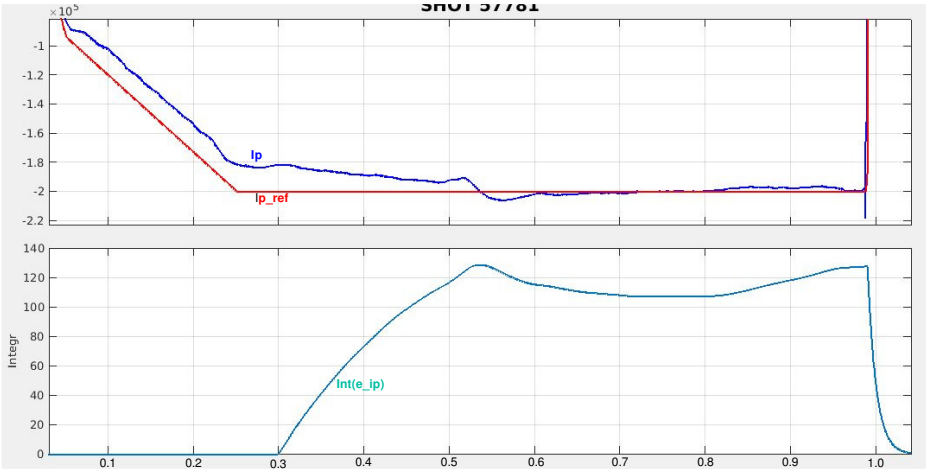

Figure 3.5: Plasma current control of the shot 57781. I_p tracking on top and switching integrator output on bottom. The switching integrator control is active from 0.3s

vectors have to be set as follows:

$$r_{gains} = [0\ 0\ -0.5\ -0.5\ 0\ 0\ -0.5\ -0.5\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1]$$
$$v_{gains} = [0\ 0\ -0.5\ -0.5\ 0\ 0\ -0.5\ -0.5\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1]$$

r_{gains} are the gains for radial control, v_{gains} for vertical.
A fast ramp controller adds its action to the PID's one. The implementation reflects exactly the fast controller MARTe block developed on FTU control system, but reported in Simulink, its description has been already provided in section (2.5). Figure (3.10) shows the block scheme of the fast ramp controller developed with the help of the student Marco Passeri during his master degree thesis period. The possibility of configuring the poloidal coils to be used during the experiments has improved significantly the control performance with respect the default controller. Besides the radial controller gains should be still refined (the performance are comparable with the default controller), the vertical controller leads to a position tracking much better with respect the vertical standard PID controller.
Figure 3.8 shows the plasma position radial tracking and control for the shot 58318. The new developed controller replace the default one at 0.25 s and we can note that the tracking performance are very good. Regarding the vertical position control, figure 3.9 shows the vertical position tracking and control for the shot 57781. The new vertical PID controller is active from 0.3s replacing the default one and there we can note a remarkable reduction of oscillations in the control action. Also the tracking is almost perfect.

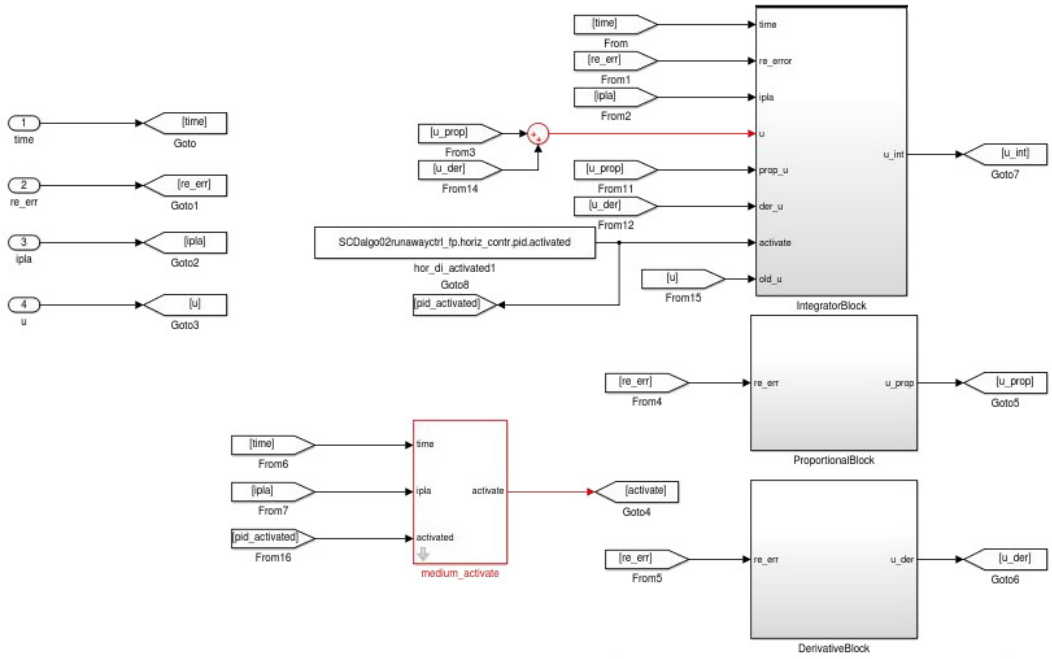

Figure 3.6: Simulink scheme of the PID implemented for radial control (the scheme is equal to the PID for vertical control)

3.6 Soft Discharge

Same as for FTU, also on TCV we have implemented a soft discharge algorithm to shut down the experiment in case of dangerous runaway activity. The plasma current reference is ramped down when a current quench followed by a runaway plateau has been detected, as explained in section (2.6.2). Moreover, since has been observed a radial outward shift of the plasma, also the radial position reference is reduced to the center of the vessel.
Figure (3.11) shows the block scheme implemented in Simulink which consists in:

- a current quench detector block that asserts when a runaway plateau has been detected. It can detect multiple runaway plateaus.
- a I_p reference generator that changes the default plasma current reference ramping it down to zero with a configurable slope when a runaway plateau has been detected.
- a V_{loop} controller that provides a negative value δ_V proportional to the I_p value when the loop voltage V_{loop} becomes greater than a configurable threshold. This δ_V is added to the I_p ramp down reference (after runaway plateau detection) to avoid to stress the ohmic actuators. Infact, when the loop voltage is huge, the

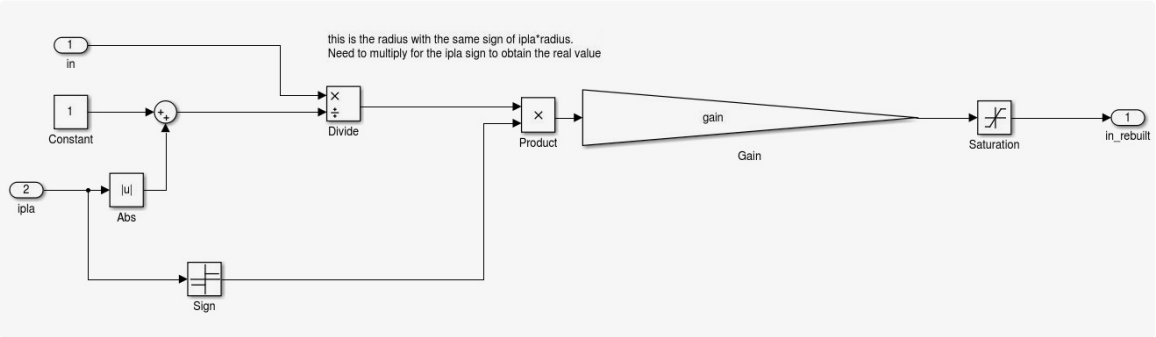

Figure 3.7: The block schemes that builds the considered radial (or vertical) position from the original signal rIp (or zIp for vertical)

plasma current is far away from its reference or MHD activities occur. In both cases reducing the I_p reference can be crucial to prevent damages.

- a square wave generator that adds a square wave with configurable amplitude and frequency to the I_p ramp down (after runaway plateau detection). It has been rarely activated to see if the square wave action accelerates the expulsion of runaway electrons obtaining a better runaway beam mitigation.

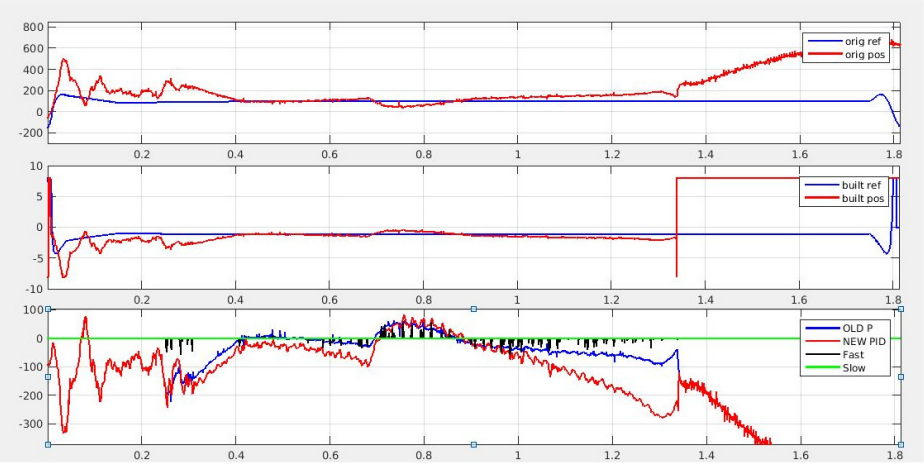

Figure 3.8: Radial plasma position control of the shot 58138. The plasma radial position tracking on top using the default coordinates rI_p, on center the built coordinates. On bottom the action of the used PID in red, the action that the default PID in open loop in blue, and the fast ramp controller action in black. The new PID radial controller is active from 0.25 s

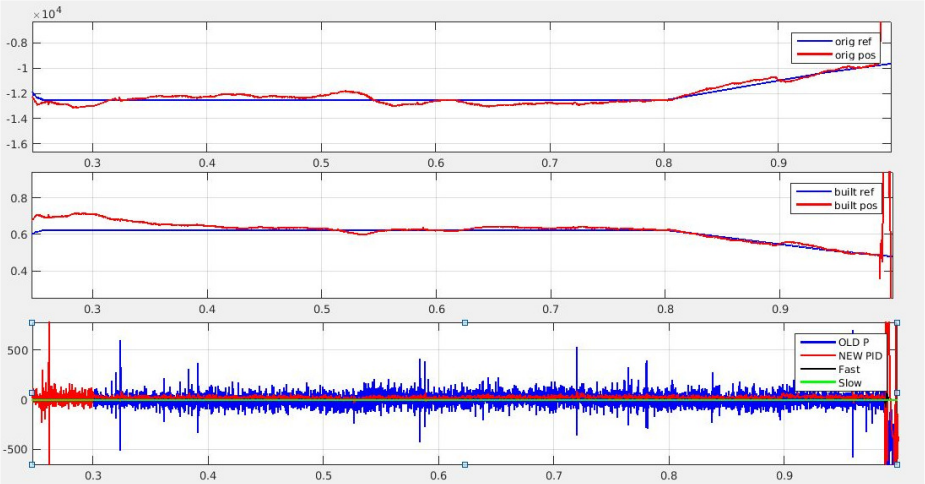

Figure 3.9: Vertical plasma position control of the shot 57781. The plasma vertical position tracking on top using the default coordinates zI_p, on center the built coordinates. On bottom the action of the used PID in red, the action that the default PID in open loop in blue. The new PID vertical controller is active from 0.3 s

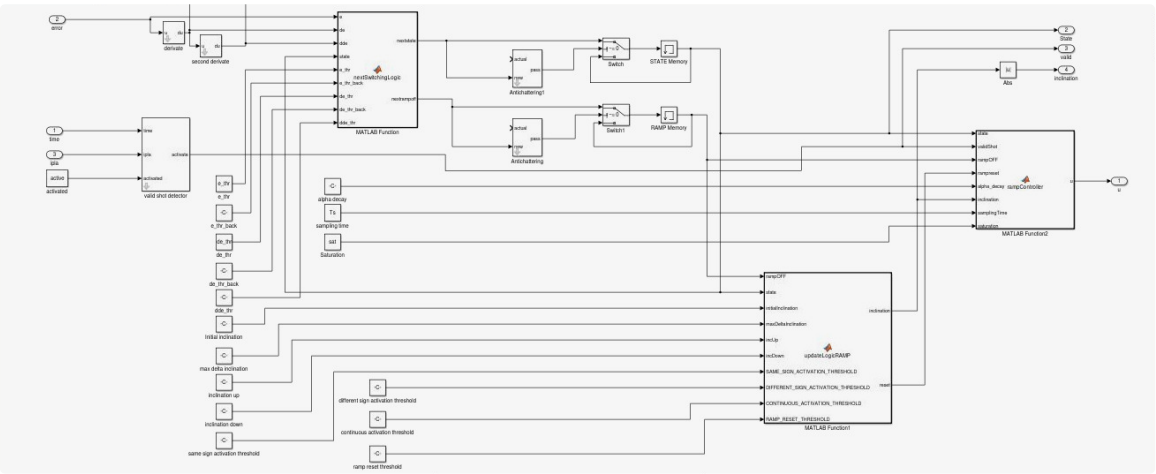

Figure 3.10: The fast ramp controller block scheme

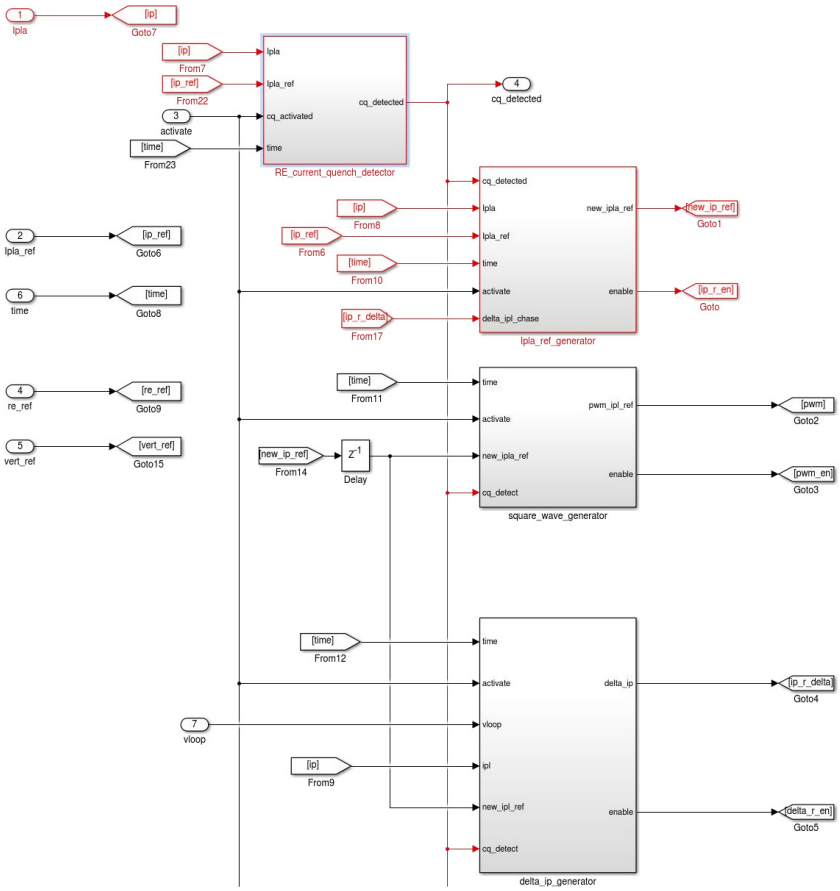

Figure 3.11: The fast ramp controller block scheme

Chapter 4

MARTe2 for robotics

This chapter is dedicated to the description of the new developed version of MARTe framework (MARTe2) and its employment on embedded boards for robotic and real-time applications.

4.1 MARTe2 Quality Process

Software frameworks are usually associated with the need to satisfy common and transversal requirements in a given development context. When a project chooses to take advantage of an existing software framework, the development team expectation is to be able to save effort in the full development cycle domain, i.e. from design to integration. As a consequence, a software framework is expected to be robust and to behave coherently with its specifications. This requires the existence of appropriate and complete documentation, complemented by thorough testing which exercises the framework in all of the anticipated use-cases. Moreover, it is also crucial to have a mechanism which guarantees that any defects found in the framework can be properly signaled and managed by the framework developers. All of the above can only be appropriately handled by guaranteeing that the framework has a sound quality assurance process associated to it.
The MARTe software framework [38] is C++ modular and multi-platform framework for the development of real-time control system applications. As of today, it has been deployed in many fusion real-time control systems [39], particularly in the JET tokamak [40]. One of the main advantages of the MARTe architecture is the bold separation between the platform specific implementation, the environment details and the real-time algorithms (i.e. the user code). The platform is defined by the target processor and the operating system, while the environment encapsulates all the interfacing details which are related to the peculiarities of the location where the final system is to be deployed. This includes both the interfacing with the hardware platform and the binding to the services that allow to configure and retrieve data from the system. This clear separation of concern has allowed to reuse many components

inside the same environment (e.g. all the systems deployed at JET share the same services for parameter configuration and data retrieval) and to develop and test the user algorithms in non-real-time operating systems and to later deploy the same exact code in a previously tested platform.
As more systems started to use MARTe, the number of supported environments and platforms has considerably grown. This has leveraged the exposure of the same core code to different environment configurations, thus increasing the confidence on its quality and robustness. Having the same infrastructure being used inside a community has also had the advantage of sharing and recycling knowledge about the framework and its architecture. In the context of an internal ITER fast controller [41] prototype project, which aimed at testing the integration of fast plant systems in the ITER software environment, a new version of the MARTe framework has been developed. One of the main objectives of this activity was to develop a software Quality Assurance (QA) strategy that is appropriate for the development of ITER real-time applications (e.g. diagnostic control systems). In particular the QA process had to be designed in order to safely integrate contributions from a large and heterogeneous development community, which includes developer profiles both from the scientific community and from the industrial suppliers.

4.1.1 Project Organization

As discussed before, one of the main objectives of this exercise was to enable a development which would enable external contributions from a very heterogeneous community, without compromising the overall quality of the project. As it can be seen in Fig. 4.1 the selected model comprises three main actors. The brainstorming group discusses and proposes requirements for MARTe. The coordinator decides which requirements are feasible and relevant for the framework and integrates them into the developing documentation. Finally, the developing team designs and implements the software accordingly to these requirements. It should be noted that it is possible to share members between the brainstorming community in the development team, provided they strictly abide to the QA processes.

Deliverables

The main project deliverables are listed in table I. In particular, with each release of the framework, a new version of each of these deliverables is to be issued. The requirements deliverable describes the features that the framework must meet and its constraints. The architecture and design deliverable details the system architecture and its design. It contains a model of the main blocks of the framework, the expected interactions between components and the links between the requirements and the components. The software deliverable comprises the source code and API documentation, as well the unit tests and integration tests source code. The QA audit

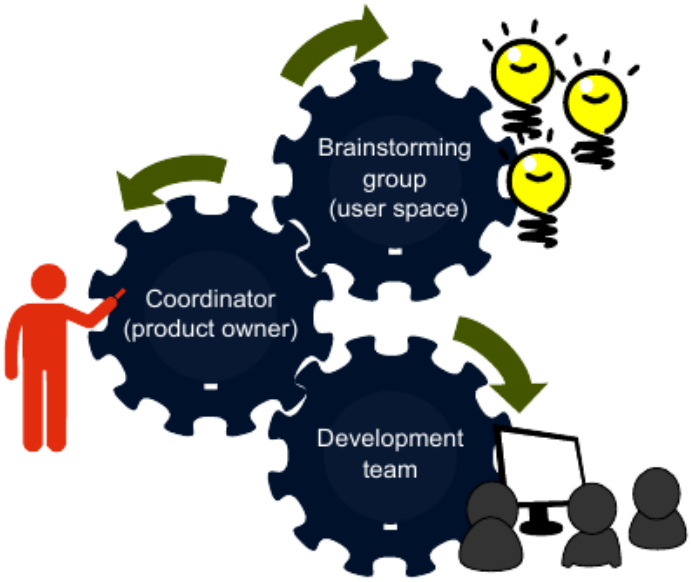

Figure 4.1: Top level project organization. The brainstorming group proposes requirements which are filtered by the coordinator and later delegated to the development team.

deliverable contains all the quality reviews made for the other deliverables and all the detected non-conformities. Finally, the traceability matrix shows the links between the requirements and the design classes.

Software activities

As shown in Fig. 4.2, the software activities are structured following a double-V-model approach where the first V (blue) is related to the development activities of the framework source-code and the second (red) implements the MARTe quality assurance. The double-V emphasises the fact that each of the activities has its traceable mirror.

The requirements activity translates the needs and objectives for MARTe, as defined by the stakeholders, into a proper set of requirements. The output is a model of requirements expressed in UML, which are grouped into fields of interest using UML packages. The review process verifies that all requirements have been correctly written, following a given set of rules, and defined in the requirements document. The architecture & design activity is driven from the requirements defined in the previous activity. The design includes a subset of classes considered the architectural foundation of the framework, complemented by any other classes which are needed for the framework to work. The review process verifies that the architecture and design are aligned with requirements and follow best practices. The responsible has to pay

Deliverable	Name scheme	Type
Requirements	MARTe requirements (vX.Y)	Report
Architecture & design	MARTe architecture & design (vX.Y)	Report
Software		
Source code	MARTe source code (vX.Y)	Source
API documentation	MARTe API documentation (vX.Y)	Report
Unit tests	MARTe unit tests (vX.Y)	Source
Integration tests	MARTe integration tests (vX.Y)	Source
QA audit	MARTe QA-audit (vX.Y)	Report
Traceability matrix	MARTe traceability matrix (vX.Y)	Report

TABLE I
MARTE DELIVERABLES FOR EACH RELEASE, WHERE X DENOTES A MAJOR VERSION AND Y A MINOR VERSION.

special attention to possible redundant modules and into the global coherence of system. The code and documentation review process includes the manual verification of the API documentation and the verification that a set of best practices were followed during the code implementation. This activity is complemented by the manual verification of the API documentation generated from the comments in the source-code and by the execution of an automated tool which verifies the compliance of the source-code against the coding standard. The software activities are concluded with the development of unit and integration tests. The unit test classes will implicitly trace the implementation classes of the source code, because it is assumed that each unit test class tests only one implementation class. The integration tests aim at exercising the architecture classes in the widest set of representative use-cases. The review of the unit tests is divided in a static and in a dynamic analysis phase. In the former, the reviewer verifies how many public functions of the source code have unit tests defined (black box unit testing is assumed). The latter, calculates what percentage of code has been executed (white/grey box unit testing is assumed). In both cases, code with a low coverage percentage ($< 80\%$) will be rejected.

4.1.2 Quality assurance

As depicted in Fig. 4.3, the QA system is described in five different documents (which combined provide an integral processes). The Project Management Plan (PMP) details the software development process described above. The Quality Assurance Plan (QAP), establishes the process and procedures that are used to achieve the objectives of the quality assurance process. In particular, it defines the role of the Quality Officer (QO) and its main responsibilities, which include independent reviews and audits of all the data and processes involved in the development, production, and maintenance of the deliverables. The QO also verifies the degree of compliance against the project applicable standards, project plans and processes. This plan also imposes the template format of the audit documentation (inputs, outputs and checklists) and sets up a procedure for process improvement. The Configuration Management Plan (CMP)

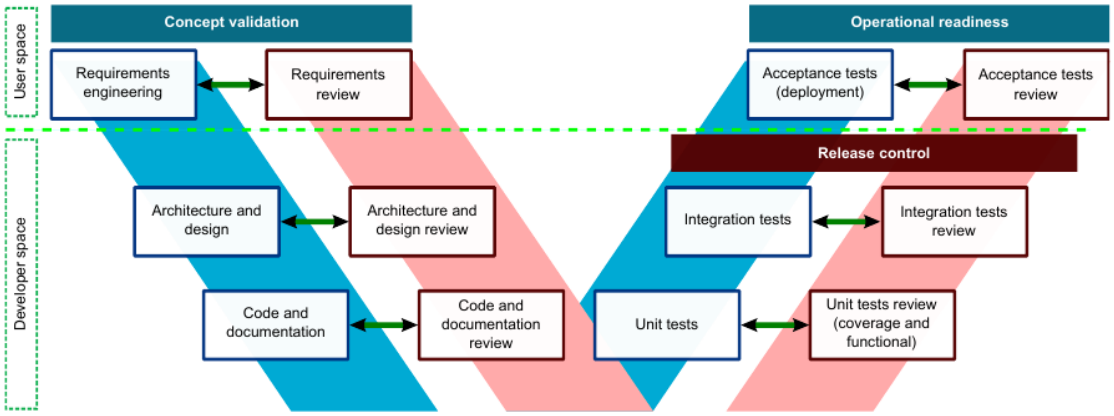

Figure 4.2: The blue V is in charge of developing the MARTe framework. The red V guarantees the framework quality assurance. It should be noted that this double-V-model does not prescribe any given software lifecycle methodology.

identifies the items which will be under configuration management and establishes the strategy for baseline and change control. The CMP defines that Git [42] is used as the version control system for the software development and imposes the workflow (described in section V). The Verification and Validation Plan (VVP) details the verification processes applied by the project to satisfy the software verification process objectives, including software testing, reviews and traceability. Specifically, it imposes that the verification team shall not have taken part in the development process, in order to ensure the independence between development and verification activities. Nevertheless, the VVP allows for the exchanging of roles during the life-cycle of the project (i.e. a developer may be a reviewer of another development activity, on which this developer was not involved). The VVP also defines the transition criteria between software activities (see Fig. 4.4) and the auditing templates in a format compatible with the issue tracking tool (detailed in section V). The Coding Standard document defines the main rules applicable to the source code development. The main goal of these rules is to assure the coherence among the code developed by the different team members and to also assure its quality and maintainability. In order to increase the robustness of the code and to avoid common errors and pitfalls, a controlled subset of the C++ language was defined for the MARTe framework. This subset is defined by means of a list of coding rules, which will address many of the dangerous aspects of the C++ language for critical systems. Thus, the C++ version used on MARTe will be that defined by the standard ISO/IEC 14882:2003 [43] (also known as C++03), while the coding rules will be those defined by the standard MISRA C++:2008 [44]. The MISRA C++:2008 is a set of software development guidelines for the C++ language targeted towards critical systems, which applies to the C++ language defined by the

Figure 4.3: Relationship between the QA plans. Quality, configuration manage- ment and verification are integral processes.

standard ISO/IEC 14882:2003. MISRA C++:2008 has emerged from the automotive industry, and is widely accepted as a model for best practices in sectors like aerospace, telecom, medical devices, defense, railway and others. MISRA C++:2008 also takes into account all those features of the language that are unspecified and undefined in the ISO/IEC 14882:2003. All the coding rules must be followed by developers at coding time, but formal reviews are also performed in order to ensure that all written code conforms to them. This is mostly performed by means of a static code checking tool (described later) which automatically detects violations to the rules in the code. For those rules which are not automatable, a manual review is performed before any major release. The output of this process is traced in a compliance matrix which will list all the rules and duly justifies any deviations. The coding standard also defines and imposes rules and guidelines for documentation, naming and code formatting.

4.1.3 Software lifecycle implementation

In order to implement the software activities described in section 4.1.1, the software lifecycle is managed using a mixture between a waterfall and an Agile [45] approach. Each user-story aims at developing the components required to satisfy at least one of the framework requirements. These user-stories live in the product backlog and at the beginning of every sprint, a subset is selected to be implemented in the scope of the sprint. During the sprint planning meeting the user-stories for the sprint are set in order of priority. During the sprint implementation, the development team moves the user-stories into the quality process depicted in Fig. 4.2. Each of the steps shown in Fig. 4.4 has to be reviewed by the quality responsible before the user-story is allowed

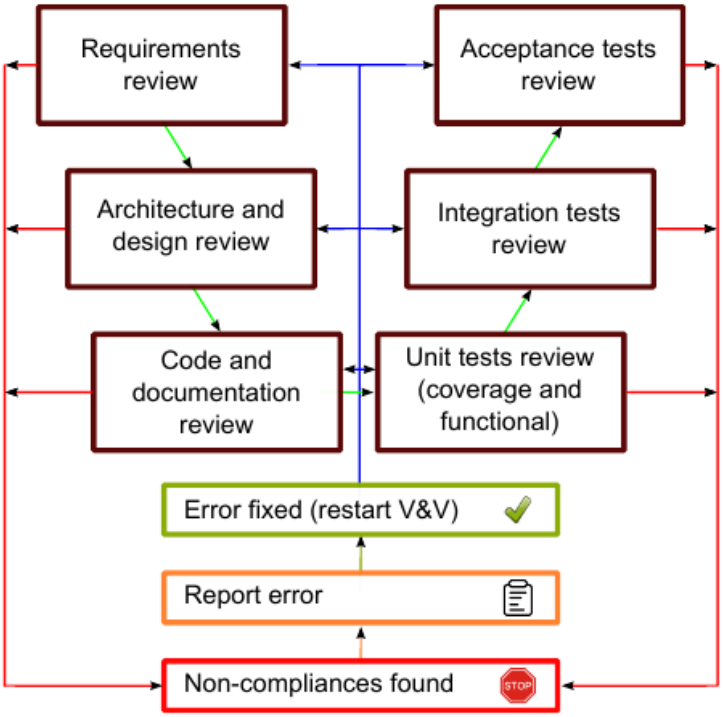

Figure 4.4: All the possible states and transitions of the verification and validation phases.

to move forward in the V-model. This means that while a user-story is being reviewed (e.g. code review) the developer is allowed to start (or continue) working in another story (e.g. the test implementation of a story which had its code reviewed already). Once a user-story successfully passes the integration test review, all of its components are set to be released with the next version of the framework. Following the spirit of an Agile development, the development team aims at meeting and reporting daily on the development status. Each sprint is concluded with a sprint review where the team demonstrates a potentially shippable product increment (i.e. that at least one of the user-stories has successfully passed the integration test review).
The process optimisation discussed before is implemented during the sprint review meeting, where the team members are asked to openly identify issues, which are then classified into one of the following three types: (i) things that the team should start doing; (ii) things that the team should stop doing; and (iii) things that the team should continue doing. Approved process improvements are then implemented by updating the appropriate standards, procedures, processes, checklists, or other related documentation (which are controlled configuration items). Software defects and other non-compliances are also translated into user-stories which follow this same QA process.

4.1.4 Development tools

The processes described above are supported by a set of tools which aim at assisting the developer at consistently meeting the coding standard rules, the quality reviewer at maximising the number of automatic verifications and the project manager at having a sound overview of the project status.
Source cove versioning is performed using Git and GitLab [46]. The Git workflow, which based in the Git branching model presented in [47], is described in Fig. 4.5 where it shows that each user-story is always created from the development branch. This branch is only merged back into development if it successfully passes all the quality checks. At the end of the sprint the development branch, which includes all the finalised user-stories, is merged into the release branch. All the quality auditing is performed over this branch and minor bug fixing is allowed. Finally, the release branch is merged into the master branch and a new tag with the version number created. It should be noted that after being merged into develop branch, the user-story feature is deleted. An hotfix branch is used to resolve critical bugs in the master version. When resolved, the branch will be merged back into the master and develop branches (if applicable it will also be merged to the release branch).
The Redmine [48] issue tracking system is used to manage the Agile workflow and to store all the quality reports, including the audits and the Agile sprint planning and review meeting minutes. More specifically, each user-story is assigned to a redmine issue and its life-cycle managed using the Redmine Agile plugin [48] (see example in Fig. 4.6). The plug-in was configured in order to provide a one-to-one mapping with the V-model, greatly simplifying the management and overview of the QA review process.
Unit testing is performed using the google-test frame- work [50]. Nevertheless, and given that MARTe is also expected to be deployed in bare-metal systems (i.e. processors without an operating system), the google-test framework is only used as a front-end engine to execute the MARTe unit test class methods, which are written without any dependencies on the google-test framework. This allows for the same tests to be easily ported to another testing framework. Code coverage is implemented with gcov, the GNU Project Compiler Collection [51] coverage testing tool and is complemented with a graphical front-end named lcov [52].
Static code analysis and compliance to the MISRA C++:2008 standard is performed using the Gimpel Software FlexeLint tool [53]. Common project deviations to the standard are added to a file that is shared among all the developers. Specific deviations to the standard are directly justified in the source-code using a special syntax inside a C++ comment. Code is expected to be developed using the Eclipse CDT [54] integrated development environment (IDE). Eclipse is configured with the project formatting, code and editor templates, as specified in the verification and validation plan. The unit tests can also be executed from with-in the IDE by using the Google Tests Runner. The source-code documentation is generated using Doxygen [55] and

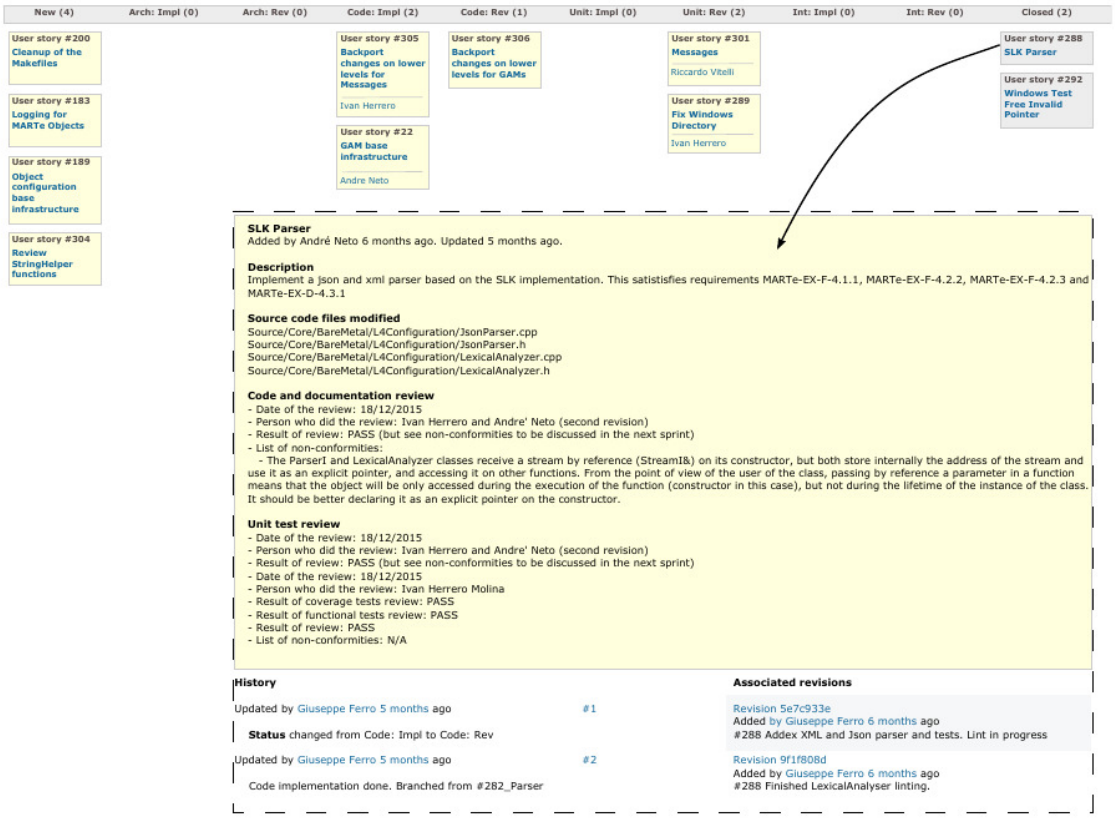

Figure 4.5: Snapshot of the Agile board. Each user-story is associated to a Redmine issue where its QA lifecycle is fully audited.

is integrated into the IDE through the eclox plug-in [56]. The output of the execution of FlexeLint has been also integrated into the editor, allowing developers to quickly navigate to the non-compliant lines of code and to use the static analysis tool as part of their development process. Finally, the complet management of the Git workflow can also be performed directly from with-in the IDE.

Software integration environment

The continuous integration environment is implemented using the Jenkins software [57]. This is configured to build and test the development version (i.e. from the Git develop branch) of the framework every night. In particular it reports on the testing coverage (using the cobertura [58], sonar [59] and JUnit reporting plug-ins) and on the percentage of compliance generated by the static code analysis tool.

4.1.5 Lesson learned

As of today there were already 13 development releases of the framework following the QA system. There are approximately 30000 lines of framework code and around 45000 lines of testing code.
The compliance to the coding standard (and to the selected MIRSA rules) is of almost 100 %. Adjusting Flexelint requires time and expertise. A misconfigured static analysis tool will generate a large number of false positives and might fail to identify true errors. Given that the standard is very demanding the amount of linting errors on any given file can be very large (this number decreases significantly with experience). Not having the output of the tool directly integrated into the IDE would greatly increase the development time and possible jeopardise the usage of the tool. It should be noted that most of these linting errors would not prevent the compiler from building without generating any warnings. Moreover, being able to justify the deviations to the standard in the source-code increases the transparency of the QA process and facilitates the review process.

Regarding the unit tests, more than 2500 tests are currently being executed with a coverage in excess of 90 %. The code sections which are most difficult to test are the ones related to operating system or processor faults. Not using all the functionalities of the google-test framework limits the amount of features that could be used to facilitate the identification of errors. On the other side, being able to execute all the tests of the framework in a bare-metal system (based on the ARM R Cortex R -M [60] has proved to be crucial for this type of development. Driven by the limitation in the amount of development effort available, it was decided not to apply the coding standard to the development of the test files. Documentation is of extreme importance and is the process with less room for automation. As a consequence a large part of the reviewing time is spent on verifying the documentation. Concerning the QA processes, associating each user-story to a new git branch has proven to be crucial in order to be able to allocate and monitor the work in a controlled way. Having the user-story following the V-model allows to have full transparency on who is doing what. Even if the model has been working very successfully, the fact that the user-story has to be reviewed several times during its life-cycle could possibly be optimised and further aligned to the Agile model. In such case, the reviewers would only verify the finalised user-story after the developers had finished the coding and testing phases. Indeed, it already happened that a user-story, upon test implementation, has required to change the original code, consequently triggering the review process to start again. Having the user-stories reviewed using the same structure of the QA audit greatly simplifies the writing of the final auditing. The Redmine agile board has also allowed to greatly simplify the management of the user-stories and its associated life cycle. Aligning sprint with releases has also proven to work very well and allows to have all the QA reports consistent and with the same release numbers of the software.

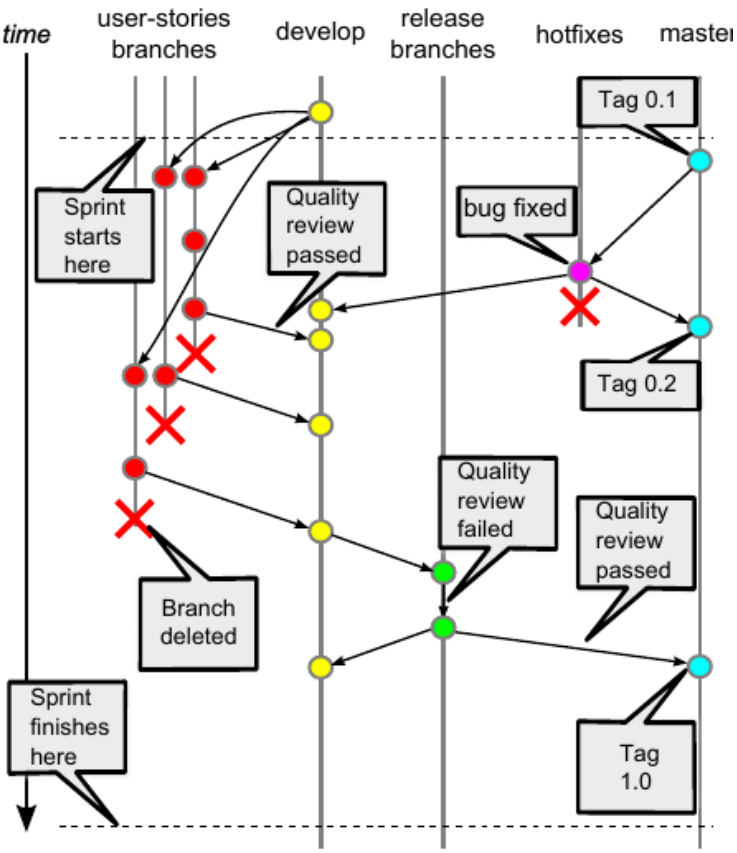

Figure 4.6: The chosen Git workflow, which is based on the branching model presented in [10].

4.1.6 Conclusions

The QA framework presented in this section can be easily adapted to the development of many types of software which are common in the fusion community, in particular for software related to control and data acquisition systems that is to be shared among different teams.
Even if the effort required to setup the QA system is not negligible, the major challenge is to make sure that it is applied throughout the full development process. This necessarily requires some extra-effort, but using some of the automation techniques described above the review time can be significantly optimised. In particular the deployment of a static analysis tool, fully integrated in the development editor, greatly increases the confidence that a large set of the coding rules was correctly applied. The obvious price of not following a complete QA system is that it will be very difficult to distribute and maintain the software, at least without the continuous support of the original developers, which in turn also has a significant trade-off cost associated to it.

The whole quality process processed on MARTe to obtain what here is called MARTe2, has been carried out with special attention to memory optimization and managing. Actually, one of the new features is that is possible to map different heap memories that can be managed within the MARTe HeapManager database. Another limitation of the previous version of MARTe was that it required an operating system to be executed. MARTe2 implements a better software architecture, being separated in three different tiers: bare-metal, scheduler and file-system, each one with its software abstraction levels. This way it is possible involve a sub-group of the entire framework for instance, on embedded architectures that do not provide file-system management or even without any operating system. That makes possible to process light versions of the framework making MARTe2 particularly interesting to be employed also on embedded architectures with limited amount of memory. 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 has been selected as the environment to prototype tests and robotic projects. Usually this board is coupled with a high-level device trough a serial communication in order to visualize diagnostic data or receive high level commands. In the developed project we have employed the Raspberry Pi boards as an high level device using the MARTe framework to visualize data and to provide an high level user interface.

4.2 MARTe2 on STM32F4-Discovery

The quality work on MARTe led to an optimization on the software architecture and to a development of new features. Follows a list of the main improvements from MARTe to MARTe2

- the MARTe core is called BaseLib; its code is divided in folders, each one representing an abstraction layer. The level 0 is the layer depending from the machine architecture and from the operating system. It is necessary to change the code in this level to make MARTe capable of supporting a new architecture. MARTe2 improved the software architecture by separating the BaseLib code in tiers, each one with its one stack of layers. The *BareMetal* tier implements all the code that does not need an operating system to be executed. In the low levels there are wrapper functions to access to the clock timer, to manage the heap memory etc. The *Scheduler* tier needs an operating system to wrap the primitives for a multi-task or multi-thread system. In the low level, depending on the operating system, the BaseLib functions wrap the primitives to launch and synchronize multiple threads and processes. The *FileSystem* tier contains all the necessary to manage file system utilities. Although the basic definition of the stream concept has been defined in the bare-metal tier, here specific streams are implemented to abstract files. By being able to compile a sub-group of these

tiers it is possible to support a greater number of architectures such as the embedded ones. For instance a big number of micro-controllers do not support any operating system or even a file system, so it is possible to compile just the bare metal tier obtaining a light version of the BaseLib for embedded architectures.

- the parser of the configuration file has been developed exploiting a table-driven LL(k) parser generator. By defining the grammar, the tool generates a parsing table that can be coded and compiled together with the rest of the BaseLib. Currently, in addiction to the standard MARTe configuration file grammar, it is possible to parse from XML and from Json configuration languages. Using this strategy is now easier to insert a new grammar.

- MARTe GAMs intercommunicate between them through signals shared in a unique database called Dynamic Data Buffer. The communication with external devices, peripherals or processes and the consequent thread cycle synchronization has been implemented by the collaboration of three components. The IOGAM calls the *Synchronize* method of a TimeTriggeringService (TTS) object that returns after that the GAcqM (Generic Acquisition Module) has triggered the synchronization (usually by calling the method *Trigger* of TimeTriggeringService). After the synchronization the IOGAM can transfer data from the GAcqM to the local buffer by the *GetData* method of the GAcqM and at the end put acquired signals on DDB. Depending on the project to be developed, it could be necessary to implement the custom GAcqM for the specific device that provides inputs, its related TTS and perhaps also the IOGAM. MARTe2 simplified the concept condensing TTS and GAcqM in an unique interface called DataSource that can abstract a shared database such as the old DDB (now called GAMDataSource) as it can implement the driver of a device. In a MARTe2 application, there are only interconnections between GAMs and DataSources, there is no need of IOGAMs, but simply a normal GAM connected to a DataSource different than the GAMDataSource. This new application architecture is particularly suitable for embedded projects, where each peripheral of the microcontroller can be abstracted by its own DataSource and GAMs can read or write from them multiple times during the thread cycle. The component that manages the communication between GAMs and DataSources is called *Broker* and its interface can be developed to allow the GAMs to access directly to the peripheral memory registers without passing trough intermediate buffers increasing the efficiency. In MARTe1, the interconnection with the external devices was directly implemented within the IOGAM interface and it need to be re-implemented for each new IOGAM. One of the advantages of MARTe2 with respect to MARTe1 is the possibility to use the same *Broker* to interconnect *GAMs* with different DataSources.

The STM32F4-Discovery board from STMicroelectronics® has been selected to test MARTe2 on an embedded platform. Follows the key features of this boards:

- STM32F407VGT6 micro-controller featuring 32-bit ARM® Cortex®-M4 with FPU core, 1-Mbyte Flash memory, 192 kbyte RAM in an LQFP100 package.
- On-board ST-LINK/V2 on STM32F4DISCOVERY or ST-LINK/V2-A on STM32F407G-DISC1 USB ST-LINK with re-enumeration capability and debug port interface
- Board power supply: through USB bus or from an external 5V supply voltage
- External application power supply: 3V and 5V
- LIS302DL or LIS3DSH ST MEMS 3-axis accelerometer
- MP45DT02 ST-MEMS audio sensor omni-directional digital microphone
- CS43L22 audio DAC with integrated class D speaker driver
- Two push-buttons (user and reset)
- USB OTG FS with micro-AB connector

Moreover it provides a great number of configurable peripherals:

- up to 14 timers that can be used to generate PWM signals, as encoder counter independent from the CPU, to generate interrupts on external trigger, etc.
- serial communication interfaces: UART, I2C, SPI.
- three ADCs, each one with up to 16 channels. The acquisition can be configured exploiting the DMA that manages the data transfer without disturbing the CPU.

This board is supported by FreeRTOS, a real time operating system (or RTOS) that has become the standard solution for micro-controllers and small microprocessors. It provides a scheduler to manage the execution of multiple tasks, and the basic synchronization primitives (semaphores) of a OS. It can be configured within the file FreeRTOSConfig.h to be preemptive, the amount of the available stack and heap, etc. By default the scheduler is synchronized on the *SysTick* STM interrupt at a period of 1 ms.
The development kit to use MARTe2 on embedded architectures consists in a palette of GAMs and DataSources that provide to the final user all the instruments to create easily an application without taking care of the low level programming. The challenge with the greater part of embedded platforms is that they are fully customizable. The first step is to define a configuration depending on the application requirements. Concerning the STM32F4-Discovery and, in general, the ST boards, it is possible to deploy

the STM-Cube tool that provides a graphic user interface to configure the ST board and to generate the related C code. It has been decided to include STM-Cube in the development kit and to implement the runtime peripheral work within the MARTe2 data sources. This approach has the advantage that, using a specific configuration tool, it is much more easy for the final user to configure the board. The tool actually shows all the available configuration options and prevents errors. The disadvantage, with respect configuring all by MARTe2, is that the platform configuration and the application development are completely decoupled, so it is up to the user be careful that the configured peripherals match MARTe2 DataSources configuration.

4.2.1 Board Configuration

The first step is to open STM-Cube, click on *New Project* and to choose the platform target. For STM32F4-Discovery, choose STM32F4 in Series, STM32F407/417 in Lines, LQFP100 in Package and select the STM32F407VGTx MCU with a flash memory of 1024 kbyte (figure (4.7)). The graphical interface shows four sections. The

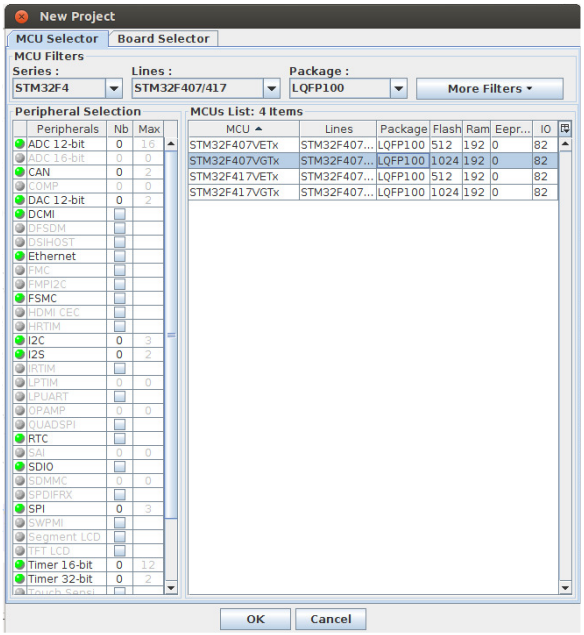

Figure 4.7: How to choose STM32F4-Discovery platform in STM-Cube configuration tool

Pinout section shows a graphic view of the LQFP100 core chip and on the left it is possible to activate the different peripherals in the desired mode. Once a peripheral has been enabled, the assigned pins will be marked (colored in green) on the pinout chip view. The tool prevents errors like to assign used pins to other peripherals and shows only the available peripherals possible configurations. It is possible to assign

manually the desired pin functionality directly in the chip pinout view.
The clock configuration section have to be used to configure the system clock. Configuring correctly the clock is crucial because doing it wrong prevents the correct behavior or even the startup of the application. The on-board crystal oscillator is 8 MHz frequency (HSE) and there is also an internal 16 MHz RC oscillator (HSI) within the MCU. The external crystal oscillator is much more precise than the internal one, so in the developed applications always HSE clock has been set. The system core clock frequency can be computed as follows

$$SystemCoreClock = HSE(Hz) \cdot \frac{N}{M \cdot P}$$

where *N, M, P* are the phase-locked loop (PLL) gains. The M gain is the divider factor and should be set as the value of the input clock frequency in MHz such that the PLL input is at 1 MHz. The N value is the multiplication factor to increase the system core clock frequency. The usual configuration to set the system core clock to 168 MHz, which nominally is the maximum for STM32F4-Discovery, is to set *M=8, N=336, P=2*. For applications that use the ADCs, since the peripheral has a limited frequency up to 75 MHz, the best configuration is to set the system core clock to 144 MHz such that the ADC can work at 72 MHz frequency (*SystemCoreClock/2*) that is closer to its maximum speed. Figure (4.8) shows the clock configuration for an application that uses the ADC peripheral. Once the system clock has been set,

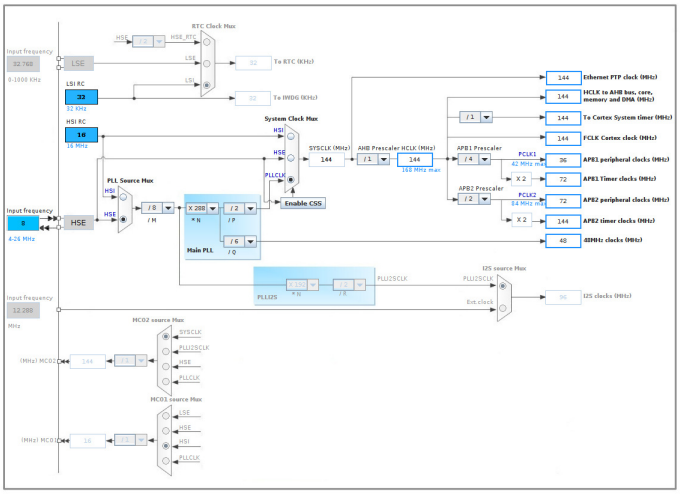

Figure 4.8: System core clock configuration

enable the use of FreeRTOS in the *Configuration* section on the left of the *Pinout* area. FreeRTOS nowadays is supported by many embedded platforms and allows to define and schedule multiple tasks. Given that MARTe2 can also be used without any operating system, it has been decided to make it compatible with FreeRTOS

and enabled it by default. In the *Configuration* area it is recommended to configure FreeRTOS as shown in figure (4.9), in particular enabling the *preemptive* mode and *heap_3* as memory management scheme. This scheme consists in dividing the RAM

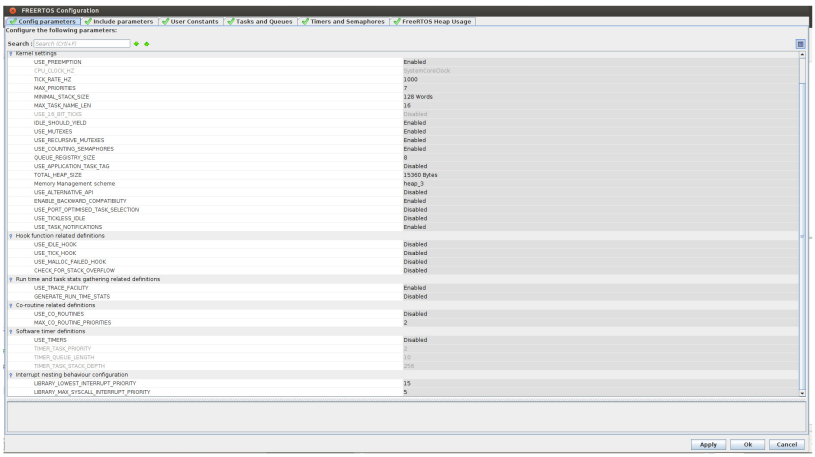

Figure 4.9: FreeRTOS configuration

memory in sections to be used as stack and heap memory areas by the concurrent tasks. Note that the default tick rate is 1 ms.
The next step is to define a timer that can be used as the MARTe *High Resolution Timer* (HRT). By default the timer TIM6 has been engaged for this work because it is a 16-bit independent basic timer. It is recommended to set its auto-reload period to 1 ms or in general equal to the *SysTick* timer period (by setting *Prescaler* and *CounterPeriod*) to calibrate the HRT timer by resetting it on the *SysTick* interrupt handler generated each time period. To obtain a resolution of 1 μs, the prescaled has been set to obtain a timer cycle frequency of 1 MHz and the HRT counter maximum value has been set to 999. Figure 4.10 shows an example of the TIM6 configuration used as HRT timer.

It is also recommended to enable at least a UART or USART serial port to be used as a logger interface, to allow the user to read errors or messages useful for debug.
Finally, it is possible to setup the desired board configuration, enabling or disabling the available peripherals in the *Pinout* area, and configuring them in the *Configuration* area.
Once the configuration has been done, click on *Project-Project Settings* and define the project path and the firmware package that can been downloaded from the ST website. Set the options *Other Toolchains (GPDSC)* in the *Toolchain / IDE* section. Then, go to the *Code Generator* area and check the *Copy all used libraries into the project folder* options. To generate the code, click on *Project-Generate Code* and all the configuration code will be generated into the selected project folder.

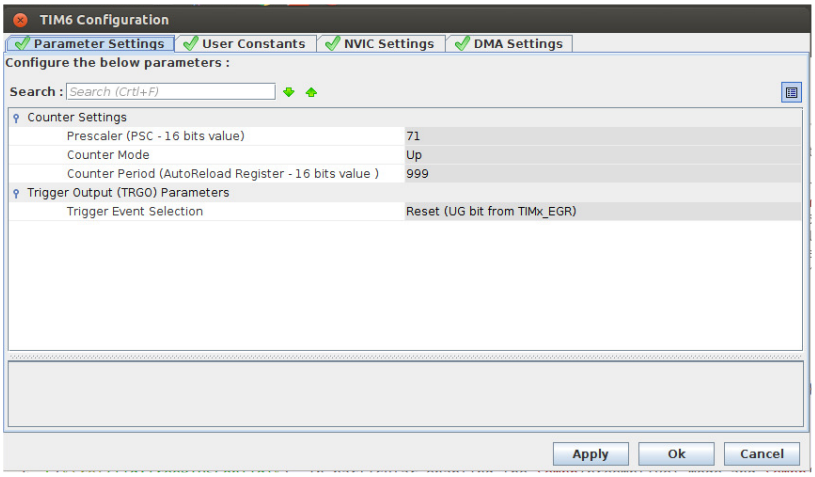

Figure 4.10: High resolution timer (TIM6) configuration with a resolution of 1μs

4.2.2 Build the Project

The whole code needed to develop a new project can be divided in three macro blocks. The first one is the MARTe2 *BaseLib*, containing all the framework code that abstracts the low level architecture and operating system calls and implements the definitions of GAMs and DataSources to build up MARTe applications. The second one is a tool-kit consisting in a set of most used GAMs and DataSources divided for each embedded platform that can be shared across different projects. The third one is the specific project code with its custom GAMs and DataSources.
First of all, it is necessary to compile the MARTe2 BaseLib by using the *make* tool to build *BareMetal* and *Scheduler* tiers.

```
cd BareMetal
make -f Makefile.gcc
cd ../Scheduler
make -f Makefile.gcc
```

The generated static libraries *Build/armv7m-freertos/Core/BareMetal/MARTeCoreB.a* and *Build/armv7m-freertos/Core/Scheduler/MARTeCoreS.a* must be linked to the final project.
The second step is to compile the needed DataSources and GAMs from the tool-kit to produce the static libraries to be linked to the project code. For instance

```
cd MARTe2-platforms/STM32/F4/Platforms/STM32F407-VGT6/DataSources/PWM
make -f Makefile.gcc
```

produces the *MARTe2-platforms/STM32/F4/Platforms/STM32F407-VGT6/PWMDataSource.a* library.

Finally compile the project including the needed static libraries from MARTe2 BaseLib and the tool-kit. The project code consists in the configuration code generated by the *STM Cube* tool added to the custom GAMs and DataSources implementations. The template of a new project consists in few folders used to separate the different parts:

- **ToolConfiguration** contains all the code generated by the *STM Cube* tool. It also contains a Makefile that generates the library *ToolConfigurationLib.a* to be linked to the project. This Makefile, before compiling code, implements few changes in the code of the generated *main.c* and *stm32f4xx_it.c* files.

```
update_main:
-grep -q UserMainFunction main.c ||
sed -i '1s/^/extern void UserMainFunction(const void *arg);\r\n /' main.c
-sed -i '/int main(void)/,/while (1)/ s/StartDefaultTask/UserMainFunction/'
main.c
-grep -q 'osThreadDef' main.c ||
sed -i 's/osKernelStart();/HAL_TIM_Base_Start((TIM_HandleTypeDef*)
GetHwHandle("TIM6")); osThreadDef(UserMainThread, UserMainFunction,
osPriorityNormal, 0, configMINIMAL_STACK_SIZE);
osThreadCreate(osThread(UserMainThread), NULL);&/' main.c
-grep -q 'CalibrateTimer' stm32f4xx_it.c ||
(sed -i 's/void NMI_Handler(void)/extern void CalibrateTimer();\n&/'
stm32f4xx_it.c && sed -i 's/osSystickHandler();/&\nCalibrateTimer();/'
stm32f4xx_it.c)
-ls Addons.hdd && grep -q 'Addons' main.c ||
sed -i 's/int main(void)/#include "Addons.hdd"\n\r&/' main.c
-if [ -f usbd_cdc_if.c ]; then cp usbd_cdc_if_template_c usbd_cdc_if.c; fi
-if [ -f usbd_cdc_if.h ]; then cp usbd_cdc_if_template_h usbd_cdc_if.h; fi
```

  After the board configuration code, it adds to *main.c* the code needed to create a new FreeRTOS thread launching the function *UserMainThread* that represents the MARTe2 application entry point. It adds also to *stm32f4xx_it.c* the *CalibrateTimer()* functions that resets the HRT timer (TIM6) within the *SysTick* auto-reload interrupt handler.
  Furthermore it adds the include of the *Addons.hdd* file in *main.c* that must be created by the user. This file must contain an array of [name, handle] couples for each peripheral or device that is going to be used, and the definition of the *GetHwHandle(const char*)* function that returns the pointer to the handle corresponding to the name given in input. This allows to link the DataSources to the peripheral handles generated by the *STM Cube* tool and let the DataSources to obtain the peripheral handles using a user friendly name to be specified in the configuration file. Follows an example of *Addons.hdd* file.

```
// to be added in main.c to get handles
```

```
UART_HandleTypeDef *errorUartHandle = &huart4;

struct identifier{
    const char *id;
    void *handle;
};


const struct identifier ids[]={
    {"ADC1", &hadc1},
    {"TIM6", &htim6},
    {"TIM4", &htim4},
    {"TIM2", &htim2},
    {"UART4", &huart4},
    {"UART2", &huart2},
    {"SPI2", &hspi2},
    {"GPIOD", GPIOD},
    {0,0}
};


void *GetHwHandle(const char *id){
    int i=0;
    while(ids[i].id!=NULL){
        if(strcmp(ids[i].id, id)==0){
            return ids[i].handle;
        }
        i++;
    }
    return 0;
}
```

This file could be also automatically generated, but currently it is up to the user to add it to the project.
Finally, before compiling the code, the Makefile substitutes two files used for USB communication with prepared files that add some useful read and write functions.

- **GAMs** contains a folder for each GAM. Inside the GAM folder a Makefile should be created to compile the GAM separately creating a static library to be linked to the project.

- **DataSources** contains a folder for each DataSource. Also in this case it is recommended create a separated static library for each DataSource.
- **ErrorManagement** contains the definition of the *DebugErrorProcessFunction* function, namely the function that takes in input an error code and description and handles the debug interface. By default the logger messages are streamed out to the serial UART interface defined as *errorUartHandle* in the *Addons.hdd* file.
- **Scheduler** contains the GAM scheduler and the HRT custom implementation. These implementations are usually already developed within the MARTe2 BaseLib, but for embedded architectures it has been decided to keep them separated because each platform or project can implement its custom behavior depending on the necessities and requirements. By default the scheduler is the *BasicScheduler* that possibly supports only a single MARTe real time thread, but using FreeRTOS it is easy to implement also a GAM scheduler to support multiple real time threads in a MARTe application. Concerning the HRT implementation, by default the *Counter()* function returns the number of sys ticks multiplied by 1000 added to the TIM6 current counter value. This is strictly related to the configuration of TIM6, that has been setup with a counter max value of 999 and a period of 1 ms and it is calibrated each ms inside the *SysTick* interrupt handler (that is 1ms period too). It provides a counter value with a resolution of 1μs.
- **Configuration** contains all the configuration files for different MARTe applications.

The MARTe application entry point is defined within the *startup_main.c* file. Here the MARTe2 *StandardParser* object parses the configuration stream generating the *ConfigurationDatabase* and successively the *ObjectRegistryDatabase* containing all the GAMs and DataSources objects of the current application. It is possible to provide the configuration stream by a string directly within the code (remember that there is no file system), or obtaining it from a communication interface like USB or SPI. The second approach is preferred because there is no need to re-compile the code, and a device connected to the embedded board can launch different MARTe applications just by sending different configurations by stream interfaces.

4.2.3 Embedded tool-kit

This section describes briefly the MARTe2 tool-kit implemented for STM32F4-Discovery embedded platform. The final user can build its own application just by assembling the already implemented DataSources blocks within the provided tool-kit. The aim is to provide a more complete as possible set of MARTe blocks to drive the embedded boards peripherals, and to support a great number of embedded platforms. The

embedded world is necessarily strictly custom and each board has its particular peripherals, assembly code and accordingly even different compilation rules. The tool-kit has been organized such that each folder contains the code and the compilation rules and definitions (Makefile) for the selected platform. Currently the work has been implemented and tested for the STM32F4-Discovery board, but the ST drivers already provide an abstraction layer allowing the same code to work on different boards of the same ST family. The DataSources implemented for the STM32F4-Discovery board can be so employed also for other boards of the ST family perhaps with very few modifications. Follows a description of the implemented DataSources:

- **TimerDataSource** is commonly used as the synchronizing DataSource that defines the cycle time. The user must configure the desired *Frequency* parameter. The current implementation launches a FreeRTOS thread that waits on an event semaphore to be released periodically with the specified frequency.
- **USBCommunication** is the DataSource managing the USB interface. It can be configured to be blocking or not blocking for read and write operations. The *SynchronizedWrite* parameter if true permits a write only if something has been read in the same cycle. It can be used to synchronize the communication with a slower device that acts as the master.
- **UARTCommunication**, same for *USBCommunication* manages the UART serial interface.
- **GPIO** is the DataSource that allows to read or write from a GPIO register. Since each GPIO register consists in 16 pins, it is enough a short integer signal type to read the pin mask or to set the desired pin register output.
- **TIMBase** allows to read or write a timer counter. The user has to specify in the configuration the timer handle name.
- **PWM** is usually used as output DataSource to provide a PWM signal. The user must specify in the configuration the timer handle name, the channel number and the initial pulse value.
- **ADC_DMA** performs a single read from ADC for each signal using DMA. The user must specify the ADC handle name and the array of activated channels.
- **EncodersCounterDataSource** is usually used as input DataSource to read the counter of a timer configured in encoder mode. The user must specify in the configuration the timer handle name.

Other DataSources have been added in the *Externals* folder to manage the interface with external sensors or devices that have been employed during the development

of the various projects. For instance there is a DataSource to drive the *nRF24L01* radio module, one to read the the external encoder counter *HCTL-2022* and one to drive the *L298N* H-bridge. Follows an example of the *Data* section of a MARTe2 configuration showing how to configure some of the DataSources described above.

```
"    +Data = {"
"        DefaultDataSource = DDB1"
"        Class = ReferenceContainer"
"        +Timings = {"
"            Class = TimingDataSource"
"        }"
"        +Timer = {"
"            Class = TimerDataSource"
"        }"
"        +USB = {"
"            Class = USBCommunication"
"            ReadBlocking = 0"
"            WriteBlocking = 0"
"            SynchronisedWrite = 1"
"        }"
"        +PWM_Motor = {"
"            Class = PWM"
"            TimIdentifier = TIM4"
"            Channel = 0"
"            ZeroVal = 0"
"        }"
"        +PWM_Drive = {"
"            Class = PWM"
"            TimIdentifier = TIM4"
"            Channel = 1"
"            ZeroVal = 240"
"        }"
"        +ADC = {"
"            Class = ADC_DMA"
"            AdcIdentifier = ADC1"
"            Channels = {8, 9}"
"        }"
"        +Encoder = {"
"            Class = EncodersCounter_opt"
"            TimIdentifiers = {2}"
"        }"
```

4.3 Scortec Robotic Arm Controller v1

The robotic project described in this section derives from the thesis of the students Andrea Monti and Alessio Moretti that, under the supervision of Daniele Carnevale and Luca Boncagni, developed a MARTe-based control system on the Raspberry Pi board which communicates with the robot power unit (PU) using the STM board in the middle. The system architecture is divided in four macro blocks: an external PC (Raspberry Pi), 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.

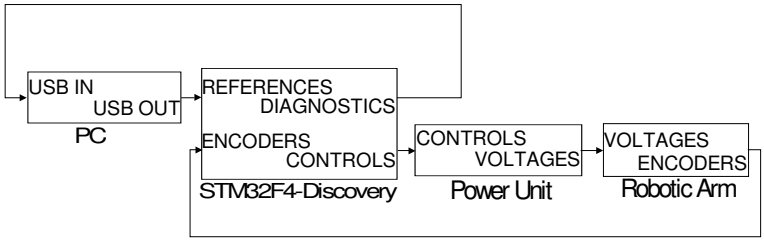

Figure 4.11: The system architecture is composed of four main components. The external PC is optional and allows to monitor and configure the embedded system.

4.3.1 Scortec Robotic Arm

The *Scortec-ER* is an anthropomorphic five degrees robotic arm manipulator. Five DC motor are directly connected through mechanical reducers to the arm rotative joints. 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. A sixth motor allows to open and close the gripper. On the crankshafts of the motors are mounted optical quadrature encoder sensors to provide the relative measurements (with respect the position where the encoder counters are zero) of the angular position of the joints. Each encoder sensor provides two channels, commonly called A and B, both providing square wave signals when the joint is moving and, being shifted by a quarter of period, allow also to examine the direction of the movement. The power cables of the DC motors (two for each motor: positive and negative poles) and the encoder counter cables (four for each joint: 5V, GND, A, B) are connected together to a 50-pin D male connector.

4.3.2 Power Unit

The 50-pin male connector of the Scortec is connected to the relative female connector on the power unit that must provide the power to the DC motors and counts the edges

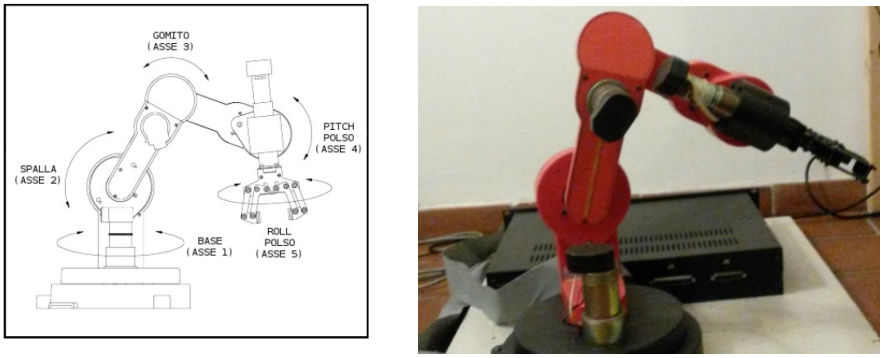

Figure 4.12: Manipolatore Scortec ER-I

of the encoder channel signals. On the other side, 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 connected to the internal PIC. Thus two ports of 8 digital pins are connected to the STM32F4-Discovery pins and they are used to read the encoder values. Each motor of the robotic arm draws about $1A$ at the maximum voltage amplitude of $12V$, thus the power unit need to supply at least $72W$ of electrical power in output. It needs, for each motor, an analog signal between +5 and -5 Volts to drive the power supply (figure 4.13 shows the PU input and output connector pinout). 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 $p_1 = p_2 = 666.67\ rad/s$. 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. 4.14 depicts the power unit block scheme, Fig. 4.15 shows the electronic circuit scheme of the filter and Fig. 4.16 depicts the output amplitude in the frequency domain by providing a signal with an amplitude of 10V in input. Fig. 4.17 shows the output in the time domain in three cases: 100%, 50% and 0% duty cycle with a 1 kHz PWM frequency. From Fig. 4.16 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.

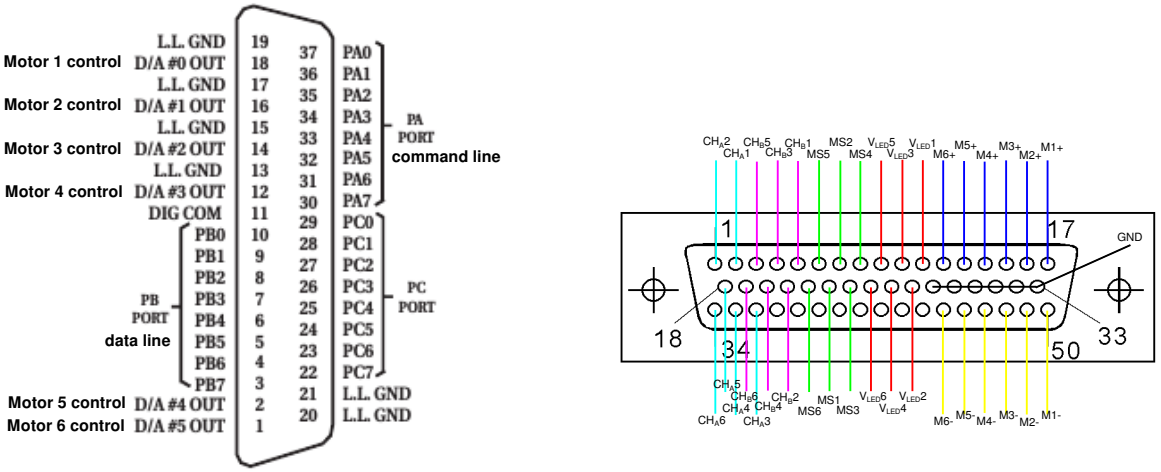

Figure 4.13: On the left, the PU input D-37-pin connector. The 8-bit parallel PA port is used as command line to communicate with the internal PIC, the PB port is used to read the encoder counters for each joints and there are 6 inputs for motor control signals. The single motor control signal must be analogic with a voltage $\in [-5, +5]\,V$. On the right the PU output D-50-pin connector. Mn+ is the positive pole of the motor n, Mn- the negative, CH_An and CH_Bn are the channels A and B of the encoder sensor on the joint n. MSn (micro switches) are the digital signals of the limit switch sensor of the joint n. These sensors are not present on the Scortec-ER, but are mounted on other models like the Scorbot robotic arms series.

4.3.3 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 board has been configured as USB communication device (slave).
- **UART2** using pins PD5=Tx, PD6=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** (General Purpose IO) pins to read the number of encoders of the motors from the power unit.

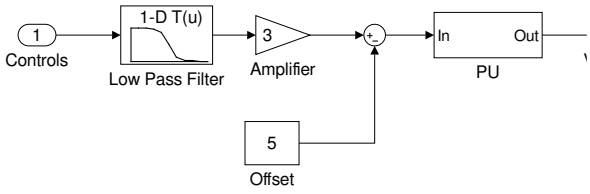

Figure 4.14: Power unit interface to the control output. A low-pass filter translate the PWM voltage to the required analog range.

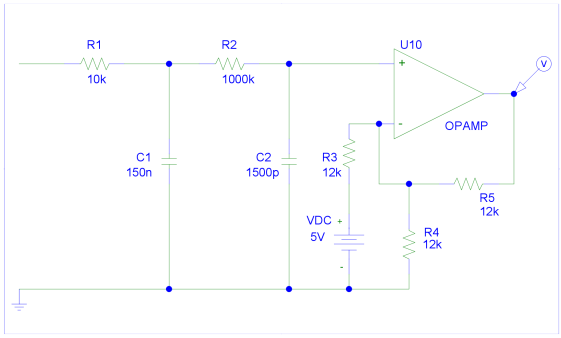

Figure 4.15: Low pass filter scheme (with amplification and offset) for the translation between the PWM and the power supply analog input.

- **5 PWM channels** (one for each motor). Since each STM32F4-Discovery timer can drive at most four pulse-width modulation (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 a high priority interrupt every 2.5 ms and post an event semaphore. This can be used as a 400 Hz synchronous control cycle mechanism.
- **Timer 5** to provide the high resolution timer. It triggers a high priority interrupt 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.
- **User-button** connected to the pin A0. The pressing of the button triggers a low priority interrupt and allows the application to react to the event.

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 sinusoidal waveforms with configurable frequency,

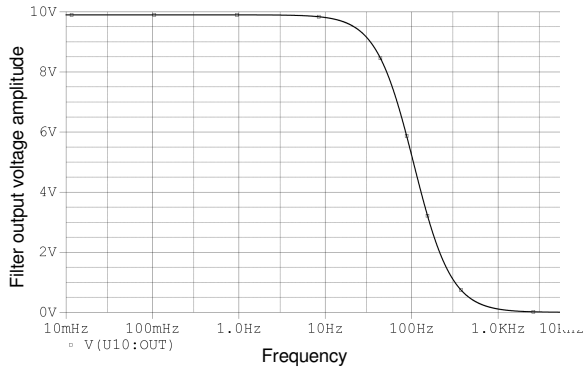

Figure 4.16: Frequency response of the filtering stage electronic circuit with a sinusoidal input of 10V amplitude.

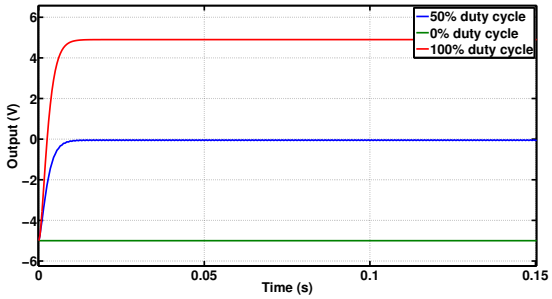

Figure 4.17: Output of the filter with 50% (blue), 0% (green), 100% (red) PWM duty cycles at 1 kHz frequency

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 single input single output (SISO) or multiple inputs multiple outputs (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 linear time invariant (LTI) plant model with single input and single output. The plant model can be parametrized by defining the state space properties of the model. 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 *ScortecControl-*

GAM) 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. 4.18.

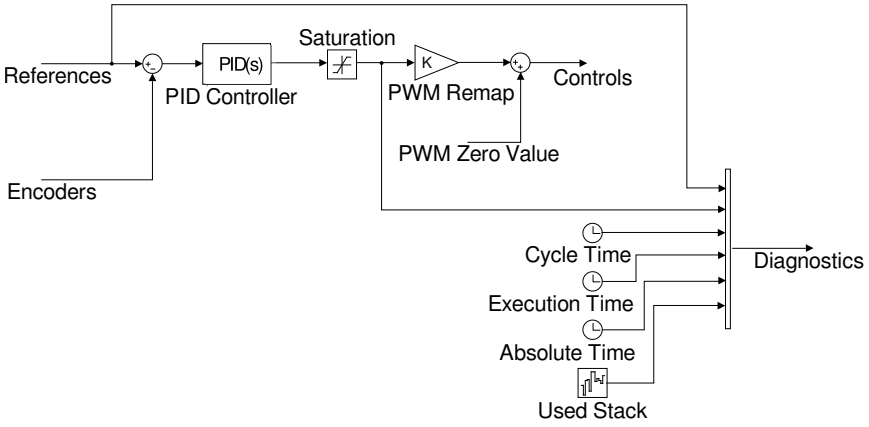

Figure 4.18: STM32F4-Discovery block functional behavior. The controller is driven by a reference set by the user and a measurement given by the encoders. The outputs are the internal variables values (for debug) and the control voltage.

As depicted in Fig. 4.19 and Fig. 4.20, 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.

The second application (Application-2) has the purpose of performing a standalone 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. 4.20 can simulate a LTI closed-loop system.

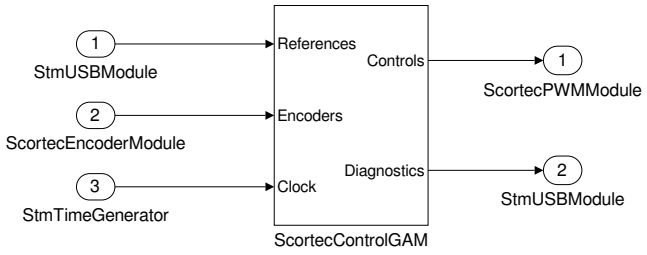

Figure 4.19: Execution sequence of the MARTe2 application that is physically connected to the robot (Application-1).

4.3.4 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. 4.21, while Fig. 4.22 shows 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

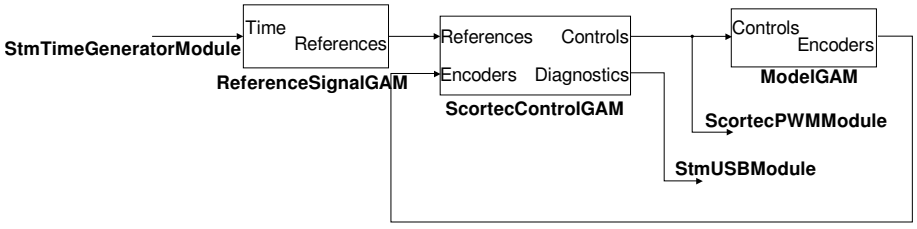

Figure 4.20: Execution sequence of the MARTe2 application that is used to simulate the connection to the robot (Application-2).

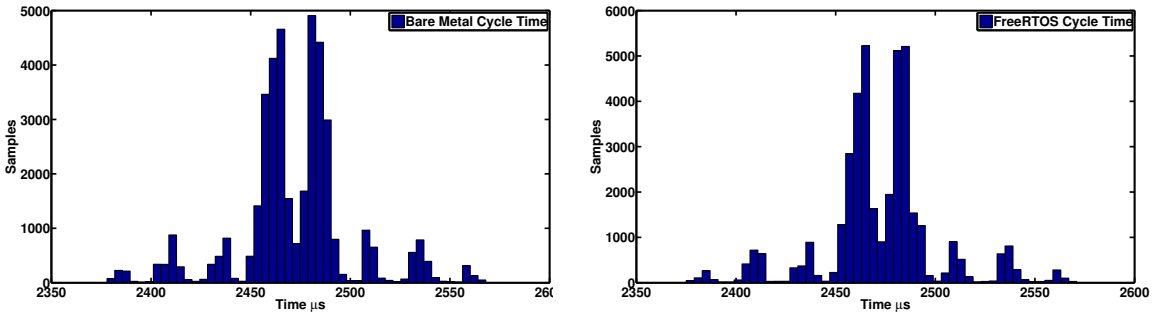

Figure 4.21: On the right *Bare-metal* application cycle time computed over 40000 samples. On the left the FreeRTOS application cycle time computed over 40000 samples. The outliers are possibly driven by the USB communication but do not greatly affect the cycle time, i.e. the application is capable of restarting a new control cycle with-in an acceptable jitter.

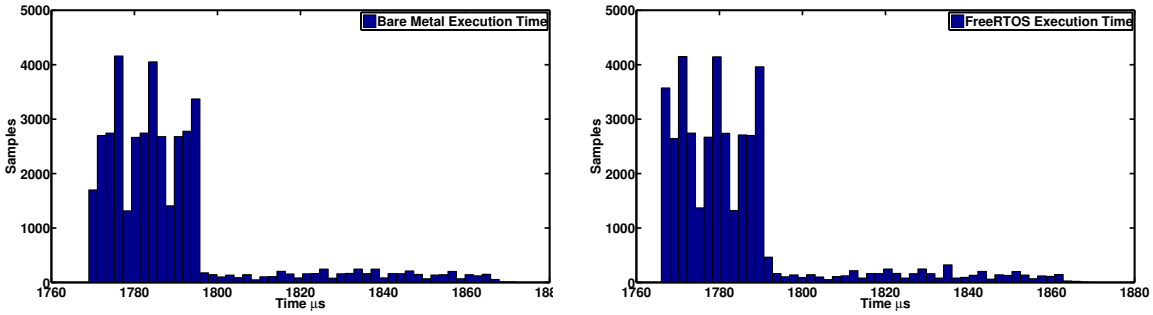

Figure 4.22: On the right the*Bare-Metal* application execution time computed over 40000 samples. On the left the FreeRTOS application execution time computed over 40000 samples. As in the *bare-metal* case the outliers are driven by a jitter in the USB interface.

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-2 a second thread which executes concurrently (but with lower-priority) to the GAM scheduler thread. Using this thread and the MARTe1 HTTP server running on the PC, 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. 4.23 shows 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

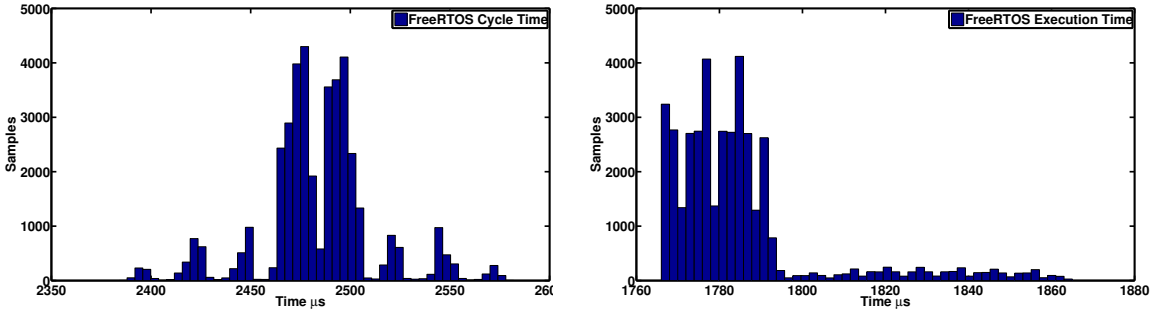

Figure 4.23: On the rith the FreeRTOS two-threading application cycle time computed over 40000 samples, where it can be seen that the system cycle-time is not affected by a second monitoring thread. On the left the FreeRTOS two-threading application execution time computed over 40000 samples.

4.3.5 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.

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. Some of other robot arm models provides switch sensors on all joints in order to define the *home* position, thus a common practice is to define the *home* position as the position in which all the joints stand at their end stops. Unfortunately the *Scortec* robot does not provide hardware switch sensors, so we have decided to implement a software procedure to check if each motor has reached its end stop which will be considered as the joint *home* position. The adopted solution consists in assuming that the joint has reached its end stop if the input voltage (control input) is different from zero (or, more realistically, inside a configurable range around zero) and the read encoder value remains for more than a configurable number of application cycles. Once these two conditions have been

checked, the program assumes that the motor cannot move the joint further in the same direction. The only way to exit from this dead zone is moving the joint on the other direction by providing a voltage to the motor with the opposite sign of the previous one.
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.

During State-1 the *ScortecControlGAM* assumes that:

$$r_i = y_i + \sigma_r$$

where r is the reference motor position, y the encoder steps and σ_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 reached their end stops, where the control and encoder steps will be zeroed. 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 motor shown in Fig. 4.24 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. In this way we have obtained an identification of the motor on the base joint of the *Scortec* robot arm. Defining:

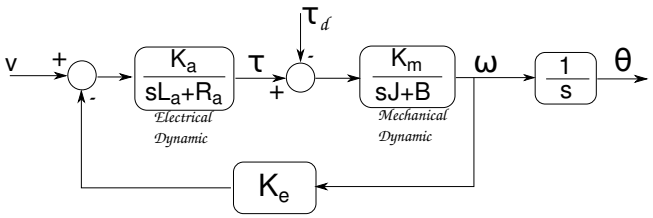

Figure 4.24: The DC motor model implemented in the *ModelGAM*

.

- 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 counterelectromotive force constant.
- K_a the motor torque constant.

- K_m the motor mechanical constant.

the transfer function between the input voltage v and the joint angular speed ω (the derivative of the encoder position θ) 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 ones:

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

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 has been saturated between [-5 V, +5 V]. As well known, the saturation usually introduces oscillations up to leading to instability. Then, we have implemented a switched PID controller which can change the value of its gains depending on the value of the tracking error. As discussed in [9] for a general class of systems, this allows to improve the tracking performances especially in the presence of saturation nonlinearities. 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 overshoot.

Fig. 4.25 shows the results of the hardware-in-the-loop (i.e. executed in the STM32F4) simulations in which a simple proportional controller with $K_p = 25$ is compared to a switched proportional controller that initially has $K_p = 25$ before and after the switch $K_p = 8$. The switch occurs when the module of the error became less

than 200 encoder steps. Fig. 4.25 on the left compares the outputs of the two controllers and it can be appreciated the higher effectiveness of the switched proportional controller. Fig. 4.25 on the right shows the control signal.

This approach can be considered as a simplified version of the controller described in [13] 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 switched proportional controller, is shown in Fig. 4.26, where the value of σ_r has been set to 100 encoder steps and the end stop to 3000 encoder steps. Note that, since the error value in this phase is constant and equal to $e = \sigma_r = 100$ encoder steps, which is less than the imposed switched threshold ($\sigma(e) = 200$ encoder steps), the gain remains $K_p = 8$, thus, before the joint reaches its end stop, the control action is always equal to $K_p \cdot e = 800$ mV. 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 joint has reached the end stop (the joint movement is blocked) and the input voltage is set to zero.

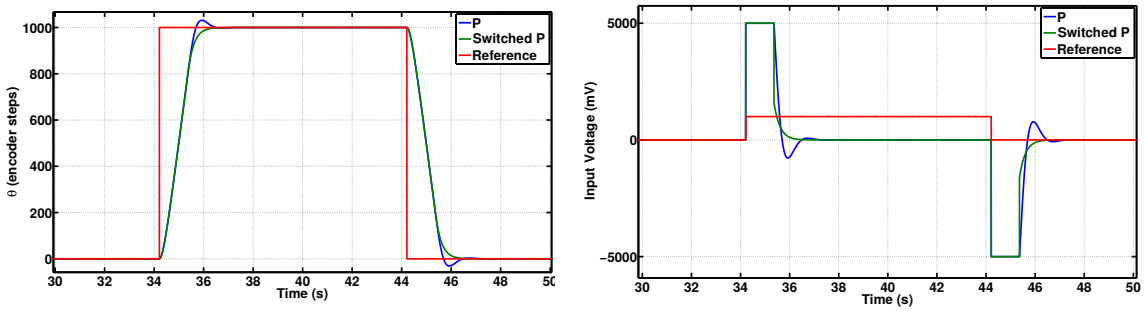

Figure 4.25: On the right the closed-loop system output with fixed proportional controller (blue), and with switched proportional controller (green). The y-axis joint angular position is in encoder units. On the left the input voltages to the motors with fixed proportional controller (blue), and with switched proportional controller (green).

4.3.6 Conclusions

In this section 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

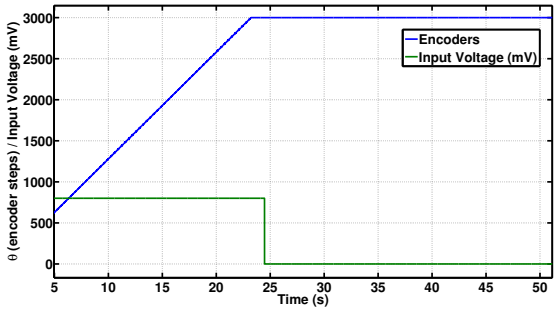

Figure 4.26: State-1 Home position *Scortec* routine for one of the joints. In blue the number of encoders, in green the input voltage. The joint home position (on the end stop) has been set to 3000 encoders. The y-axis joint angular position is in encoder units and the input voltage in mV.

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.

4.4 Scortec (Scorbot) Robotic Arm Controller v2

The limits of the architecture described in section 4.3 are the communication protocol with the power unit (that requires some delays), and the implemented DAC filters (to transform the PWM control signals provided by STM board to the analogical control signals required by the PU). Moreover, it is recommendable to work with a well known system architecture that can be repaired easily in case of damages. Thus, it has been decided to develop our own 'power unit' and to remove the standard one to control Scortec and Scorbot manipulators.

4.4.1 Hardware

Follows a list of the components needed to implement the new PU:

- **STM32F4-Discovery**: it is possible configure some internal timers to encoder counter mode. Each timer owns two pins to receive the A and B encoder sensor's channels and counts the edges without bothering the CPU with any interrupt. In this project we have selected three timers in encoder mode: TIM2 (A15, B3) for motor 1 encoders, TIM3 (A6, A7) for motor 2 and TIM5 (A0, A1) for motor 3.
  The STM board can also read directly the micro-switches (MS) digital signals.

The micro-switches are little buttons located in proximity of the limit end of the robot joints. When the digital signal is high, the joint has reached its limit end. The Scortec robot does not mount micro-switches sensors, but they are mounted on other models like the Scorbot, so 6 pins of the STM board have been configured as digital inputs (E9, E10, E11, E12, E13, E15) to read MS signals.
The board must also provide the control signals for each DC motor of the robotic arm. Each internal timer consists in up to four channels that can be configured to supply four different PWM signals with the same frequency. Four channels of the timer TIM4 have been selected to provide PWM signals for four motors (D12, D13, D14, D15) together with two channels of TIM9 (E5, E6). These PWM signals go in input to a H-bridge module which needs also two further digital inputs to select the direction of the motor. These pins are: (B10, B11) for motor 1, (D8, D9) for motor 2, (D10, D11) for motor 3, (E0, E1) for motor 4, (E2, E3) for motor 5, (A8, A9) for motor 6.
Six ADC inputs have been provided to obtain a measurement of the current flowing in each motor. These sensors can be used for a future control in torque. Six channels of the ADC1 have been enabled on pins (A2, A3, A4, A5, B0, B1).
The three remaining encoder counters are external to the STM board but have to communicate with it. Three external HCTL2022 chips 32-bit encoder counters have been employed, but are used as 16-bit counters grounding the SEL2 pin. The OE pin, to be set low when we need to read the counter register, is connected to D1 for motor 4, to D3 for motor 5, to D4 for motor 6. The SEL1 pin which has to be set low to read the most significant byte (MSB) and high to read the less significant byte (LSB) is connected to D0 for motor4, to D2 for motor 5, to D7 for motor 6. The reset pint (RST) used to reset the counter register to zero is connected to E7 for motor 4, to E8 for motor 5, to E4 for motor 6. The clock is common for the three chips and it is generated from a PWM output with 50% duty cycle from the channel 4 of TIM1 (E14 pin) with 1 MHz frequency. To read the single byte 8 parallel digital inputs have to be configured for each external encoder counter chip. These pins are (C0-C7) for motor 4, (C8-C15) for motor 5, (B2, B4, B5, B6, B7, B8, B9, B12) for motor 6. The configuration provides a USB-FS (full speed, up to 12 MHz) communication device, a SPI and a UART serial interfaces. Figure (4.27) shows the STM32F4-Discovery configuration

- **HCTL-2022**: three of these encoder counter chips have been employed to count the edges of the A and B channels signals emitted from optical quadrature encoder sensors located behind the motor 4 (pitch movement), the motor 5 (roll movement) and the motor 6 (gripper) respectively. They are 32-bit counters but, for this application, 16-bit counters are enough so they have been used as 16-bit counters (also because the internal STM encoder counters for the motors 1,2 and 3 are 16-bit). The OE pin has to be set low to read the internal chip

counter in a stable mode. Two SEL pins, SEL1 and SEL2 can select up to four bytes to read. (LOW, LOW), (LOW, HIGH), (HIGH, LOW) and (HIGH, HIGH) SEL1 and SEL2 couples select from the LSB (less significant byte) to MSB (most significant byte) of a 32-bit counter. For this application SEL2 pin has been grounded (always LOW) to read the two less significant byte of the internal register implementing a 16-bit counter. The pins D0-D7 provide the byte selected by SEL1 when OE is low (D0 is the less significant bit D7 most significant bit). The pin RST has to be set low to reset the internal counter. The CLK pin receives in input a square wave clock signal with 1 MHz frequency from STM micro-controller.

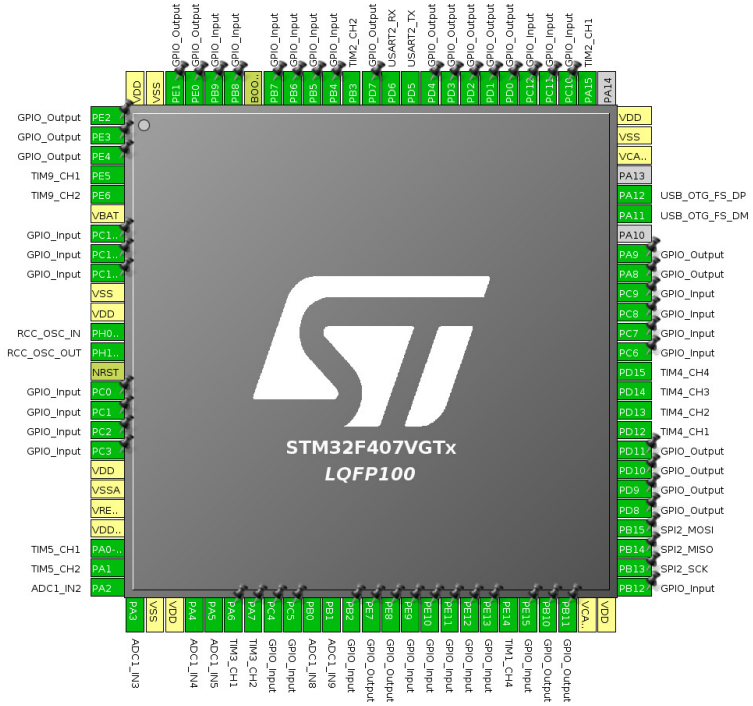

Figure 4.27: STM32F4-Discovery CPU (LQFP100) configuration for robotic arm controller application

- **L298N H-bridge**: three of these chips have been used to drive the robotic arm DC motors. Each one of these chips contains two separated H-bridges and can be so employed to control two different DC motors on one step motor. They can drive motors with a DC current up to 2A per channel and 3A of non repetitive peaks. The motors mounted on the Scortec/Scorbot do not absorb more than 1A, except when the joint reaches its limit. In this case the current consumption can raise up to 3A, thus it is important to decrease the power request in order to avoid hardware damages. The pins OUT1 and OUT2 provide the power to the

first DC motor and must be connected to its negative and positive poles. Same is for OUT3 and OUT4 for the second motor. The pins IN1 and IN2 can be used to select the direction of the first motor electric power and accordingly its angular movement versus; depending on how the OUT1 and OUT2 are connected to the motor poles, different digital levels of IN1 and IN2 (i.e (HIGH, LOW) or (LOW, HIGH)) will allow a clockwise or counterclockwise motor movement while if they are at the same digital level the motor cannot move. The PIN ENA can receive a PWM signal that modulates the amount of power to be provided to the first motor. Concerning the second motor, its direction can be selected by the IN3 and IN4 pins and its voltage by the ENB pin. The supply voltage is shared by the two internal H-bridges and can be provided on the V_S pin. The V_{SS} is the digital HIGH level voltage reference can be connected to 5V in this case. Two pins SENSA and SENSB are in series to the bottom sides of the two H-bridge and can be directly grounded. Since the absolute value of the consumed current of each motor pass trough these two pins (SENSA for motor 1 and SENSB for motor 2), it is possible to add a resistor in series and measure its voltage to have a feedback of the current in output. In this case the implemented circuit allows the connection of these pins to ADCs and to insert a resistor in series in order to measure the current consumption of each motor that can be used in feedback for torque control. The electric scheme of the L298N is shown in figure (4.28).

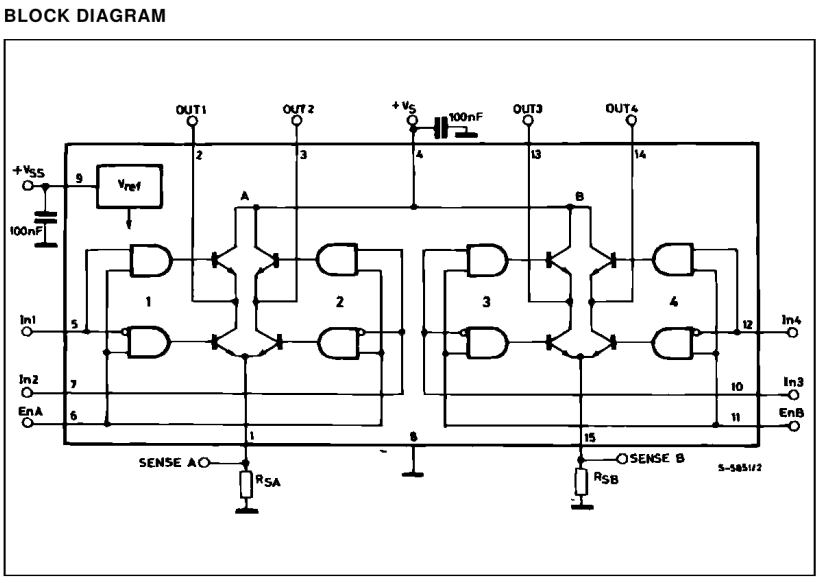

Figure 4.28: L298N H-bridge scheme

A first prototype has been developed on a strip-board and tested successfully to control a Scorbot robotic arm. The project moved to the implementation of a printed circuit board (PCB) to obtain a feasible object capable of controlling up to six motors.

Although the hardware architecture has been designed for the specific purpose of controlling a 5-degrees robotic arm, depending on how STM board is programmed (and this is very easy thanks to the MARTe2 framework) it can be employed for a very large amount of projects and applications. The design of the PCB has been carried forward trying to meet a plug and play approach. The work done so far contemplates three separated boards with printed circuits on top and bottom sides:

- The first PCB consists in two 2×25 pin-headers where the STM32F4-Discovery must be plugged in and the sub D-50 female connector to be connected to related male one of the robotic arm. In the board are the connections of the micro switches and of the first three encoders from the D-50 connector to the STM pins. Each the encoder sensor led is supplied by 5V with a 100Ω resistor in series. The other routes are connected to pin-headers to be plugged on the second PCB. Figure (4.29) shows the schematic.

- On the second PCB are implemented the connections of the three external encoder counters for motor 4, 5 and 6. Each HCTL-2022 must be connected to the CHA and CHB encoder sensor channels and to the STM that reads the counter. Two 10 kΩ pull-up resistors are inserted to read the encoder channels. The schematic is shown in figure (4.30).

- The third PCB implements the connections to the externals H-bridge boards and the motors supply. The H-bridges chips are not welded on the board but have to be connected through pin-headers. This allows to use different types of H-bridges or to easily change the components in case of damages. A removable 3A fuse is inserted on each motor power supply path to prevent to burn the motors in case of over-currents. The schematic is shown in figure (4.31).

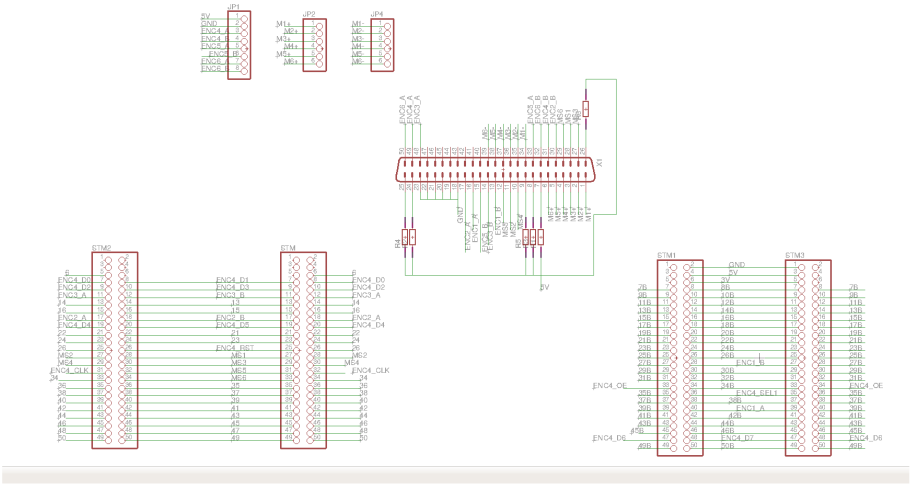

Figure 4.29: First PCB schematic.

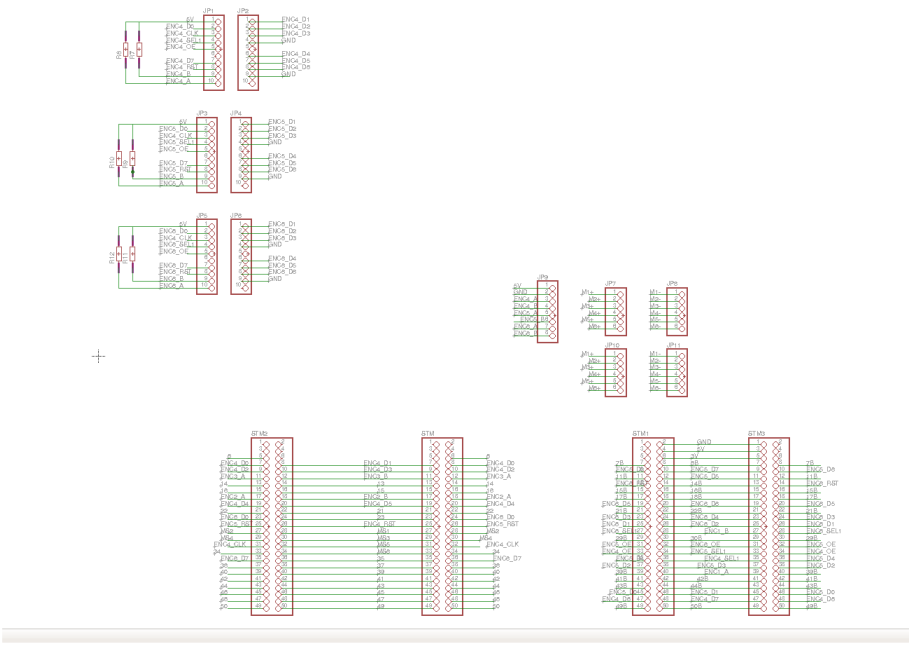

Figure 4.30: Second PCB schematic.

Follows the configuration file of the *Scorbot* controller application.

```
"$Application1 = {"
"    Class = RealTimeApplication"
"    +Functions = {"
"        Class = ReferenceContainer"
"        +GAM1 = {"
"            Class = RobotArmGAM"
"            NumberOfMotors = 5"
"            Kp={80, 80, 80, 80, 80}"
"            Ki={0, 0, 0, 0, 0}"
"            Kd={0, 0, 0, 0, 0}"
"            SwitchPinMask=0x3e00"
"            MaxPwm={999, 999, 999, 999, 999}"
"            MinPwm={0, 0, 0, 0, 0}"
"            MinControl={0, 0, 0, 0, 0}"
"            MaxControl={12000, 12000, 12000, 12000, 12000}"
"            EndSwitchBound={10, 10, 10, 40, 40}"
"            MotorDirection = {1, 1, -1, 1, -1}"
"            CounterDirection = {-1, -1, 1, -1, 1}"
"            InputSignals = {"
"               TimerSignal = {"
"                    DataSource = Timer"
```

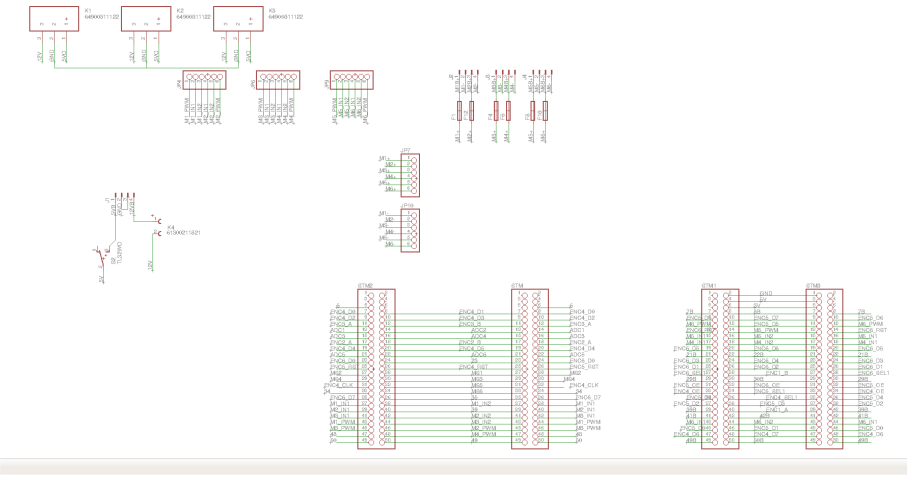

Figure 4.31: Third PCB schematic.

```
"                    Frequency = 50.0"
"                    Type = uint32"
"                }"
"                FromUSB = {"
"                    DataSource = USB"
"                    Type = int32"
"                    NumberOfDimensions = 1"
"                    NumberOfElements = 2"
"                }"
"                FromEncoder1 = {"
"                    DataSource = Encoder1"
"                    NumberOfDimensions = 1"
"                    NumberOfElements = 3"
"                    Type = uint32"
"                }"
"                FromEncoder2 = {"
"                    DataSource = Encoder2"
"                    Type = uint32"
"                }"
"                FromEncoder3 = {"
"                    DataSource = Encoder3"
"                    Type = uint32"
"                }"
"                FromMS = {"
"                    DataSource = MS"
"                    Type = uint32"
```

```
"                }"
"            }"
"            OutputSignals = {"
"                ToPWM = {"
"                    DataSource = PWM"
"                    Type = int32"
"                    NumberOfDimensions = 1"
"                    NumberOfElements = 5"
"                }"
"                Diagnostics = {"
"                    DataSource = USB"
"                    Type = int32"
"                    NumberOfDimensions = 1"
"                    NumberOfElements = 18"
"                }"
"            }"
"        }"
"    }"
"    +States = {"
"        Class = ReferenceContainer"
"        +State1 = {"
"            Class = RealTimeState"
"            +Threads = {"
"                Class = ReferenceContainer"
"                +Thread1 = {"
"                    Class = RealTimeThread"
"                    Functions = { GAM1 }"
"                }"
"            }"
"        }"
"    }"
"    +Data = {"
"        DefaultDataSource = DDB1"
"        Class = ReferenceContainer"
"        +Timings = {"
"            Class = TimingDataSource"
"        }"
"        +Timer = {"
"            Class = TimerDataSource"
"        }"
"        +USB = {"
```

```
"            Class = USBCommunication"
"            ReadBlocking = 0"
"            WriteBlocking = 0"
"        }"
"        +PWM = {"
"            Class = L298N"
"            TimIdentifiers = {4, 4, 4, 4, 9}"
"            Channels = {0, 1, 2, 3, 0}"
"            DirPorts = {1, 1, 3, 3, 3}"
"            DirPins = {10, 11, 8, 9, 10}"
"            MaxPwm = {999, 999, 999, 999, 999}"
"        }"
"        +Encoder1 = {"
"            Class = EncodersCounter_opt"
"            TimIdentifiers = {2, 3, 5}"
"        }"
"        +Encoder2 = {"
"            Class = HCTL2022"
"            OE_Port = GPIOD"
"            OE_Pin = 1"
"            SEL_Port = GPIOD"
"            SEL_Pin = 0"
"            RESET_Port = GPIOE"
"            RESET_Pin = 7"
"            CLK_Id = TIM1"
"            CLK_Channel = 3"
"            ByteReg_Port = GPIOC"
"            ByteReg_Mask = 0xff"
"        }"
"        +Encoder3 = {"
"            Class = HCTL2022"
"            OE_Port = GPIOD"
"            OE_Pin = 3"
"            SEL_Port = GPIOD"
"            SEL_Pin = 2"
"            RESET_Port = GPIOE"
"            RESET_Pin = 8"
"            CLK_Id = TIM1"
"            CLK_Channel = 3"
"            ByteReg_Port = GPIOC"
"            ByteReg_Mask = 0xff00"
```

```
"        }"
"        +MS = {"
"            Class = GPIO"
"            GpioIdentifier = GPIOE"
"        }"
"    }"
"    +Scheduler = {"
"        TimingDataSource = Timings"
"        Class = BasicScheduler"
"    }"
"}"
```

The *RobotArmGAM* receives two control values from USB that are used to change the application state. If the first received value is zero, depending on the second received value, the *RobotArmGAM* performs different tasks:

- val=(0,0): the robot is controlled manually by receiving the two USB control values: $val = (i, k)$ with $i > 0$ and $k > 0$ applies to the motor i the maximum positive allowed voltage, with $k < 0$ the opposite voltage is applied.
- val=(0,1): the robot home procedure starts moving the robot to the initial standard position.
- val=(0,2): the robot is controlled by sending the desired joint position in encoder steps. $val = (i, k)$ moves the joint i at k encoder steps from the home position. A PID controller can be configured within the configuration file.

4.5 CoolCar

Another project where MARTe2 has been used on the STM32F4-Discovery embedded board is the *CoolCar* mobile robot. Starting from the chassis of a radio-commanded car, the STM32F4-Discovery has been employed to drive the DC motor responsible of the car speed and the servo motor that manages the car direction. A Raspberry Pi 2 has been engaged as the high level controller and connected to a USB camera mounted in front of the car. A MARTe application running on Raspberry Pi communicates through the USB interface to the MARTe2 application on the STM32F4-Discovery micro-controller sending the control values to be provided to the motors and receiving diagnostic data. This data includes the cycle time registered by the embedded board and the value of the counter of a quadratic optical encoder sensor mounted on the DC motor. Two different applications have been developed. Figure (4.33) shows the STM32F4-Discovery board configuration developed using *STM Cube* tool. Two channels of TIM4 have been enabled in PWM mode to drive respectively the DC motor for speed control and the servo motor for direction control.

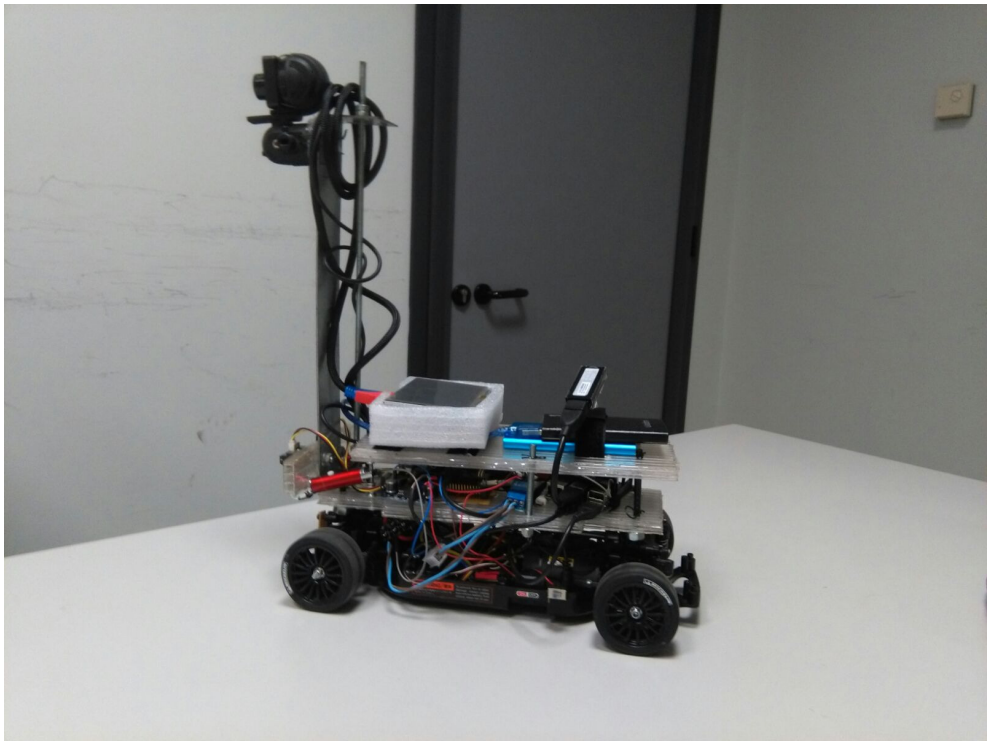

Figure 4.32: The CoolCar

4.5.1 Remote Control

The first developed application is the remote control of the car. The MARTe application on the Raspberry located on the car acts as a client, streaming the frames captured by the USB camera to a server. The server receives the camera stream and the user can send back to the Raspberry the desired value of the voltage to provide to the DC motor and the desired car direction to be mapped to the PWM control signal to provide to the servo motor. This allows the user to drive the car remotely having the visual feedback of the camera mounted in front of the car. From the MARTe configuration, the user can choose the connection mode between UDP and TCP and the camera view mode. For instance, setting *CamMode=1*, the captured frames are converted in gray-scale images before sending, reducing the payload size and accordingly the connection delay. There are theoretically no distance limits but, obviously, the real-time performance depends by the distance and by the quality of the connection links.
Concerning the server, two options have been implemented. The first one is a standalone server application written in C++ that shows the frames received continuously by the car and using the *ncurses* library allows the user the remote drive using the direction arrows of the keyboard. The second one is a MARTe based application that exploits the MARTe HTTP interface to provide the user interface to drive remotely

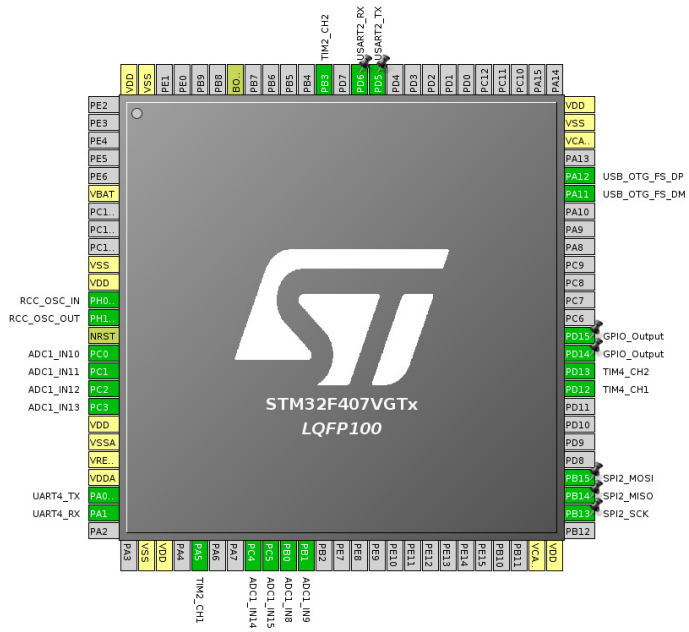

Figure 4.33: STM32F4-Discovery configuration for the CoolCar project

the car. Using a web browser, the user can visualize the camera stream received by the car and, using also in this case the directional arrows of the keyboard, change the control values to be sent to the car. In this case a javascript library called *p5.js* has been employed to realize an animated 3D model of the car and to capture keyboard input. Follows an example of the configuration of the implemented MARTe *RemoteDriverClient* block

```
+DriverClient = {
    Class = RemoteDriverClient
    PositionIndex = 20
    DirectionIndex = 16
    PositionFactor = -0.00637
    DirectionFactor = 0.2639
    DirectionMinIn = 140
    DirectionMinOut = -26.39
    ChassisLength = 26
    InitialXposition = 0.0
    InitialYposition = 0.0
    InitialOrientation = 0.0
    ServerIPaddress = "160.80.97.45"
    Port = 4444
```

```
    MinDriveControl = -5000.0
    MaxDriveControl = 5000.0
    MinDrivePwm = 140
    MaxDrivePwm = 340
    MinSpeedControl = -7200.0
    MaxSpeedControl = 7200.0
    MinSpeedPwm = 200
    MaxSpeedPwm = 400
    ZeroSpeedControl = 0.0
    ZeroDriveControl = 0.0
    FromRow = -1
    ToRow = -1
    FromColumn = -1
    ToColumn = -1
    PacketSize = 16384
}
```

4.5.2 Line Follower

The second application makes CoolCar a line follower robot. The car is capable to follow a path marked on the floor and to execute pre-configured actions depending on the color of the signals located along the path. Assume that the path to follow is defined by two tracks, one on the left and one on the right, and that the car should follow it, remaining between these two tracks. Follows the procedure of the line follower algorithm implemented within the MARTe application on the Raspberry Pi.

- A RGB frame is captured from the USB camera. A rectangular section of the frame is extrapolated with a width equal to the frame width and few pixels of height.
- The section is converted to grey-scale and a threshold filter is applied. All the pixels above the configurable threshold become white (255), the others become black (0).
- If there is enough contrast between the tracks and the floor, the resulting image should distinguish the position of the tracks. These positions go in feedback to a proportional controller that provides a control to the servo motor that handles the car direction. The control value is proportional to the difference between the desired and the current detected track position.
- The car can follow alternatively one of the two tracks and by default the right one is followed. Colored signals can be used to change the track to be followed allowing the car to turn left or right in case of crossroads. In general, the car

can be configured to implement the desired action depending on the detected color. In the developed application, for instance, the orange signal tells the car to follow the left track, the green signal tells the car to follow the right track.

- A state machine allows the car to recover, in some cases, the lost path. Figure (4.34) shows the states transition.

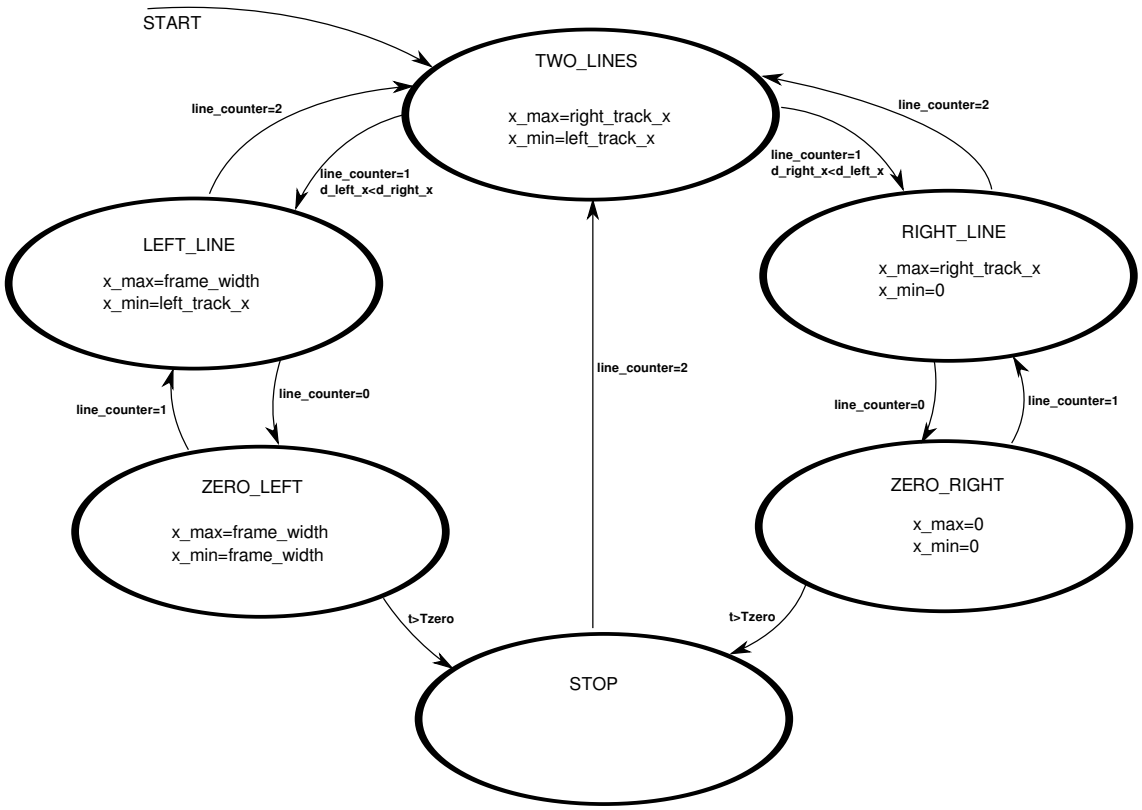

Figure 4.34: Line Follower application state machine. The car can follow the right or the left track alternatively, thus one x_max (when following right track) or x_min (when following left track) goes in feedback to a proportional controller. The state machine changes the value of x_max and x_min allowing the car to recover in some cases the path when one or both tracks have been lost.

- The computed direction control value is sent from the Raspberry Pi to the STM32F4-Discovery together with the control value of the DC motor responsible of the car speed. In the developed application the car speed in maintained constant and can be manually changed by the user from the HTTP interface, but nothing prevents to add new colors recognition that tells to accelerate, decelerate or even stop the car.

Follows the MARTe configuration on Raspberry Pi of the *LineFollower* block.

```
+LineFollower = {
    Class = LineFollower
    ThresType = 1
    LineThres = 15
    LineBandXFactor = 0.0
    LineBandYFactor = 0.5
    LineBandHeightFactor = 0.1
    MinDriveControl = -5000.0
    MaxDriveControl = 5000.0
    MinDrivePwm = 140
    MaxDrivePwm = 340
    MinSpeedControl = -7200.0
    MaxSpeedControl = 7200.0
    MinSpeedPwm = 200
    MaxSpeedPwm = 400
    ZeroSpeedControl = 0.0
    ZeroDriveControl = 0.0
    StandardSpeedControlPwmDelta = 12
    ZeroLineCycles = 60
    SignalMinWidth = 20
    CurvatureThres = 0.3
    BufferIndex = 0
    +Controller = {
        Class = PIDController
        Kp = 30.0
        Ki = 0.0
        Kd = 0.0
        IntegralCI = 0.0
    }
    +TurnLeftAction = {
        Class = TurnLeftSign
        RGBmin = {1 103 209}
        RGBmax = {38 190 255}
        Color = "Orange"
    }
    +TurnRightAction = {
        Class = TurnRightSign
        RGBmin = {71 150 99}
        RGBmax = {98 250 235}
        Color = "Green"
    }
}
```

Chapter 5

Conclusions and Recommendations for Further Work

In chapter 1 numerical and analytical approaches to find the optimal or sub-optimal switching time instants across a set of switching controllers have been discussed. The results show that a well-designed switching strategy can improve significantly the tracking performance of a closed-loop controlled system by employing a quadratic cost functional that weights the tracking error state. The limits of this approach is that the system is assumed to be perfectly known. Future works along this path can address the study of the robustness of such types of controllers by analyzing the sensitivity of the results with respect the uncertainties of the plant parameters. Moreover, adaptive strategies can be considered to identify the plant model on-line in parallel with the switching policy execution.
The study of the robustness of these type of switching strategies can be very useful for plasma control in tokamaks, where usually the model of the plant is absent or very approximated. Some robust switching and hybrid control approaches have been discussed in chapters 2 and 3 for FTU and TCV tokamaks and also in this case the results are encouraging. Adaptive control strategies like the approach described in section 2.5.1 can be considered, implemented and tested soon. The adaptive control can be a valid alternative to PIDs in this area, where to identify a valid model is so difficult.
The FTU data acquisition and control system has been developed using the MARTe framework. This framework is largely used by many working tokamaks in the world, first of all the JET that is the bigger one. During this Ph.D, I met the MARTe1 framework as an end-user, and I was one of the main contributors to the development of the new MARTe2 framework, where I was also responsible for the porting and integration to the STM32 platform. In chapter 4 I have described the quality process that makes MARTe2 compliant with specific certifiable rules. The remarkable work on the testing and the documentation of the whole framework aims at enlarging the application area of MARTe. Indeed, one of the most important new feature is the possibility of deploying MARTe2 on very limited embedded platforms providing to

the end user a high level program interface, block scheme based, to develop easily new applications. Some robotic projects have been described in this thesis, where the STM32F4-Discovery micro-controller have been selected as the first embedded platform that supports MARTe2. New GAMs and new embedded platforms will be supported in the future by adding new implementations in the tool-kit that will be available to the final user.
The MARTe2 framework is already being used in some ITER relevant projects, in particular in the gyrotron test facility that is being developed in EPFL. The more embedded version of the framework is also expected to be used in the data acquisition system of the magnetic diagnostic. In this case a Zynq SoC will be used to interface the magnetic sensors to the ITER interlock and control systems. I am currently working in the development and integration of the framework in this device.

List of Figures

1.1 The considered system block scheme. It is possible to switch a pre-defined number of time instants N across two pre-defined controllers which ensure asymptotic stability of the closed-loop system. 6
1.2 Tracking performances using the controller C_1 (cyan) and C_2 (magenta) without switches, with a single (green) and double (blue) switch. . . 11
1.3 The cost functional (1.3.1) in case of two switching times with a resolution of $10^{-2}s$. 12
1.4 (Top) Tracking performances: (red dashed) reference signal $r(t)$, the C_1 (cyan) and C_2 (magenta) controllers (no switch). Single-switch (blue), two-switch (green). (Bottom) Cost functionals J: only the switching time optimization (black), adding the controller reset state (blue) together with the feedforward (green) signal. The time resolution is $10^{-5}s$. 15
1.5 On top, a comparison of the system outputs obtained with and without bumpless constraint. In red is depicted the reference signal to be tracked, the light blue and purple trajectories are the output achieved employing separately C_1 and C_2 controllers respectively. The green line is the output achieved with a single-switch controller from C_1 to C_2 without bumpless constraint, the black one is the output achieved by the single-switch controllers with bumpless constraint. On bottom, the cost functionals for the two different cases: the green line obtained without bumpless constraint and the black one obtained introducing and optimizing with bumpless constraint $(e(t_1^+) - e(t_1) = 0$. The time resolution is $10^{-5}s$. 17

1.6 Comparison of the system outputs obtained in three different cases of on-line optimization. In red is depicted the reference signal to be tracked, the light blue and purple trajectories are the output achieved employing separately C_1 and C_2 controllers respectively. The green line is the output achieved with a single-switch controller optimizing only the switching time, the green one has been achieved optimizing also the controller reset state and the black one has been achieved optimizing also the initial condition of the feedforward signal $w(t) = x_w(t^\star)e^{-100(t-t^\star)}$. The time resolution is $10^{-4}s$ 21

2.1 FTU coils layout. The central T coil is used for plasma current control, the F and V coils supply the plasma radial confinement, the coil H supplies the vertical plasma confinement. 23
2.2 On the left is shown a GAM interacting with an hardware device across the shared DDB memory. The DDB is the only available way for GAMs to interchange signals. On the right is shown a group of seven GAMs scheduled within a RTT. 25
2.3 Poloidal section of FTU chamber in Cartesian coordinates. In blue the position of the meshes used to perform the plasma shape reconstruction. 27
2.4 The FTU plasma current control loop. 28
2.5 Comparison between the shots 35975 and 35976 without the double integrator and the shot 38786 when the double integrator has been activated from 0.25s 30
2.6 On the left column two shots: 41815 and 41817 are shown. On top the plasma current, in the middle the DEP radial displacement, on bottom the ramp controller action. On the right column a comparison between a shot without the ramp controller (41902) and one with the ramp controller (41903). 33
2.7 On top the I_p current for the shots: 38513, 18519, 39903 and 40714. Follow, from top to bottom, the HXR, FC, and Vloop diagnostics. . . 40
2.8 shot 41899: after the pellet injection the density value raises but no remarkable effects on FC and HXR diagnostics have been observed. . 42

3.1 TCV poloidal cross section showing the ohmic transformer coils A, B, C, and D, the shaping coils E and F. 49
3.2 The isoflux surfaces of the radial magnetic field on the left (for vertical position control) and of the vertical magnetic field on the right (for radial position). 50
3.3 Simulink scheme of runaway control system at TCV 51
3.4 Simulink scheme of the switching integrator on plasma current: x_1 will be added to plasma current PID regulator 53

3.5 Plasma current control of the shot 57781. I_p tracking on top and switching integrator output on bottom. The switching integrator control is active from 0.3s 54
3.6 Simulink scheme of the PID implemented for radial control (the scheme is equal to the PID for vertical control) 55
3.7 The block schemes that builds the considered radial (or vertical) position from the original signal rIp (or zIp for vertical) 56
3.8 Radial plasma position control of the shot 58138. The plasma radial position tracking on top using the default coordinates rI_p, on center the built coordinates. On bottom the action of the used PID in red, the action that the default PID in open loop in blue, and the fast ramp controller action in black. The new PID radial controller is active from 0.25 s 57
3.9 Vertical plasma position control of the shot 57781. The plasma vertical position tracking on top using the default coordinates zI_p, on center the built coordinates. On bottom the action of the used PID in red, the action that the default PID in open loop in blue. The new PID vertical controller is active from 0.3 s 57
3.10 The fast ramp controller block scheme 58
3.11 The fast ramp controller block scheme 58

4.1 Top level project organization. The brainstorming group proposes requirements which are filtered by the coordinator and later delegated to the development team. 61
4.2 The blue V is in charge of developing the MARTe framework. The red V guarantees the framework quality assurance. It should be noted that this double-V-model does not prescribe any given software lifecycle methodology. 63
4.3 Relationship between the QA plans. Quality, configuration management and verification are integral processes. 64
4.4 All the possible states and transitions of the verification and validation phases. 65
4.5 Snapshot of the Agile board. Each user-story is associated to a Redmine issue where its QA lifecycle is fully audited. 67
4.6 The chosen Git workflow, which is based on the branching model presented in [10]. 69
4.7 How to choose STM32F4-Discovery platform in STM-Cube configuration tool 73
4.8 System core clock configuration 74
4.9 FreeRTOS configuration 75
4.10 High resolution timer (TIM6) configuration with a resolution of 1μs . 76

4.11 The system architecture is composed of four main components. The external PC is optional and allows to monitor and configure the embedded system. 82
4.12 Manipolatore Scortec ER-I 83
4.13 On the left, the PU input D-37-pin connector. The 8-bit parallel PA port is used as command line to communicate with the internal PIC, the PB port is used to read the encoder counters for each joints and there are 6 inputs for motor control signals. The single motor control signal must be analogic with a voltage $\in [-5, +5]\,V$. On the right the PU output D-50-pin connector. Mn+ is the positive pole of the motor n, Mn- the negative, CH_An and CH_Bn are the channels A and B of the encoder sensor on the joint n. MSn (micro switches) are the digital signals of the limit switch sensor of the joint n. These sensors are not present on the Scortec-ER, but are mounted on other models like the Scorbot robotic arms series. 84
4.14 Power unit interface to the control output. A low-pass filter translate the PWM voltage to the required analog range. 85
4.15 Low pass filter scheme (with amplification and offset) for the translation between the PWM and the power supply analog input. 85
4.16 Frequency response of the filtering stage electronic circuit with a sinusoidal input of 10V amplitude. 86
4.17 Output of the filter with 50% (blue), 0% (green), 100% (red) PWM duty cycles at 1 kHz frequency 86
4.18 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. 87
4.19 Execution sequence of the MARTe2 application that is physically connected to the robot (Application-1). 88
4.20 Execution sequence of the MARTe2 application that is used to simulate the connection to the robot (Application-2). 88
4.21 On the right *Bare-metal* application cycle time computed over 40000 samples. On the left the 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. 89
4.22 On the right the*Bare-Metal* application execution time computed over 40000 samples. On the left the 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. 89

4.23 On the rith the 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. On the left the FreeRTOS two-threading application execution time computed over 40000 samples. 90
4.24 The DC motor model implemented in the *ModelGAM* 91
4.25 On the right the closed-loop system output with fixed proportional controller (blue), and with switched proportional controller (green). The y-axis joint angular position is in encoder units. On the left the input voltages to the motors with fixed proportional controller (blue), and with switched proportional controller (green). 93
4.26 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 (on the end stop) has been set to 3000 encoders. The y-axis joint angular position is in encoder units and the input voltage in mV. 94
4.27 STM32F4-Discovery CPU (LQFP100) configuration for robotic arm controller application 96
4.28 L298N H-bridge scheme 97
4.29 First PCB schematic. 98
4.30 Second PCB schematic. 99
4.31 Third PCB schematic. 100
4.32 The CoolCar 104
4.33 STM32F4-Discovery configuration for the CoolCar project 105
4.34 Line Follower application state machine. The car can follow the right or the left track alternatively, thus one x_max (when following right track) or x_min (when following left track) goes in feedback to a proportional controller. The state machine changes the value of x_max and x_min allowing the car to recover in some cases the path when one or both tracks have been lost. 107

Bibliography

[1] Jung-Shik Kong, Bo-Hee Lee and Jin-Geol Kim *Design of a Switching PID Controller Using Advanced Genetic Algorithm for a Nonlinear System*, Computational Intelligence and Security: International Conference, CIS 2005, Xi'an, China, 2005

[2] Shabnam Armaghan, Arefeh Moridi and Ali Khaki Sedigh *Design of a Switching PID Controller for a Magnetically Actuated Mass Spring Damper* , Proceedings of the World Congress on Engineering 2011 , Vol III WCE, July, London, 2011.

[3] Rein Luus and Yang Quan Chen *Optimal Switching Control via Direct Search Optimization*, Asian Journal of Control, Vol. 6, No. 2, pp. 302-306, 2004.

[4] Efstratios Skafidas, Robin J. Evans, Andrey V. Savkin and Ian R. Petersen. *Stability results for switched controller systems*, Automatica, Vol. 35, No. 4, pp. 553-564, 1999.

[5] D. Carnevale, S. Galeani and M. Sassano. *On semiclassical solutions of hybrid regulator equations*, 21st Mediterranean Conference on Control and Automation (MED), pp. 868-876, 2013, doi: 10.1109/MED.2013.6608824.

[6] Daniel Liberzon. *Switching in Systems and Control*, Springer, 2003

[7] L. Zaccarian and A. R. Teel, *The $\mathcal{L}_2$ bumpless transfer problem for linear plants: Its definition and solution*, Automatica, Vol. 41, pp. 1273-1280, 2005.

[8] Hai Lin and P.J. Antsaklis. *Stability and stabilizability of switched linear systems: A survey of recent results.*, Automatic Control, IEEE Transactions on, Vol. 54, No. 2, pp. 308?322, 2009.

[9] D. Carnevale, S. Galeani, L. Menini and M. Sassano, *Robust semiclassical internal model based regulation for a class of hybrid linear systems*, IFAC Proceedings Volumes, Volume 47, Issue 3, pp.1519 - 1524, 2014.

[10] D. Carnevale, S. Galeani and M. Sassano, *A linear quadratic approach to linear time invariant stabilization for a class of hybrid systems*, 22nd Mediterranean Conference on Control and Automation, pp. 545-550, 2014.

[11] K. Lau and R.H. Middleton. *Switched integrator control schemes for integrating plants.*, European Journal of Control, Vol. 9, No. 6, pp. 539 ? 559, 2003.

[12] L. Zaccarian, D. Nesic and A.R. Teel. *First order reset elements and the clegg integrator revisited*, Proceedings of the American Control Conference, Vol. 1, pp. 563?568, 2005.

[13] 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*, 54th Conference on Decision and Control (CDC), December 15-18, Osaka, Japan, 2015.

[14] D. Carnevale, S. Galeani and L. Menini, *Case study for hybrid regulation: Output tracking for a spinning and bouncing disk*, 21st Mediterranean Conference on Control and Automation, pp. 858-867, 10.1109/MED.2013.6608823, 2013.

[15] D. Carnevale, S. Galeani, L. Menini and M. Sassano, *Output Regulation of Hybrid Linear Systems with Unpredictable Jumps*, IFAC, Vol.47, No. 3, pp. 1531 - 1536, 201.

[16] G. Ferró, *https://github.com/gferro90/SwitchOptimization/tree/master*, simulation code.

[17] G. De Tommasi, F. Piccolo, A. Pironti and F. Sartori, *A flexible software for real-time control in nuclear fusion experiments*, Control Engineering Practice, Vol. 14, No. 11, pp. 1387 - 1393, 10.1016/j.conengprac.2005.10.001, 2006.

[18] 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, 10.1109/TNS.2011.2120622, 2011.

[19] M.L. Walker and D.A. Humphreys. *A Multivariable Analysis of the Plasma Vertical Instability in Tokamaks.* 2213 - 2219. 10.1109/CDC.2006.377347.

[20] M. Lehnen et al., *Disruptions in ITER and strategies for their control and mitigation*, Journal of Nuclear Materials, 2014 (doi 10.1016/j.jnucmat.2014.10.075).

[21] H. M. Smith and E. Verwichte, *Hot-tail runaway electron generation in tokamak disruptions*, Physics of Plasmas Vol. 15 N. 7, 072502, 2008.22

[22] S. Putvinski, *Disruption Mitigation in ITER?, 23rd IAEA Fusion Conference*, Vol 43, N. 20, ITR/1-6, 2010.

[23] E Hollmann et al., *Status of research toward the ITER disruption mitigation system* In Physics of Plasmas, Vol. 22 , 021802, 2015.

[24] B. Esposito et al. , *Disruption control on FTU and ASDEX upgrade with ECRH*, Nucl. Fus. 49, 065014, 2009.

[25] N. W. Eidietis et al. , *Control of post-disruption runaway electron beams in DIII-D*, Phys. Plasmas 19, 056109, 2012.

[26] V. Lukash et al., *Study of ITER plasma position control during disruptions with formation of runaway electrons*, 40th EPS, 2013 - http://ocs.ciemat.es/EPS2013PAP/pdf/P5.167.pdf.

[27] J. A. Snipes et al., *Physics of the conceptual design of the ITER plasma control system* Fusion Engineering and Design 89, pp. 507?511, 2014.

[28] M. Lehnen et al., *Suppression of Runaway Electrons by Resonant Magnetic Perturbations in TEXTOR Disruptions*, Physical review Letters, 100, 255003, 2008.

[29] G. Papp et al. , *Runaway electron losses enhanced by resonant magnetic perturbations*, Nucl. Fus., 51, 043004, 2011.

[30] A. Matsuyama, M. Yagi and Y. Kagei, *Stochastic Transport of Runaway Electrons due to Low- order Perturbations in Tokamak Disruption*, JPS Conf. Proc. , 015037, 2014.

[31] M. Vlainic et al., *Post-Disruptive Runaway Electron Beam in COMPASS Tokamak*, arXiv:1503.02947v1, physics.plasm-ph, 2015.

[32] F. Saint-Laurent et al., *Control of Runaway Electron Beam Heat Loads on Tore Supra*, 38th EPS Conference on Plasma Physics, 03.118, 2011.

[33] E. M. Hollmann et al., *Control and dissipation of runaway electron beams created during rapid shutdown experiments in DIII-D*, Nucl. Fusion, 53, 083004, 2013.

[34] M. Lehnen et al., *Disruption mitigation by massive gas injection in JET*, Nucl. Fusion, Vol. 51, N. 12, 123010, 2011.

[35] O. Tudisco at al. , *The Diagnostic Systems in the FTU*, Fusion Science and technology, Chapter 8, Vol. 45, 2004.

[36] Cianfarani C. et al., *MHD signals as disruption precursors in FTU*, Proc. 40th EPS Conf. on Plasma Physics, P5.165, 2013.

[37] N. Commaux, L. Baylor, S. Combs, N. Eidietis, T. Evans, C. Foust, E. Hollmann, D. Humphreys, V. Izzo, A. James, T. Jernigan, S. Meitner, P. Parks, J. Wesley, J. Yu, *Novel rapid shutdown strategies for runaway electron suppression in DIII-D*, Nucl. Fusion 51 (10) (2011) 103001. doi:10.1088/0029-5515/51/10/103001.

[38] A. C. Neto, F. Sartori, F. Piccolo, R. Vitelli, G. De Tommasi, L. Zabeo, A. Barbalace, H. Fernandes, D. F. Valcarcel, and A. J. N. Batista, *Marte: A multi-platform real-time framework*, Nuclear Science, IEEE Transactions on, vol. 57, no. 2, pp. 479 ?486, april 2010.

[39] 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.

[40] D. Alves, A. C. Neto, D. F. Valcrcel, R. Felton, J. M. Lpez, 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.

[41] A. Wallander, L. Abadie, H. Dave, F. Di Maio, H. Gulati, C. Hansalia et al., *ITER instrumentation and controlstatus and plans*, Fusion Engineering and Design, vol. 85, no. 3 - 4, pp. 529 ? 534, 2010.

[42] *git*, https://git-scm.com/, 5/26/2016.

[43] *Programming Languages C++*, 2003, ISO/IEC 14882:2003(E).

[44] *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).

[45] *The Scrum Guide*, https://www.scrumalliance.org/why-scrum/scrum-guide, 5/25/2016.

[46] *GitLab*, https://about.gitlab.com/, 5/26/2016.

[47] *A successful Git branching model*, http://nvie.com/posts/a-successful-git-branching-model/, 5/26/2016.

[48] *Redmine*, http://www.redmine.org/, 5/26/2016.

[49] *Scrum and Agile project management plugin for redmine*, http://www.redminecrm.com/, 5/25/2016.

[50] *Google Test*, https://github.com/google/googletest, 5/26/2016.

[51] *gcov*, https://gcc.gnu.org/onlinedocs/gcc/Gcov.html, 5/26/2016.

[52] *COV-the LTP GCOV extension*, http://ltp.sourceforge.net/coverage/lcov.php, 5/26/2016.

[53] *FlexeLint for C/C++*, http://www.gimpel.com/html/flex.htm, 5/25/2016.

[54] *Eclipse CDT (C/C++ Development Tooling)*, https://eclipse.org/cdt/, 5/26/2016.

[55] *Doxygen*, http://www.stack.nl/ dimitri/doxygen/, 5/26/2016.

[56] *Eclox*, http://home.gna.org/eclox/, 5/26/2016.

[57] *Jenkins*, https://jenkins.io/, 5/26/2016.

[58] *Cobertura*, http://cobertura.github.io/cobertura/, 5/26/2016.

[59] *SonarQube*, http://http://www.sonarqube.org/, 5/26/2016.

[60] G. Ferrò, A. C. Neto, F. Sartori, L. Boncagni, D. Carnevale, M. Gospo- darczyk, A. Monti, A. Moretti, R. Vitelli, L. Capella, and I. Herrero, *Embedded implementation of a real-time switching controller on a robotic arm,* in Proc. 20th IEEE-NPSS Real Time Conf. (RT), 2016.