POLITECNICO
MILANO 1863

Microcontroller Implementation of Event-Based Controllers

SUPERVISOR
ALBERTO LEVA

Shakeel Haider (901458) | MS Automation & Control Engineering

Academic Year 2019-2020

Table of Contents

Table of Figures

Abstract

Feedback control algorithms are essential for the proper functioning of many industrial systems. A lot of research in the digital feedback control field is devoted to periodic and event-based control systems. Moreover, in a considerable number of applications, these control algorithms are implemented in an embedded and real-time software environment, namely on microcontrollers. These control systems, due to their periodic nature, pose strong, non-negotiable requirements on the implementations of their algorithms as the only way to guarantee the required control performance. This, however, might lead to a non-optimal utilisation of the hardware resources: for example, in a periodic digital controller a new value for the control variable is computed in every cycle, also when the process reached steady state. This clearly results in an unnecessary resource usage, like processor time and communication bandwidth.

This thesis analyses control algorithms applicable to embedded control systems, allowing for a more balanced resource usage and dropping the strict requirement in terms of periodic execution. This results also in a reduced power consumption and cost. A possible way to obtain this, can be varying the sample frequency over time and dynamically schedule the control algorithms to optimize over processor load. Another possibility is performing a control update only when the controlled variable goes beyond a defined threshold

The second technique outlined, also allows to reduce the resources' utilisation in terms of communication bandwidth. The main idea is to only execute the control algorithm when it is necessary from a control performance point of view.

In summary, two types of control methods are analyzed experimentally. The results clearly indicate the potential benefits of the second control method, with respect to the overall system performance and in making trade-offs between control performance, software performance and power efficiency.

Sommario

Gli algoritmi di controllo in anello chiuso sono essenziali per il corretto funzionamento di molti sistemi industriali. Molta della ricerca nel campo del controllo digitale in anello chiuso è dedicata ai sistemi di controllo periodici ed event-based. Inoltre, in un numero considerevole di applicazioni, questi algoritmi di controllo sono implementati in un ambiente software di tipo embedded e in tempo reale, in particolare su microcontrollori.

Le tecniche di controllo a passo fisso, a causa della loro natura, pongono requisiti stringenti per quanto riguarda l'implementazioni dei loro algoritmi come unica modalità in grado di garantire le prestazioni di controllo richieste. Ciò, tuttavia, potrebbe portare a un utilizzo non ottimale delle risorse hardware: ad esempio, in un sistema di controllo digitale a passo fisso, viene calcolato un nuovo valore per la variabile di controllo in ogni ciclo, anche quando il processo ha raggiunto il punto di lavoro richiesto. Ciò comporta chiaramente un impiego di risorse non più necessarie, come il tempo di processore e la banda di trasmissione dati.

Questa tesi analizza gli algoritmi di controllo applicabili a sistemi di controllo embedded, consentendo un utilizzo più equilibrato delle risorse ed eliminando i severi requisiti in termini di esecuzione periodica. Ciò comporta anche un consumo di energia ridotto. Un possibile modo per ottenere questo può essere la variazione della frequenza di campionamento nel tempo e la pianificazione dinamica degli algoritmi di controllo al fine di ottimizzare il carico del processore. Un'altra possibilità è eseguire un aggiornamento del controllo solo quando la variabile controllata supera una soglia ben definita.

La seconda tecnica delineata consente inoltre di ridurre l'utilizzo delle risorse di trasmissione dati. L'idea principale è quella di eseguire l'algoritmo di controllo solo quando è necessario dal punto di vista delle prestazioni di controllo.

In sintesi, vengono analizzati sperimentalmente due tipi di metodi di controllo. I risultati indicano chiaramente i potenziali vantaggi del secondo metodo di controllo, rispetto alle prestazioni complessive del sistema e nel bilanciamento tra prestazioni di controllo, prestazioni del software ed efficienza energetica.

Chapter 1

Introduction

In this chapter we mainly discuss about the selection of microcontroller on which we want to implement our control algorithms and why event-based controllers are important for industries.

SELECTION OF MICROCONTROLLER

To efficiently implement a real-time control system, a microcontroller must be used as hardware platform. There is a wide variety of these devices available on the market, however the selection process has to be very accurate and covering many aspects like its processing power, the available hardware support for various communication protocols, etc.

In our case, the "Wandstem" platform has been selected.

Wandstem

This platform has been chosen for its main characteristics:

- High Performance
- MIOSIX OS
- Ultra-Low power consumption

The Wandstem platform is designed to be used in Wireless Sensor Network (WSN) applications ranging from low power to high performance. The on-board microcontroller has a 48MHz 32bit ARM CPU with memory protection, guaranteeing security and dependability of the application.

As regards power consumption, in deep-sleep mode this platform is characterised by having an average drawn current of 2.4μA, enabling years of continuous operation.

The software development for this platform is based on the Miosix OS, a versatile operating system for embedded devices which allows to write application code in C and C++ languages and offers a wide varied of peripheral drivers to interface with the on-board hardware.

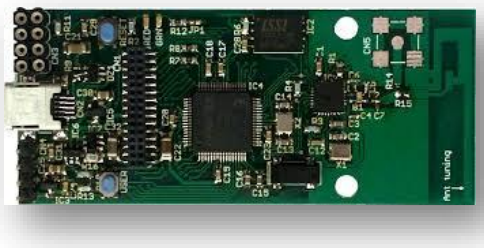

Figure 1 WandStem Microcontroller

INDUSTRIAL POINT OF VIEW

The demand for wireless control networks in the industrial context continues to grow due to their advantages such as flexibility, straightforward installation, and low costs. There are, however, still some prudence in widely investing in wireless solutions due to well-known concerns about the reliability and security of the existing wireless standards and compatibility with established control network infrastructures. This thesis discusses the eligibility and deployment aspects of wireless controls and how the presented control technique is beneficiary in terms of power consumption.

Historically, process industry adopted new technologies cautiously. However, because energy costs are rapidly increasing, emerges an escalating need of reducing energy consumption by means of more sophisticated building control strategies. According to a study conducted by ARC Advisory Group - a research and advisory firm for manufacturing, energy, and supply-chain solutions - the worldwide market for wireless technology in industrial automation will have an annual growth rate of 32 percent, and reached $1.1 billion in 2012.

ORGANIZATION OF THESIS

This thesis is organized as follows: in Chapter 2 the different sampling and event-triggering mechanisms are described, in Chapter 3 is presented the implementation of a software library for periodic control, while in the following one (Chapter 4) its event-based counterpart is presented. In Chapter 5 the results obtained with the two control methods are compared, also in terms of power consumption, while Chapter 6 presents the general conclusions and outlines future research work.

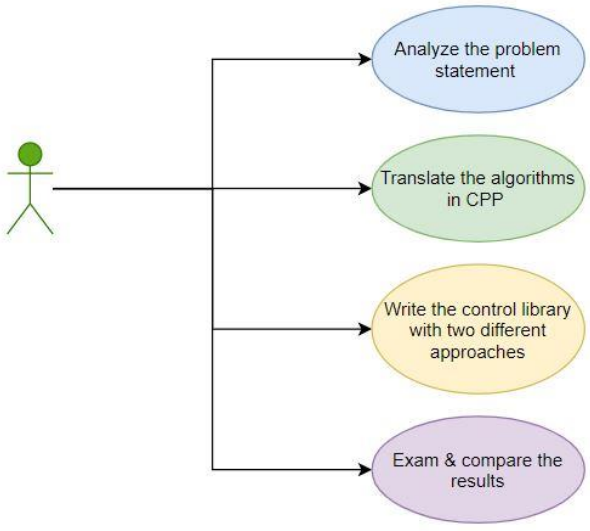

Figure 2 Task division

Chapter 2

Implementation of Sensor Algorithms in C++

The algorithms analyzed were taken from the treatise developed in a precedent thesis entitled “Periodic event-based control with past measurements transmission and multiple control computations”. In this work, the algorithms have been implemented in C++ language to make them suitable for a microcontroller-based control system. For the simulation point of view, only their C++ implementation is relevant.

LEBESGUE SAMPLING

The first technique analyses the *Lebesgue sampling* one. According to this technique, a sample of the process' variable *y* is taken only when its value crosses a predefined level line. As a consequence, if the value of *y* remains for a long time between two subsequent level lines, no event is generated at all. On the other hand, a variation of *y* crossing several level lines, causes a burst of events. Usually the level lines used for event generation are spaced uniformly requiring only a single parameter (spacing) to characterize a Lebesgue sampler.

Lesbesgue Sampling Flowchart

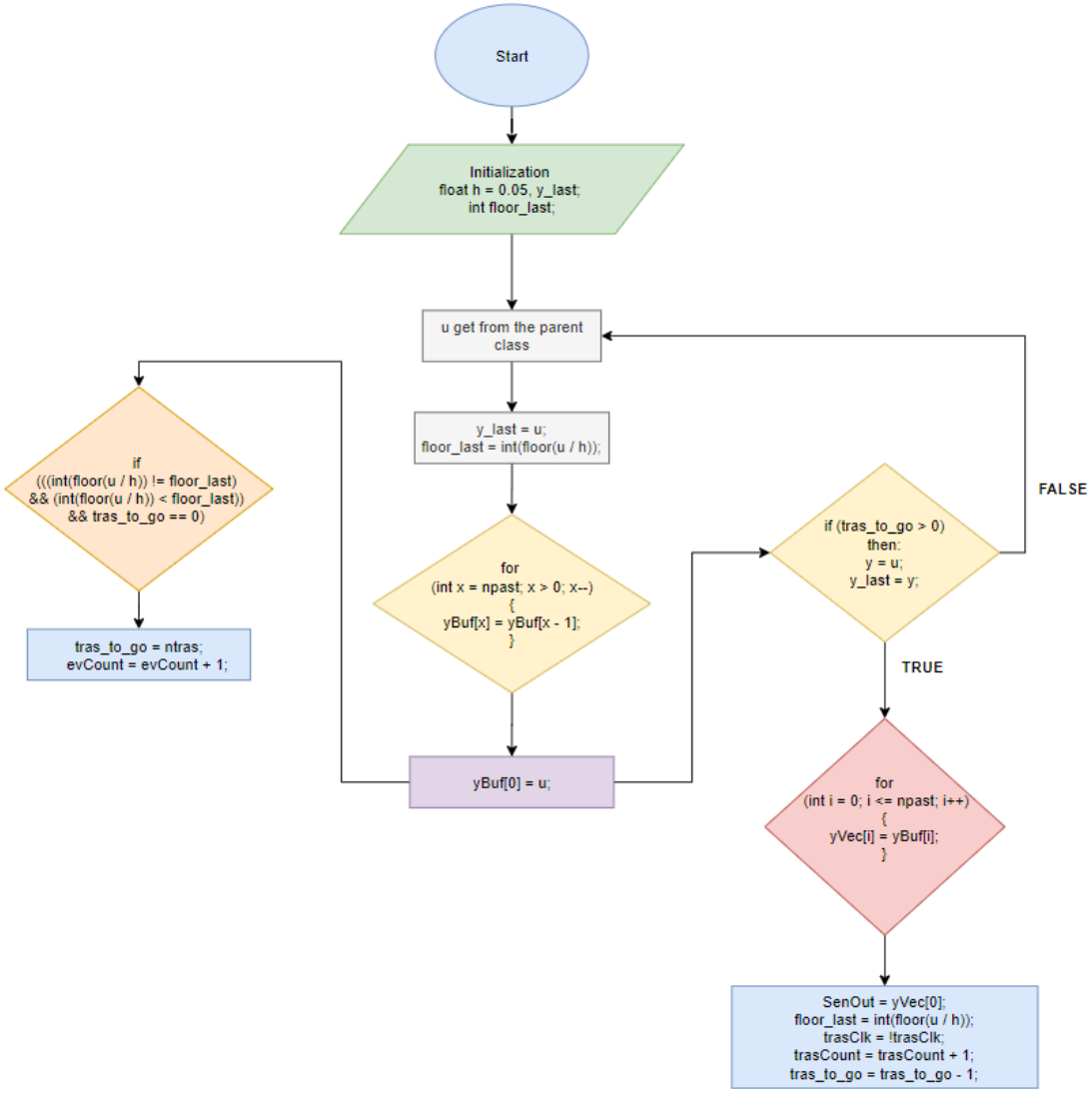

Figure 3 Lesbesgue Sampling Flowchart

EVENT GENERATE ON DELTA

This technique is the most straightforward, as an event is triggered whenever the value Δy of the difference between the current value of the process' variable and the one at the previous event, is greater than a certain threshold. In some cases, two different thresholds for Δy can be used, one for positive and one for negative values, giving rise to symmetric or asymmetric versions of this algorithm.

Sample Send on Delta Flowchart

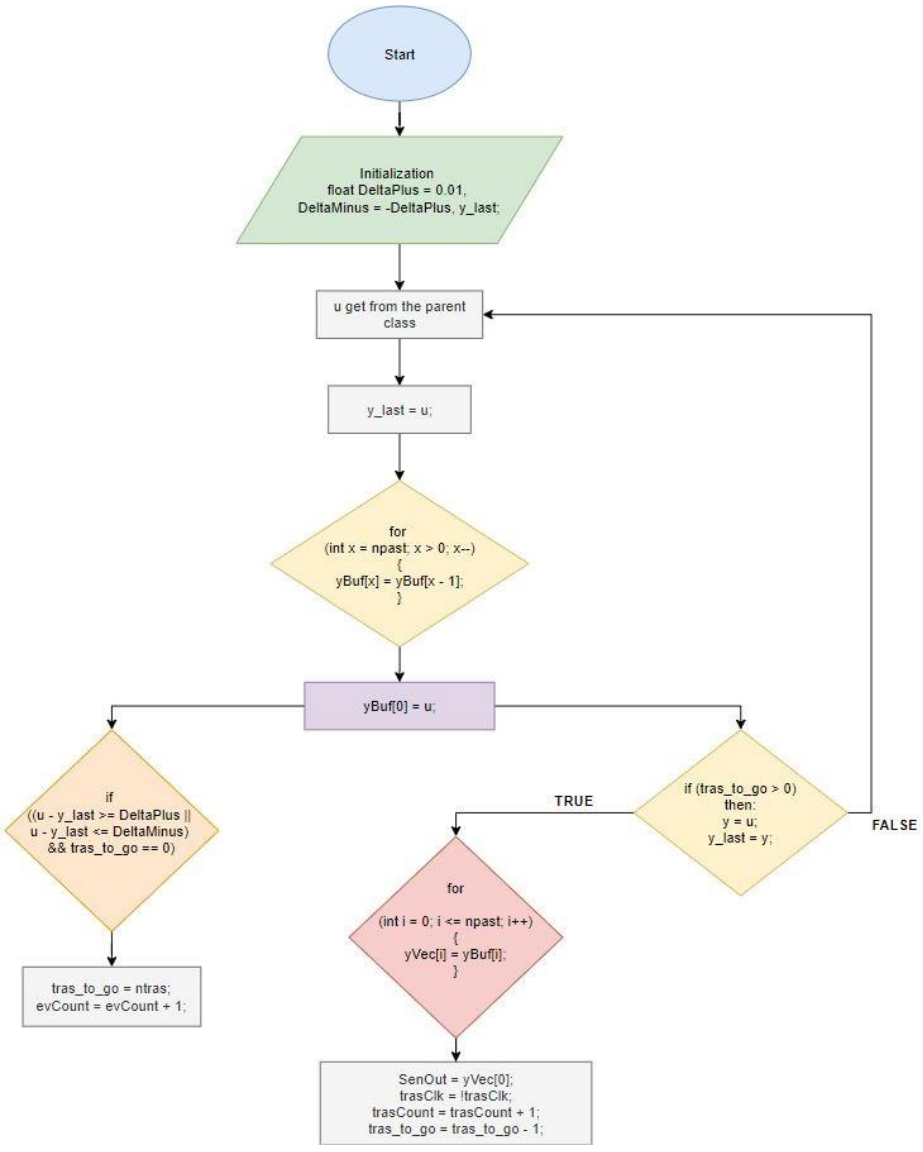

Figure 4 Send on Delta

INTEGRATE AND FIRE

This sampling technique is similar to the previous one, with the exception that here the time integral of Δy is compared against a given threshold. The integral is taken starting from the time instant of the last event triggered and is reset to zero when an event occurs.

Integrate & Fire on Absolute Value Flowchart

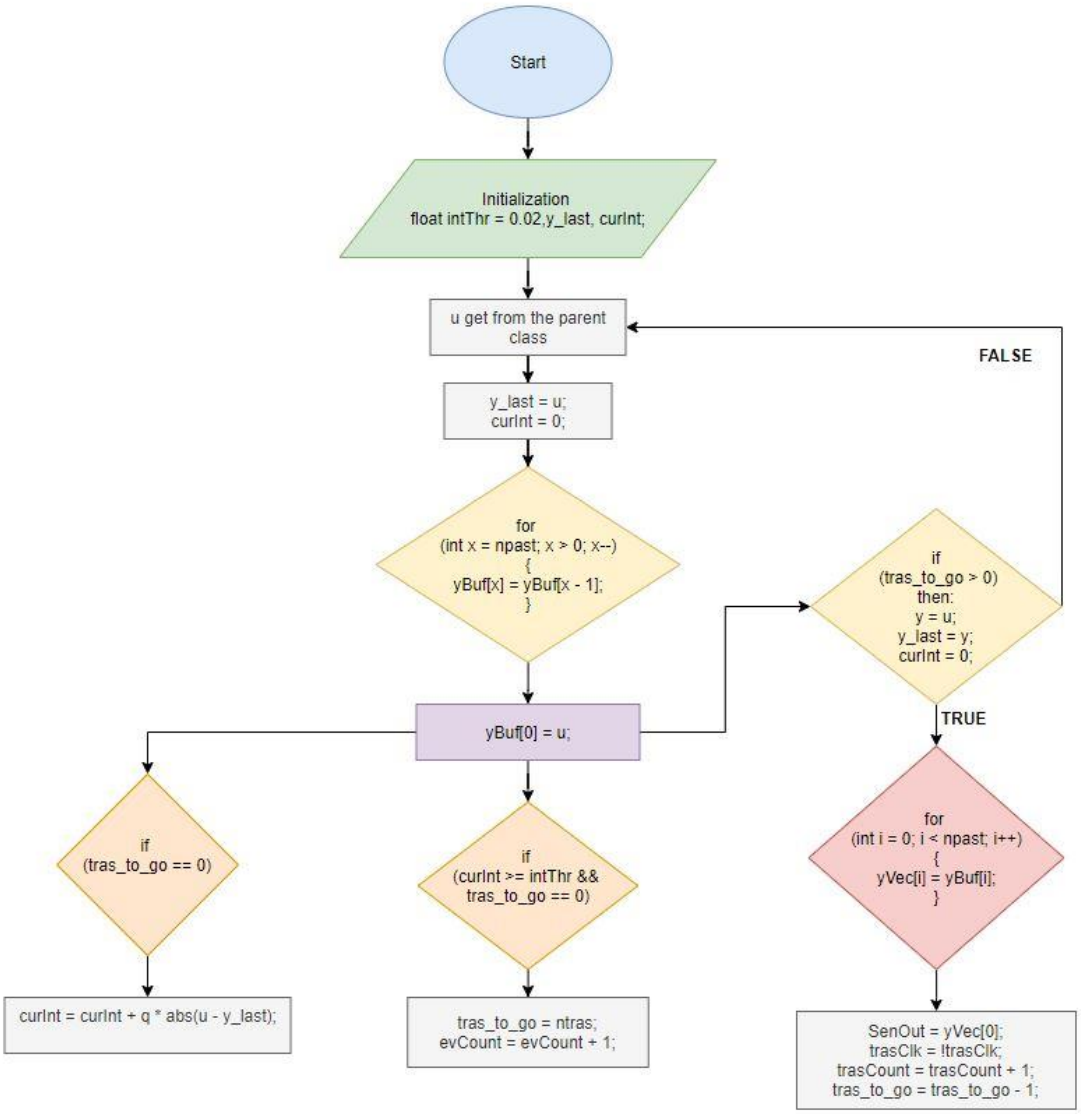

Figure 5 Integrate & fire on abs value

Integrate & Fire on Square Value of y Flowchart

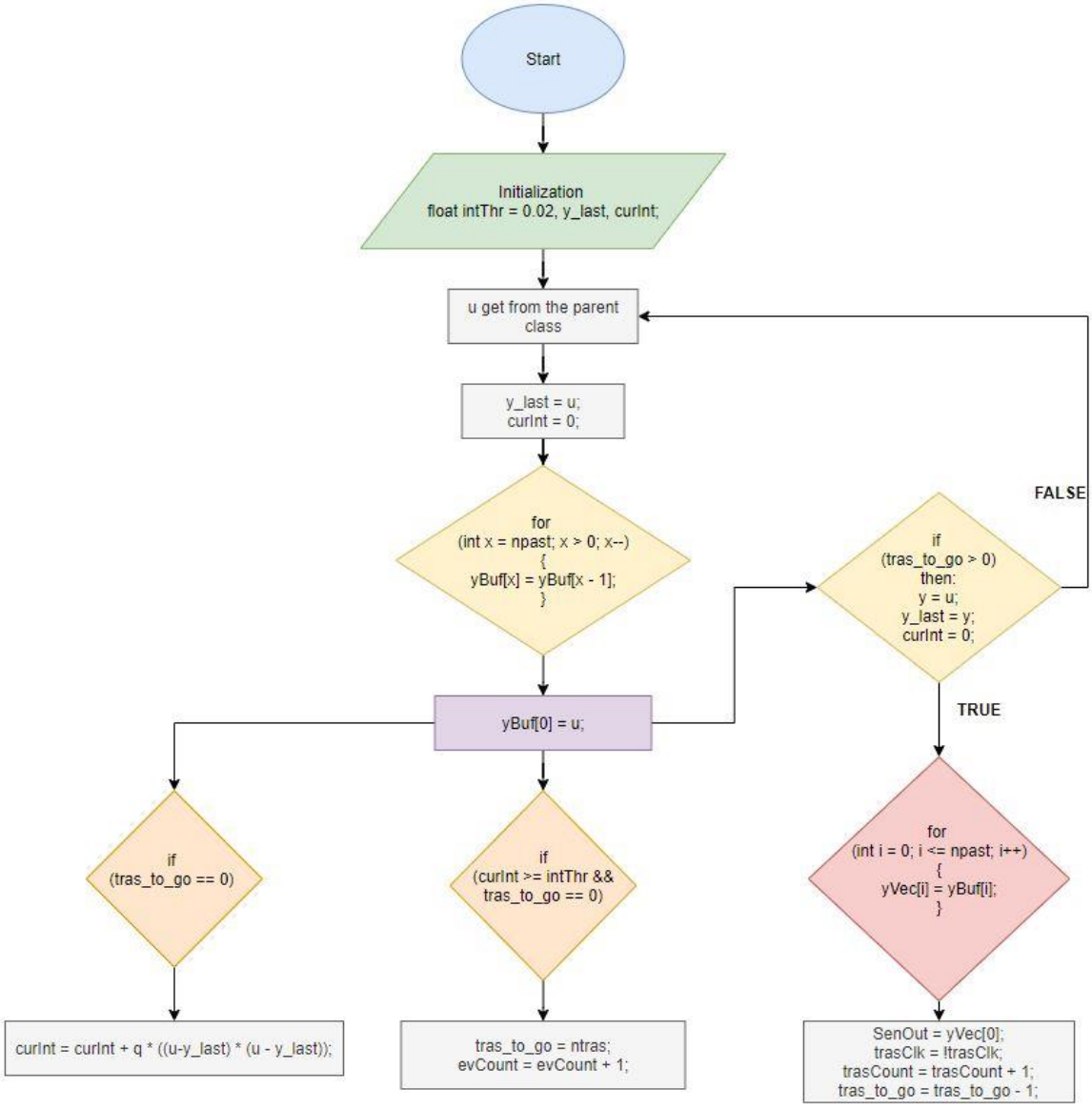

Figure 6 Integrate & Fire on Square value of u

EVENT TRIGGERED ON DELTA TIME OUT

The absence of triggered events generally can be attributed to two causes: the controlled process reached a steady-state situation or, less likely, in case the transducer measuring the process' output. However, being in the first of the above situations, does not necessarily mean that the error between y and its reference value is zero: given the structure of the triggering mechanism, no event is generated whenever Δy is smaller than the triggering threshold, but no check is performed on the control error.

The solution to problems above comes with the introduction of a timeout mechanism, unconditionally triggering an event when a given amount of time elapses.

Event Triggered on Delta Time Out Flowchart

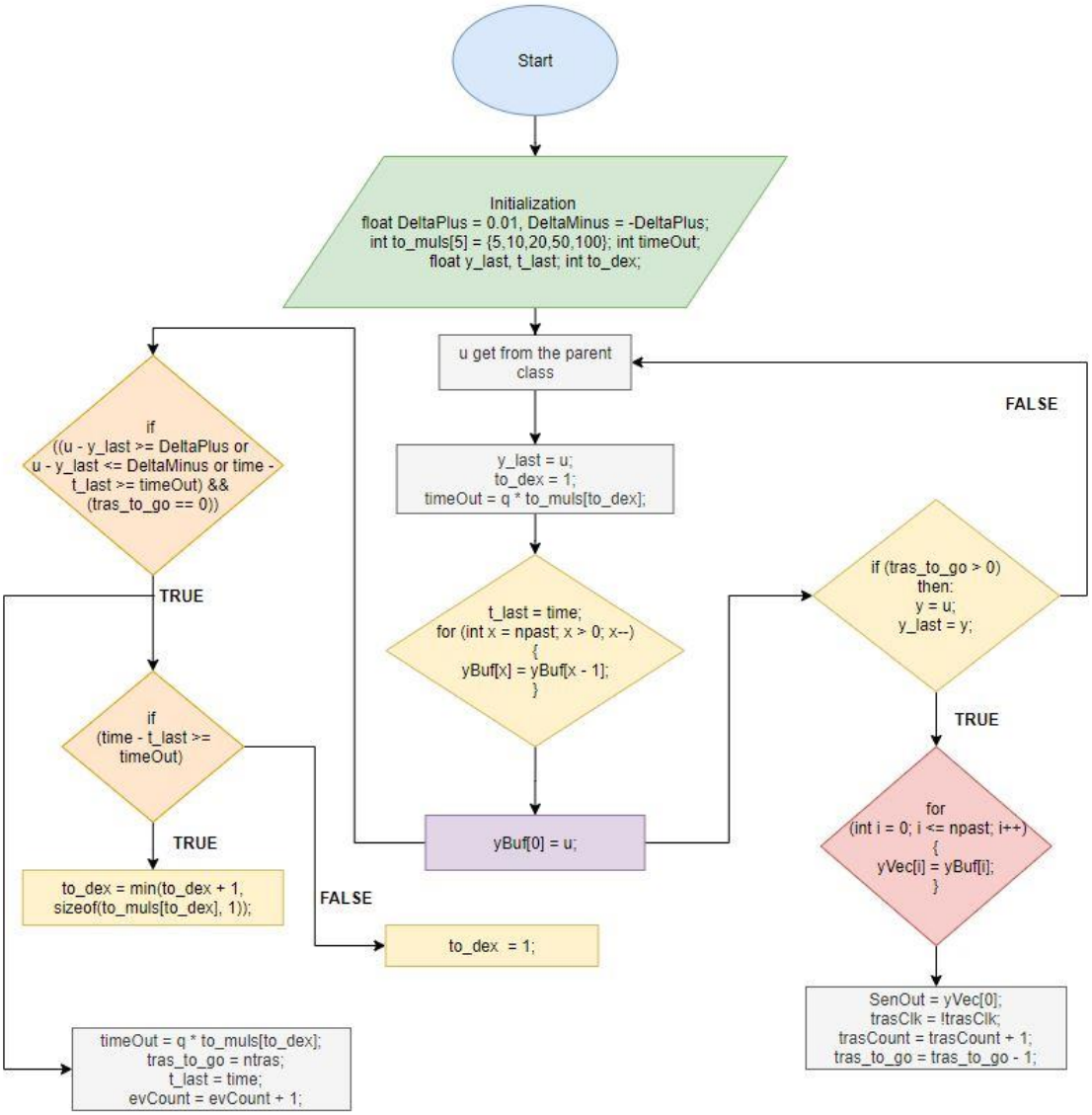

Figure 7 Event triggered on delta timeout

BASE CLASS

This class is used internally by all the implementations by inheritance and contains all the basic and fundamental variables. This module allows to randomly generate the value of *u* or to assign it a user-defined value

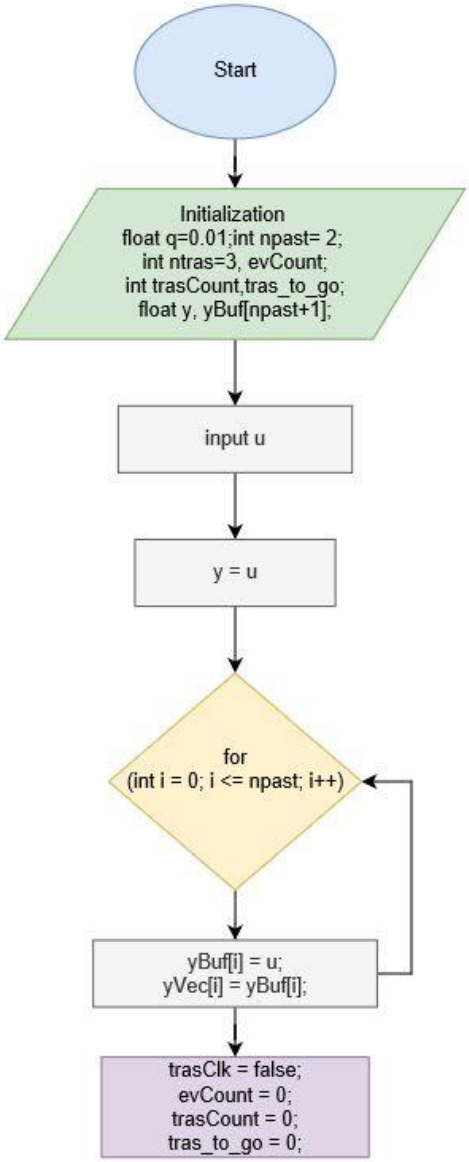

Figure 8 Base Class flowchart

EXECUTION OF SENSOR ALGORITHMS IN CPP

As mentioned before, all the sensing algorithms were implemented in C++ language. Before prosecuting with the analysis, some concepts of object-oriented programming are given to the reader to allow for a better understanding of the subsequent chapters.

Inheritance

Inheritance is one of the major concepts of object-oriented programming, allowing to define a class in terms of another one, making easier to create and maintain an application. This also gives the opportunity to reuse code functionalities and speeds up implementation time: when creating a class, instead of writing completely from scratch new data members and member functions, the programmer can make the new class inherit the members of an existing one, also called "base class". Moreover, a class can be derived from more than one classes, inheriting data and functions from all of them. The inheritance can take different forms, based on an access specifier. The three possible inheritances are listed below:

Public Inheritance: When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class but can be accessed through calls to the public and protected members of the base class.

Protected Inheritance: When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.

Private Inheritance: When deriving from a private base class, public and protected members of the base class become private members of the derived class.

So, for execution we need to consider all these points in our mind.

Working principle

Our implementation is structured in a way such that there is one base class containing all basic and fundamental variables and a simple algorithm described in the previous section. The different sampling techniques, then, are implemented in ad-hoc subclasses, inheriting from the common base one and eventually overriding some of its methods.

The following UML gives a complete outline of the sub classing structure.

UML Diagram of the System

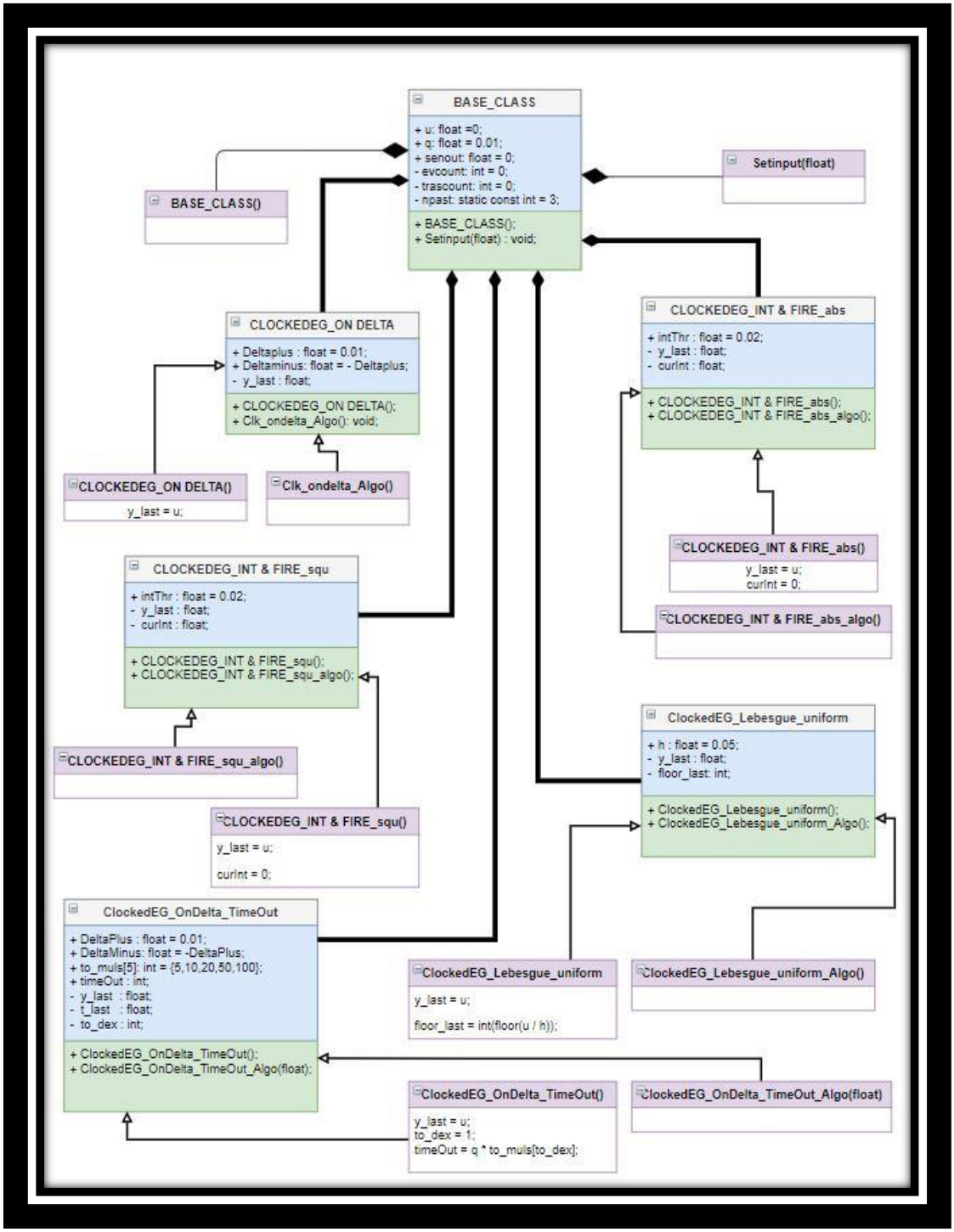

Figure 9 overall system without control lib

Chapter 3

Implementation of control library (1st Approach)

INTRODUCTION

Nowadays, many control techniques - like H2, H∞, LQ, LQG, MPC, PID,… - are eligible to be implemented on a microcontroller. Among them, the control technique chosen for this thesis is the PID one in virtue of its maturity and usability from the industrial point of view. PID control is by far the most common way of implementing feedback control systems and is commonly used in both industrial and laboratory equipment. Currently, in process control more than 95% of the control loops are employ a PID controller structure and most of them in its PI form.

This control technique plays a relevant role also distributed control system and is often combined with logic process control as a building block for large automation systems, like the ones for energy production, transportation and manufacturing. The advent of microcontrollers, then, allowed to provide additional features like automatic tuning, gain scheduling and continuous adaptation.

In this chapter concepts of PID control is discussed, as long as the methods for choosing its tuning parameters. During the analysis of this control technique, which has a linear mathematical behavior, an important nonlinear phenomenon has to be considered, being the actuator's saturation. This, in combination with controllers based on an integral action, gives rise to an effect called integral windup being detrimental for the control system's performance. This phenomenon will be discussed in depth in the following also presenting anti-windup techniques Finally, the microcontroller-based implementation of PID control systems will be presented.

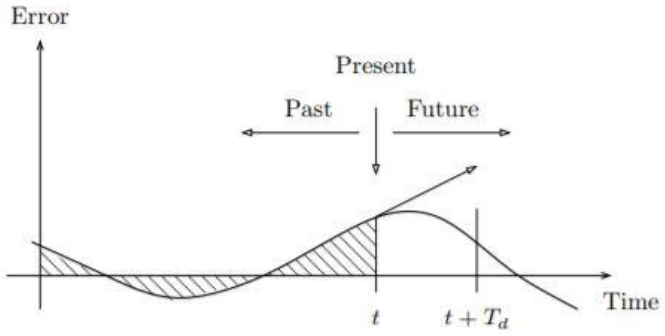

Figure 10 sources of PID controller action

CONTROLLER ALGORITHM

The PID control technique is based on the following equation:

$$u(t) = Kp * e(t) + Ki \int_0^t e(\tau)d\tau + Kd\frac{de}{dt}$$

where *u* is the control signal and *e* is the control error ($e = w - y$). The reference value, *w*, is also called set-point. The control signal, then, is given by the sum of three terms: one proportional to the error, one proportional to the time integral of it and one proportional to the derivative of the error. The controller parameters are proportional gain *Kp*, integral gain *Ki* and derivative gain *Kd*. The above equation can take the following equivalent form:

$$u(t) = Kp\left(e(t) + \frac{1}{Ti}\int_0^t e(\tau)d\tau + Td\frac{de(t)}{dt} \right)$$

where *Ti* is the integral time constant and *Td* the derivative time constant. The proportional part acts on the present value of the error, the integral represents an average of past errors and the derivative can be interpreted as a prediction of future errors. Note that the control signal *u* is formed entirely from the error *e* without any feed forward term.

Effects of Proportional, Integral and Derivative Action

A pure proportional control is obtained setting Ti = ∞ and Td = 0 in the equation above. In this case the steady state error is never zero and decreases with increasing gain which, however, increases the probability of having an oscillatory response.

Figure 12 illustrates the effects of adding the integral action, whose effect increases by decreasing integral time Ti.

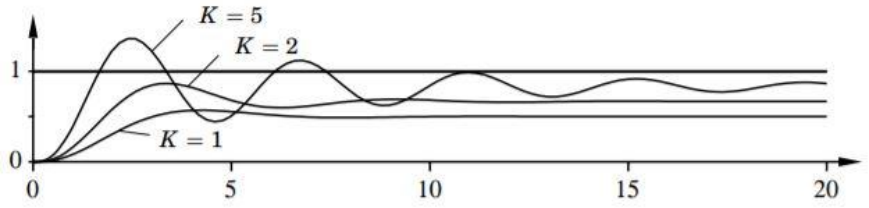

Figure 11 Closed loop system which proportional control

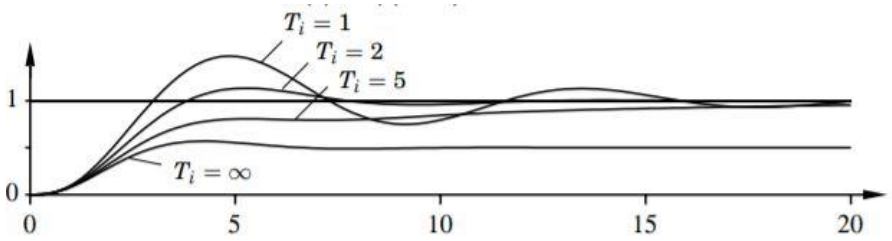

Figure 12 closed loop with proportional & integral action

Figure 13 shows the effects of adding a derivative action: the parameters *K* and *Ti* were deliberately chosen such that the closed-loop has an oscillatory response. Damping increases with derivative time but decreases again when derivative time becomes too large.

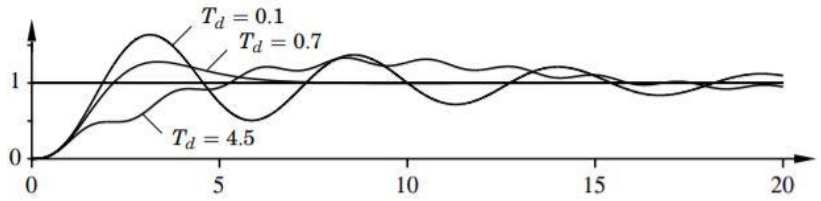

Figure 13 Closed loop with proportional, integral and derivative action

Windup

Windup is a nonlinear phenomenon caused by the presence of actuator saturations when a controller with an integral action is used. All actuators have limitations and, for a control system with a wide range of operating conditions, it may happen that the control variable reaches the actuator limits. When this happens the feedback loop is broken, and the system runs in open loop because the actuator will remain at its limit independently from the process' output. If a controller with integrating action is used, the error will continue to be integrated making integral term to become very large. It is then required that the error has opposite sign for a long period before things return to normality. The consequence is that any controller with integral action gives large transients when the actuator saturates.

Design of discrete time PID controller

In the last years, analog controllers have been replaced by their digital, or discrete-time, counterparts. As all the other digital control systems, the discrete-time PID controller requires the usage of analog to digital converters in order to obtain numerical data from the process' output, which is usually measured using an analog transducer. The equations used to compute a new control action, instead, are obtained by the continuous-time ones by applying the so-called *Z* transform. In the end, a digital to analog converter is required to obtain an analog signal for the process actuators.

The block diagram below shows a closed-loop system having a digital controller, evidencing the converters mentioned before:

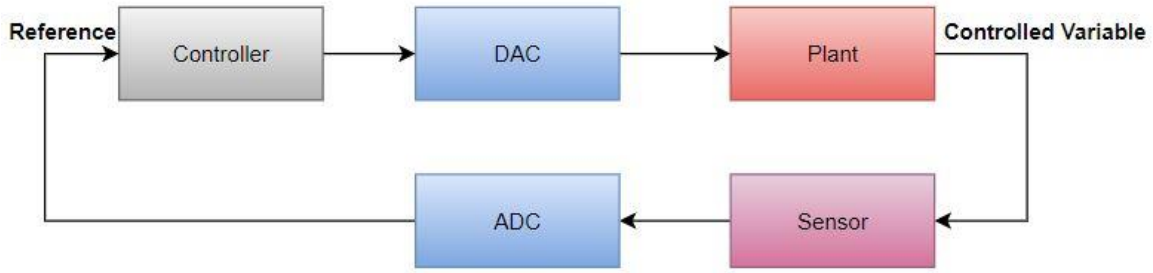

Figure 14 Digital Control

As regards the PID controller's equation, the application of the Laplace Transform to the time-domain equation presented in the previous paragraph lead to the following one, in the frequency domain:

$$C(s) = K_p + \frac{K_i}{S} + K_d * s$$

Then, is a common practice to modify the derivative term to a low-pass filter to make the control variable less susceptible to noise:

$$C(s) = K_p + \frac{K_i}{S} + \frac{K_d * N}{1 + N/s}$$

The continuous-time equation above can be discretised using three methods, namely forward Euler, backward Euler and the trapezoidal ones.

Given a sampling period *Ts*, the integral term can be represented in discrete from as:

$$\frac{Ki*Ts}{Z-1} \text{ or } \frac{Ki*Ts*Z}{Z-1} \text{ or } \frac{Ki*Ts}{2} * \frac{Z+1}{Z-1}$$

Where the first term comes from the usage of the forward Euler method, the second one from backward Euler and the last one from Trapezoidal one.

Similarly, the derivative term is discretised in the following way:

$$\frac{N(Z-1)}{Z-1+N*Ts} \text{ or } \frac{N(Z-1)}{(1+N*Ts)Z-1} \text{ or } \frac{N}{(1+N*Ts)*Z+1/2(Z-1)}$$

It is apparent that, for all terms above the sampling period has an effect on the integral and gains.

For example, using the backward Euler method to obtain the discrete-time representation of a PID controller leads to:

$$C(z) = K_p + \frac{K_i * T_s * z}{z-1} + \frac{K_d * N * (z-1)}{(1+N*T_s)*z-1}$$

In the following a comparison between continuous-time and discrete-time realistions of the same controller is given. The controlled process consists of a DC motor with a low-pass filter at its input. The discretisation step has been set to 0.01s.

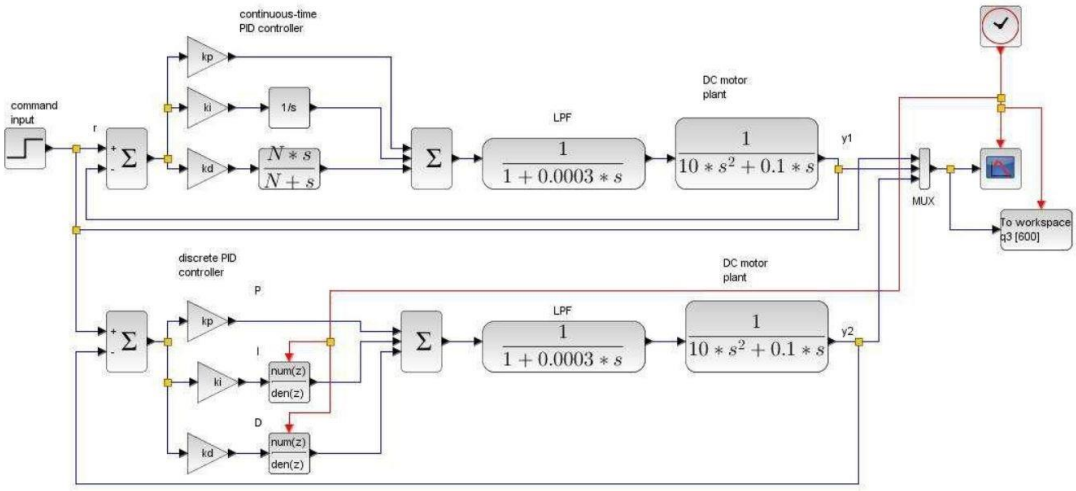

Figure 15 Continuous VS Discrete PID simulation

The images below show the integral and derivative blocks of the controller:

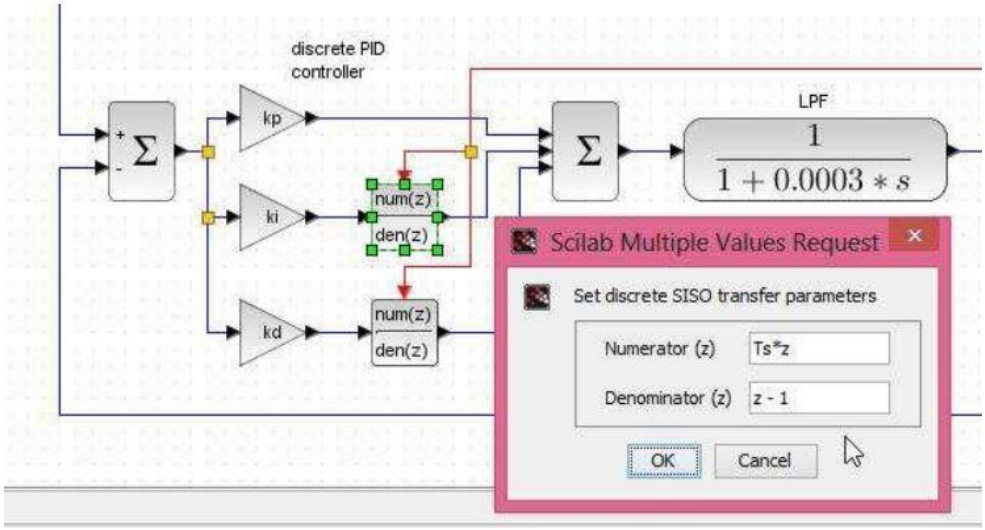

Figure 16 Integral block

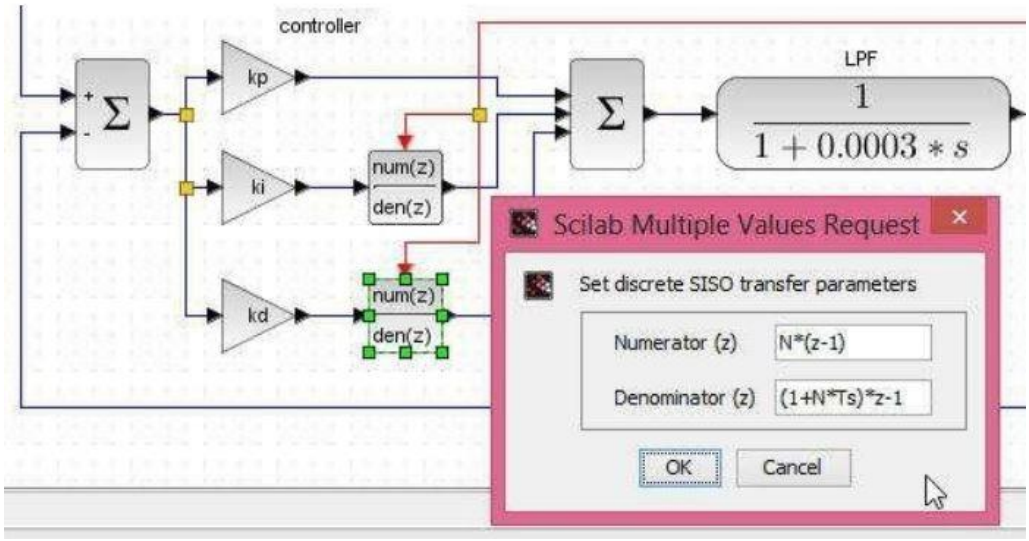

Figure 17 Derivative Block

The results are shown below: the step responses from the continuous (green color) and discrete (red color) time realizations match almost exactly.

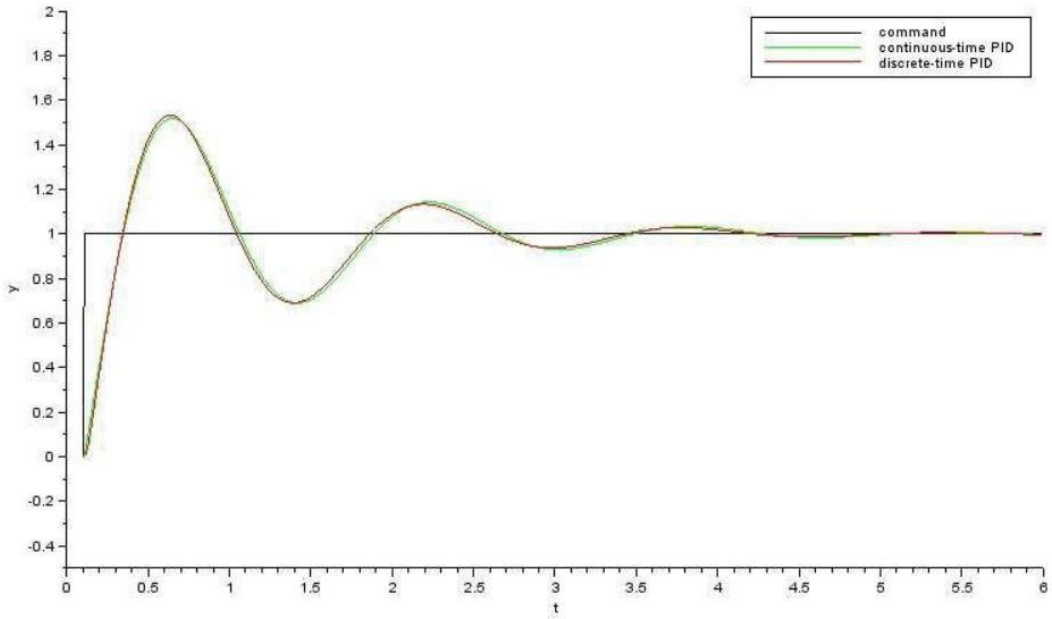

Figure 18 Comparison Response

PID Flowcharts

The periodic calling of the function performing the control update can be realized following two different approaches: the first one is based upon a polling mechanism, while the second makes use of system interrupts. The first approach is intended to be called from the main application code that updates both the error variable and the PID error state variable.

The flow chart above shows the PID controller implementation using the polling mechanism:

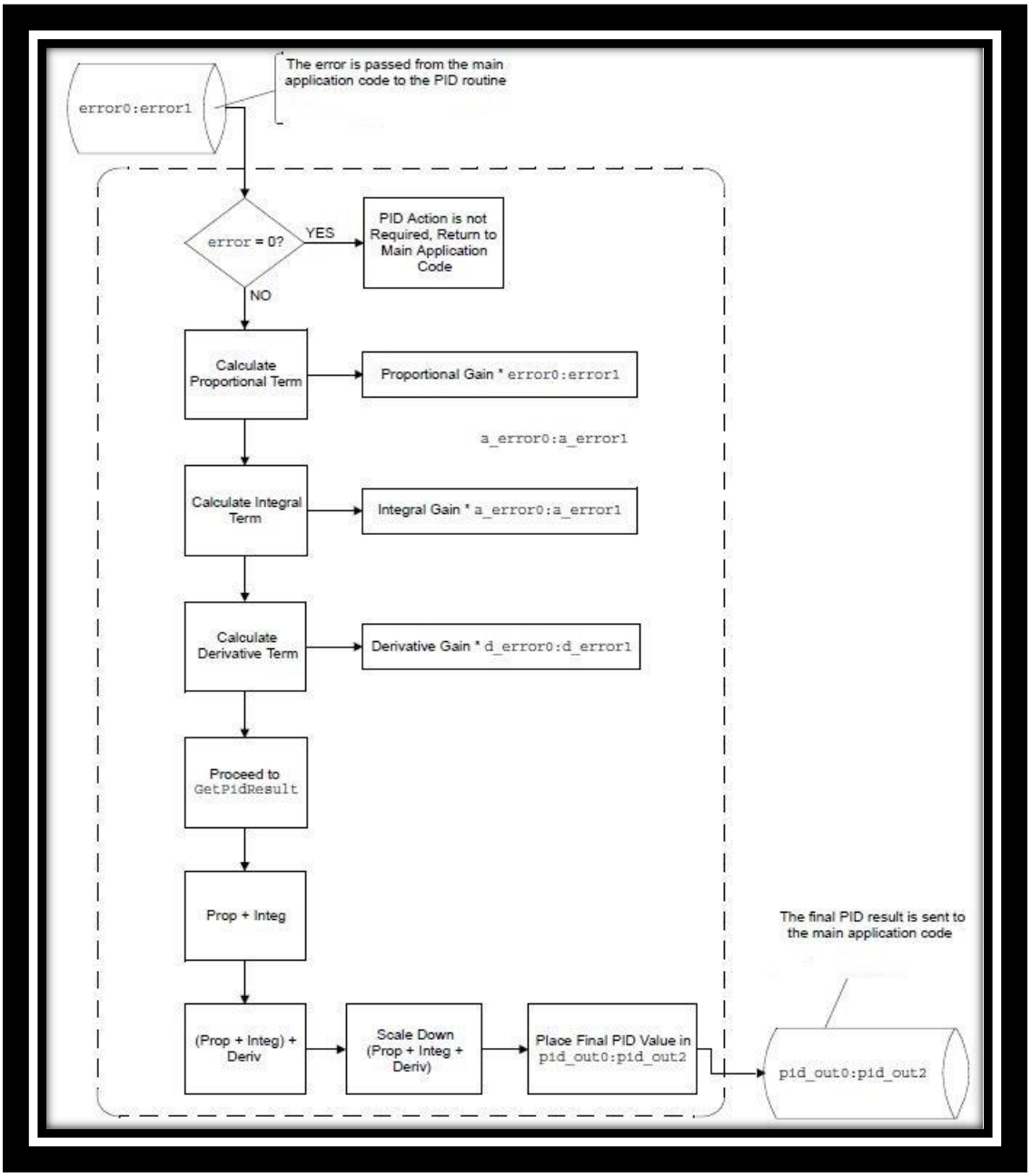

Figure 19 PID main polling method

On the other hand, in the interrupt-based approach, the function computing a new control update is called by an interrupt service routing usually triggered by a periodic timer.

The flow chart below summarizes this approach:

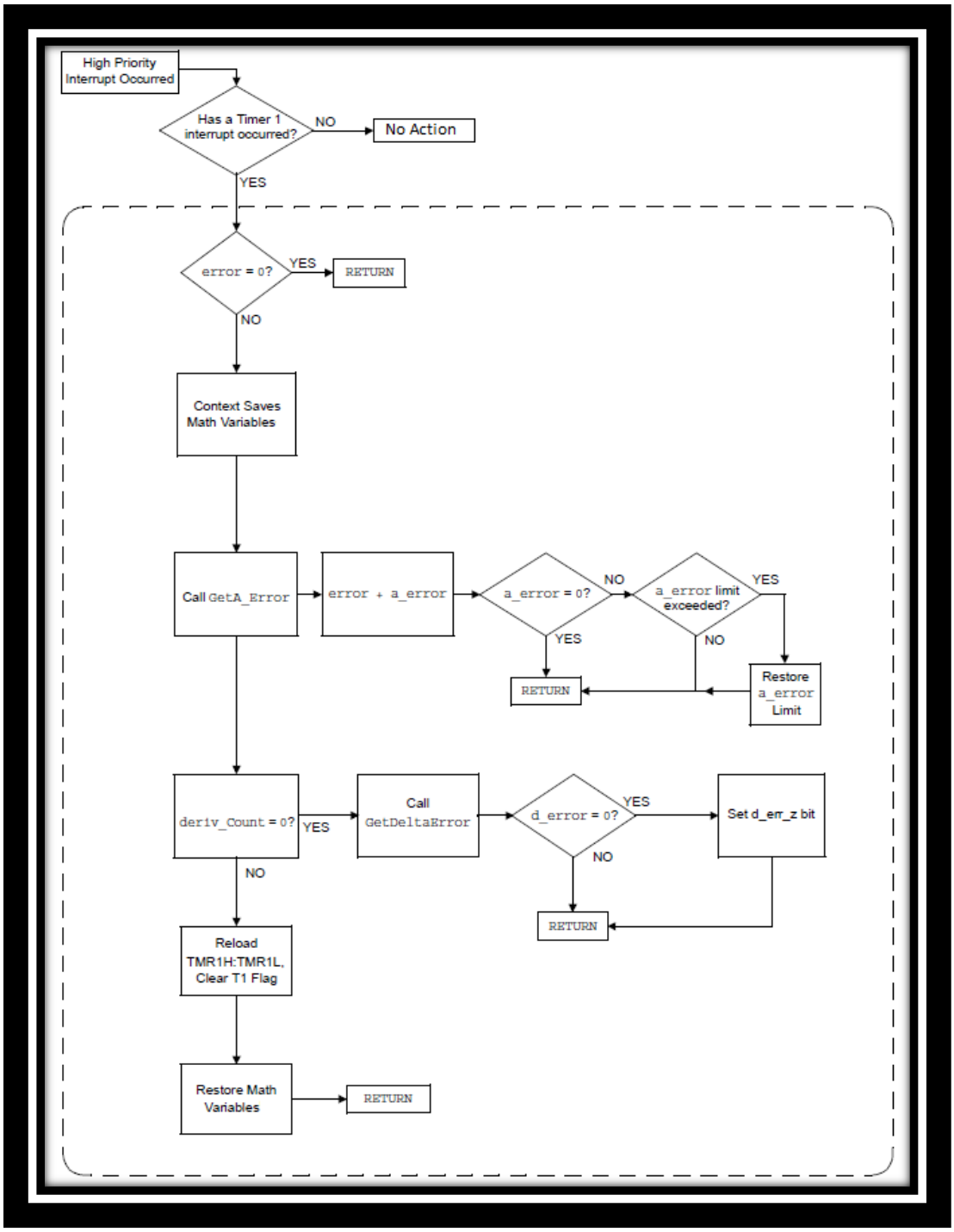

Figure 20 PID using Interrupt

EXECUTION OF PID

In our simulation environment, the process has been assumed to have a first-order transfer function:

$$\frac{Y(s)}{U(s)} = \frac{\mu}{(1 + s * \tau)}$$

Where *u* is input and *y* the output.

- τ: pole's time constant.
- μ: gain

Plant Transfer Function UML

The following UML diagram shows the implementation of the class representing the process transfer function. The class' constructor takes as input all the parameters and assign them to the corresponding class members, while the transfer function equation is stored and evaluated in a dedicated function.

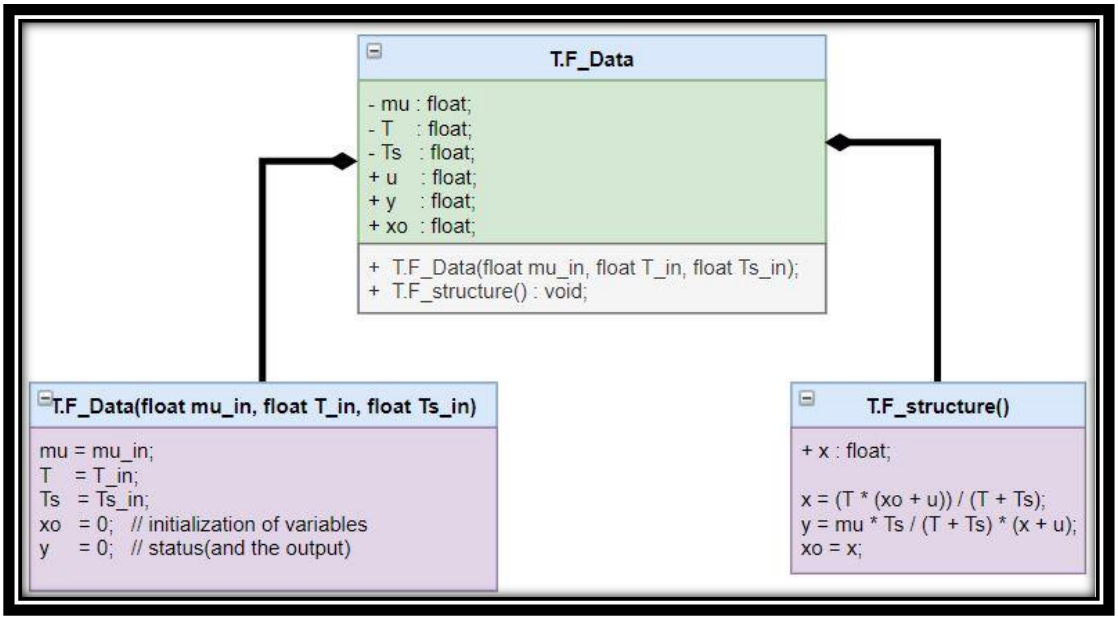

Figure 21 Plant transfer function UML

PID Class UML

For the controller, a dedicated class has been made, keeping some of the parameters in private member functions. The constructor of this class allows to specify the regulator's tuning parameters, while a separate function allows for the computation of a new value for the control variable that is, allows to perform a control step.

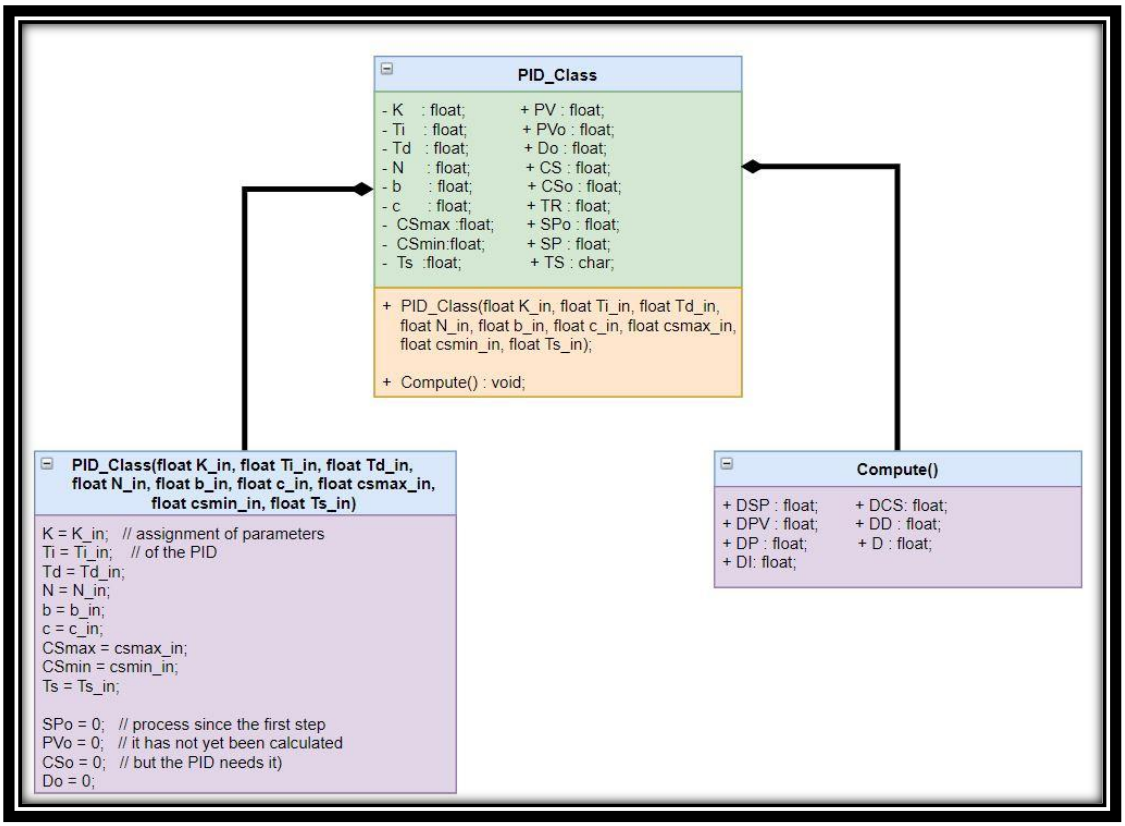

Figure 22 PID_Class UML

Complete system UML

To give the reader a complete picture of the code structure, a summarizing UML diagram has been made. This diagram connects the one shown in figure 9 with the ones describing the structure of the controller and process classes and specifies for a main class governing the behavior of the various parts.

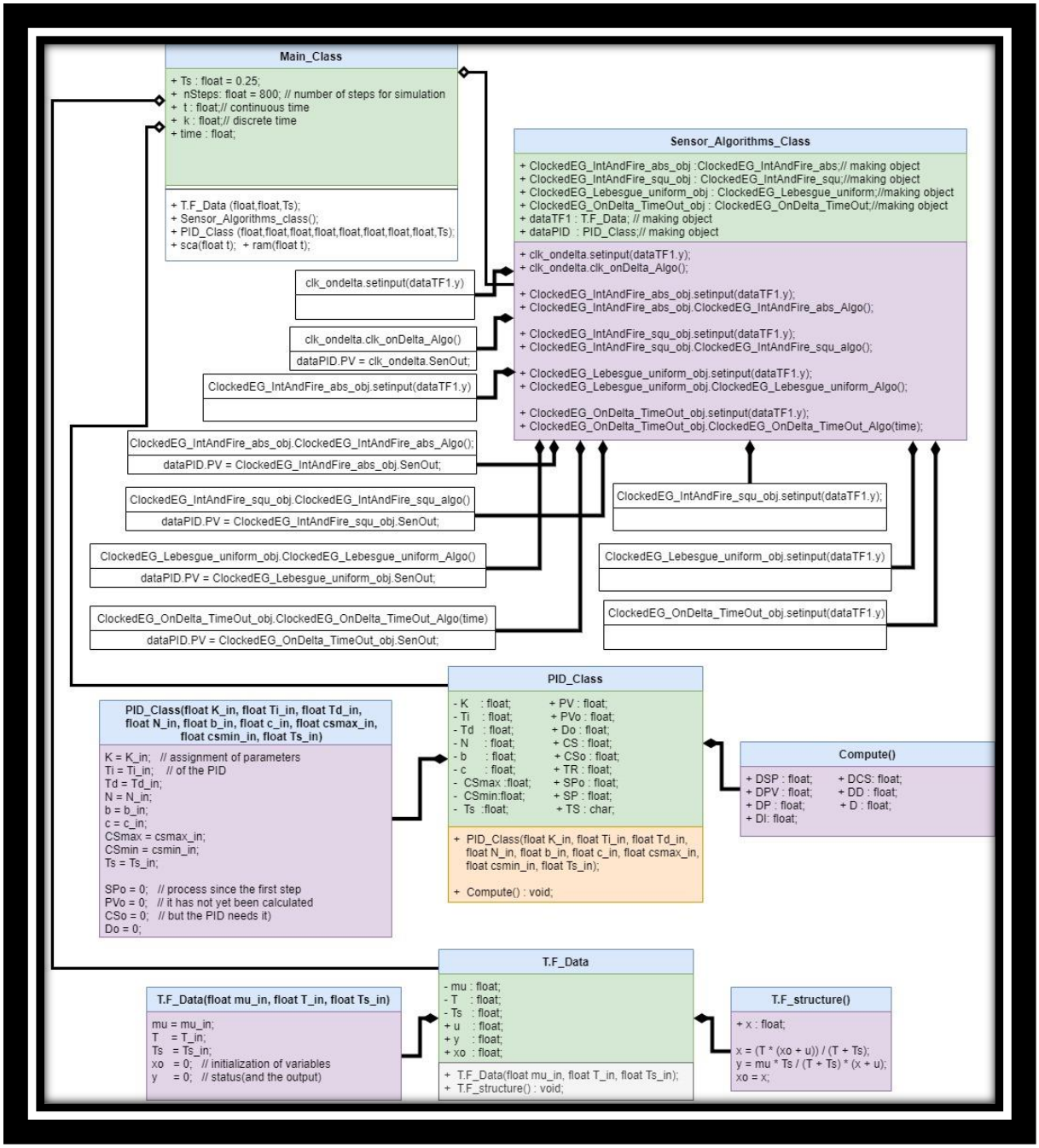

Figure 23 Overall System UML

Code Structure

The complete codebase has been subdivided in different source files, each of which is related to a particular functionality, as shown below.

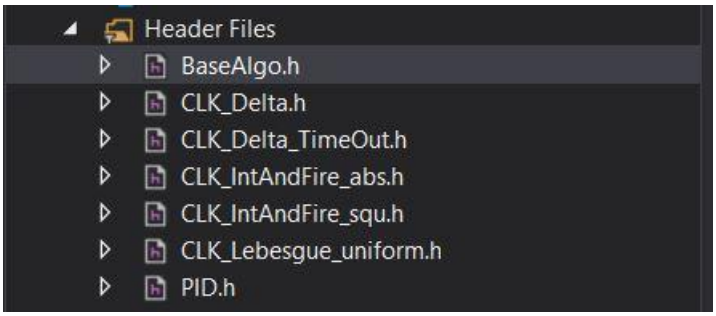

Figure 24 header files

The five different sampling algorithms all inherit from a common base class and there is a dedicated header file for all of them. There is also a separate header file for the class representing the PID controller. This structure allows to select the sampling method to use by including the corresponding header file in the main application source code, as shown below:

```
1  #include <iostream>
2  #include <fstream>
3  #include "CLK_Lebesgue_uniform.h"
4  #include "PID.h"
```

Figure 25 including algorithm & PID in the main source code

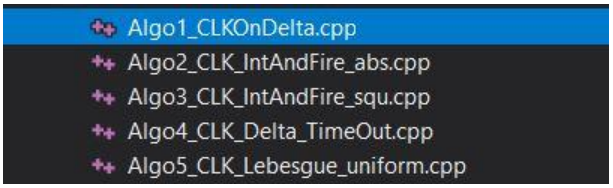

Figure 26 Algo.cpp

Chapter 4

Implementation of control library (2nd Approach)

INTRODUCTION

In the previous chapters we have discussed the effects of proportional, integral, and derivative control and analyzed the continuous and discrete time realizations of the PID controller, also comparing their performance.

Here, an alternative approach is presented, whose key point lies on having the controller not computing a new value for the control action even when steady state error is zero, allowing for savings both in terms of energy and bandwidth. In this implementation, when the controlled process reaches steady state, the microcontroller on which the control system is hosted is put in a deep-sleep mode, keeping active only a small set of its peripherals. The microcontroller is then put back in active mode whenever the value of the error becomes different from zero.

ACTIVE VERSUS SLEEP MODE IN MICROCONTROLLER

In many embedded applications the processor does not need to run continuously, and peripherals are idle for most of the time. The overall power consumption, then, can be lowered by taking advantage of various “sleep” modes available for the microcontroller.

The most common sleep modes are the following:

- Power Down Mode
- Power Save Mode
- Idle Mode

In Power Down mode everything is shut down, including the clock source. In Power Save mode everything is turned off except a real-time clock running from a crystal oscillator which keeps track of time. Idle mode is a shallow sleep mode where only parts of the device are shut down, leaving the other ones running. These different sleep modes allow to have great flexibility in shutting down any part of the microcontroller that is not necessary for the current task. The amount of power that can be saved depends on the sleep mode being used.

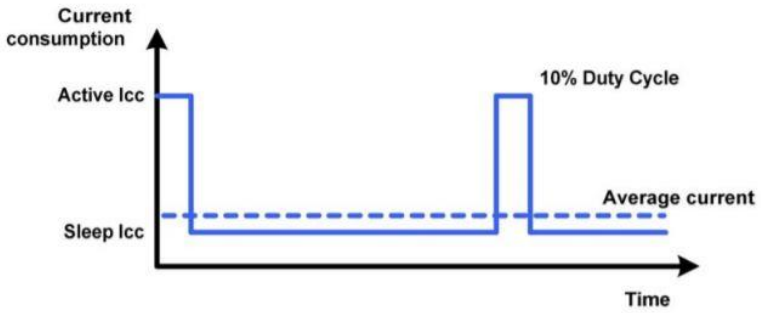

Figure 27 Power saving

Since a microcontroller can spend substantial amounts of time inactive, it is important to consider power consumption in sleep modes as well as active power consumption. Many designers use a power budget methodology to determine the average power consumption and to calculate the battery requirements. Although a great attention is paid to active power consumption, the level of importance to be attributed to a given power saving mode depends on the time spent with the microcontroller in that operating mode and on the duty cycle between them and active mode. For example, in applications like thermostats, keyless entry, or security systems the processor spends most of its time in an idle state. In these applications, sleep mode may have the greatest influence on the overall power consumption, and it will be the most important parameter to be considered.

EXECUTION OF PID

According to the previous considerations, the PID controller has to be implemented differently from the methodology outlined in the previous chapters making the controller run intermittently. Suppose that the control system starts with the microcontroller in running mode and executing some control steps: whenever the controlled variable reaches steady state, it is apparent that control intervention is no more required thus, the execution of the control algorithm is stopped and the microcontroller is, possibly, put in a deep sleep mode. However, some parts are kept active and monitoring the process' output variable in order to wake up the control system whenever is necessary.

For the simulation of this technique, the same process structure and the same controller and process C++ class' structures of Chapter 3 were used.

Overall system UML

The UML diagram for the overall system according to the implementation presented in this chapter, is shown in figure 28. Comparing this diagram with the one of Figure 23, emerges that, here, the function of the PID class computing a new control action is no more called directly from the main class block, but from within one of the sampling algorithms. This makes the complete program computing a new control action only when the process' output diverges from its steady-state value for more than a prescribed amount, as detected by one of the sampling algorithms presented in the previous chapters.

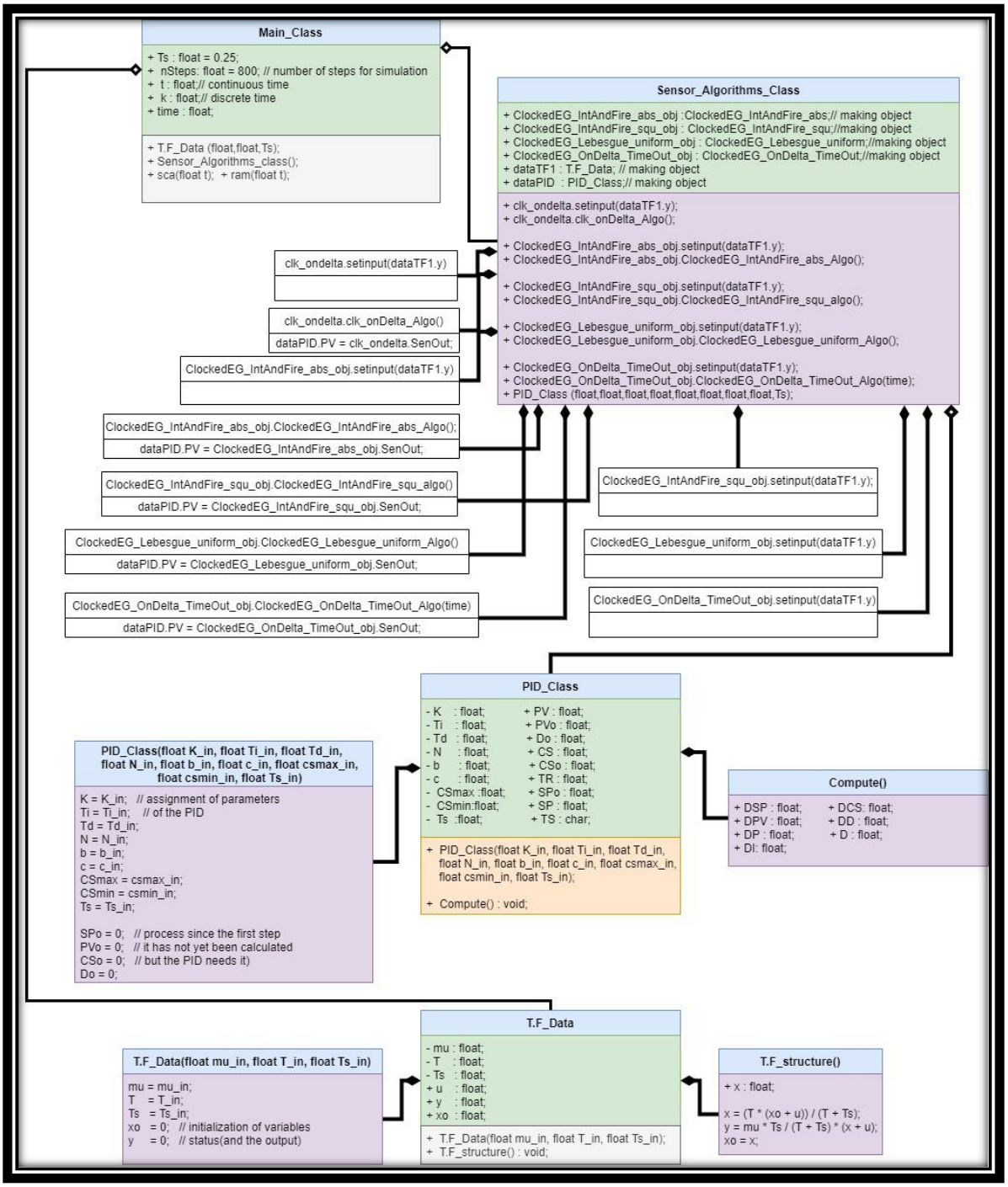

Figure 28 Complete System UML

Code Structure

The source code has been structured in a way similar to the one described in Chapter 3, to which the interested reader is addressed.

Chapter 5

Results

INTRODUCTION

In this chapter the results coming from the application of each of the sampling approaches described in Chapter 2, with both the control structures of Chapters 3 and 4, are presented and discussed. Finally, a comparison between the two control approaches is made, drawing conclusions about which of the two is the best in terms of energy savings and response time.

CLK_ON_DELTA USING PID

In this case the sampling method used is the "send on delta" one, applied to the always-active PID control structure. Blue line is the set-point, while the red one is the process' output variable.

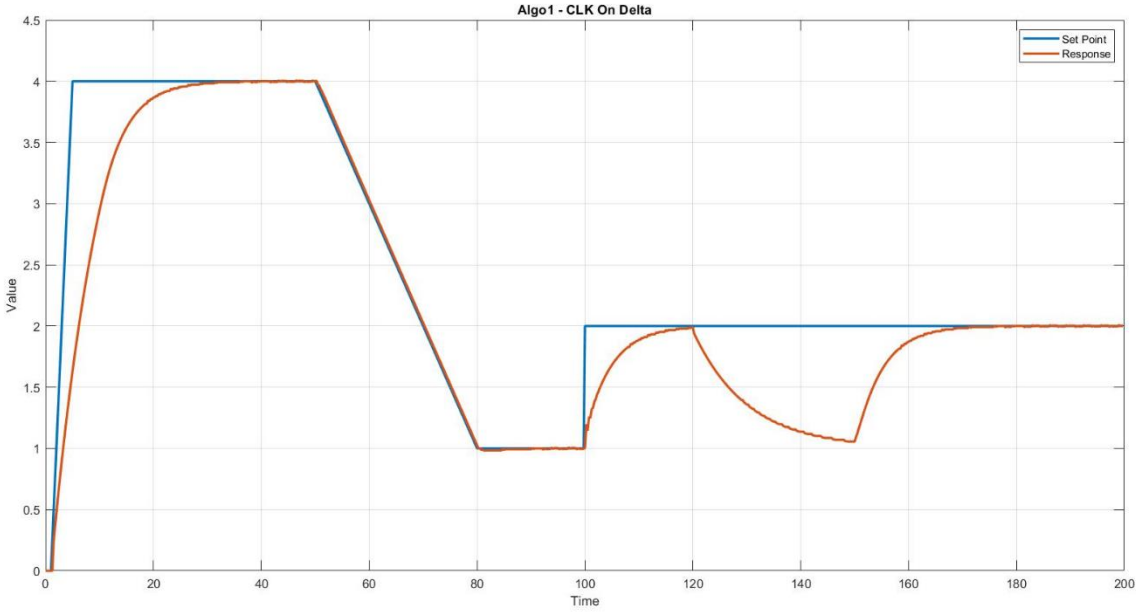

Figure 29 algo1 response by PID

CLK_INTANDFIRE_ABS USING PID

In this case the sampling method used is the "integrate and fire" one where the integral is computed on the absolute value of the error, coupled with the always-active PID control structure.

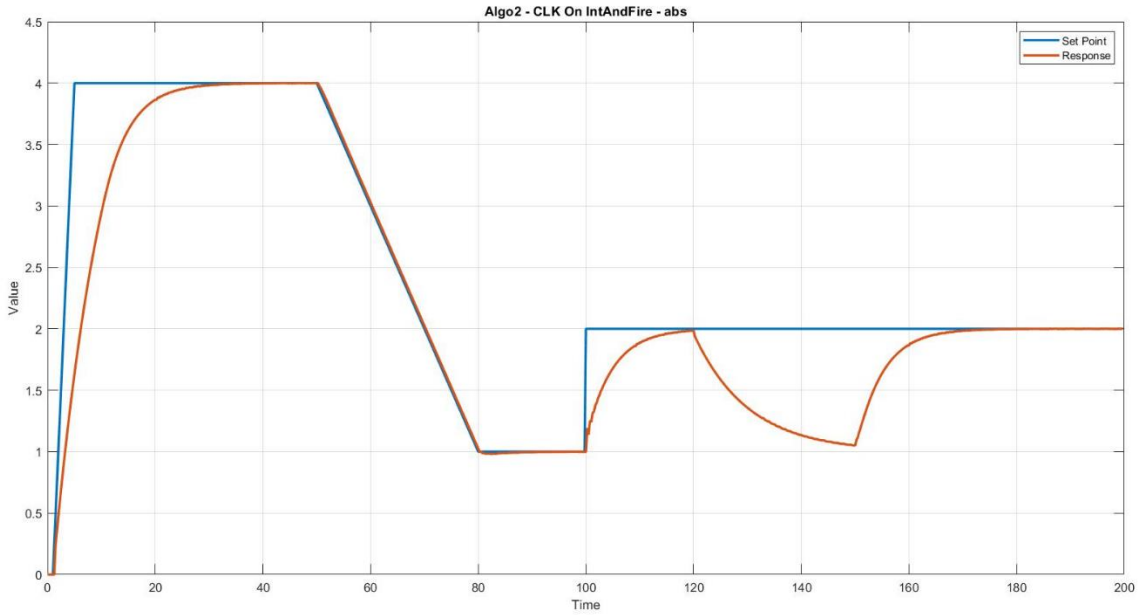

Figure 30 algo2 response by PID

CLK_INTANDFIRE_SQU USING PID

In this case the sampling method used is the "integrate and fire" one where the integral is computed on the squared value of the error, coupled with the always-active PID control structure.

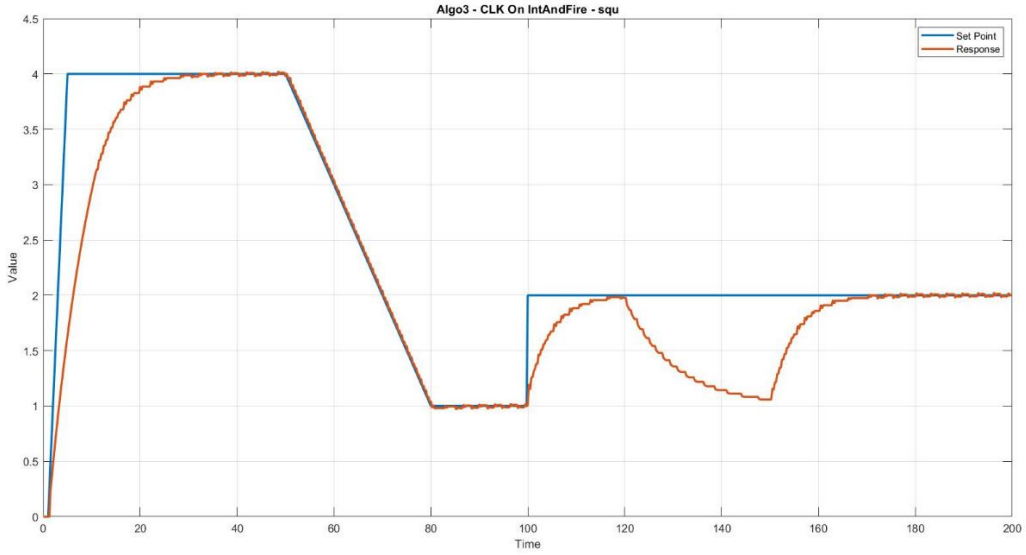

Figure 31 algo3 response by PID

CLK_DELTA_TIMEOUT USING PID

Here the "send on delta" method is used, extended with the timeout mechanism and with the always-active PID control structure.

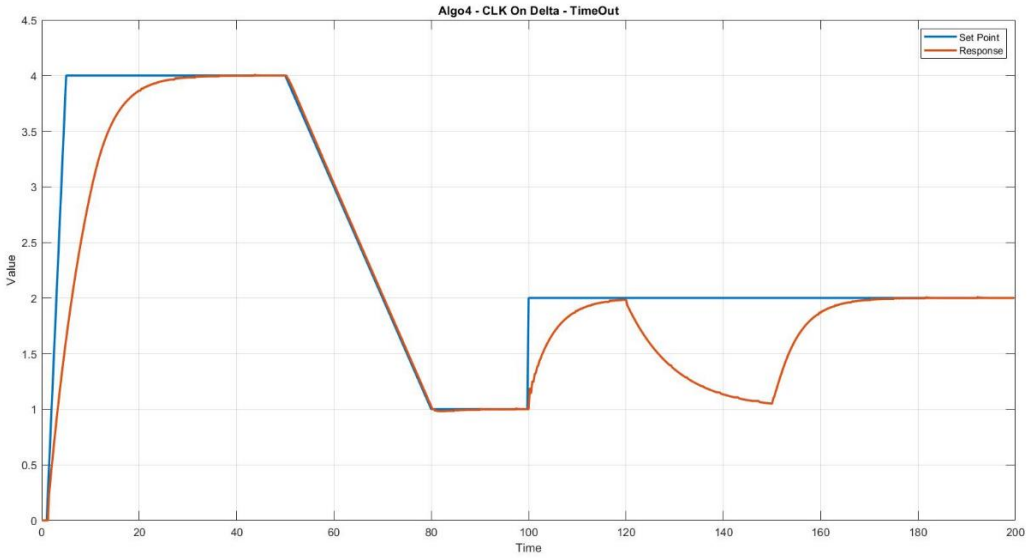

Figure 32 algo4 response by PID

CLK_LEBESGUE_UNIFORM USING PID

Here the "Lebesgue sampling" method is used, together with the always-active PID control structure.

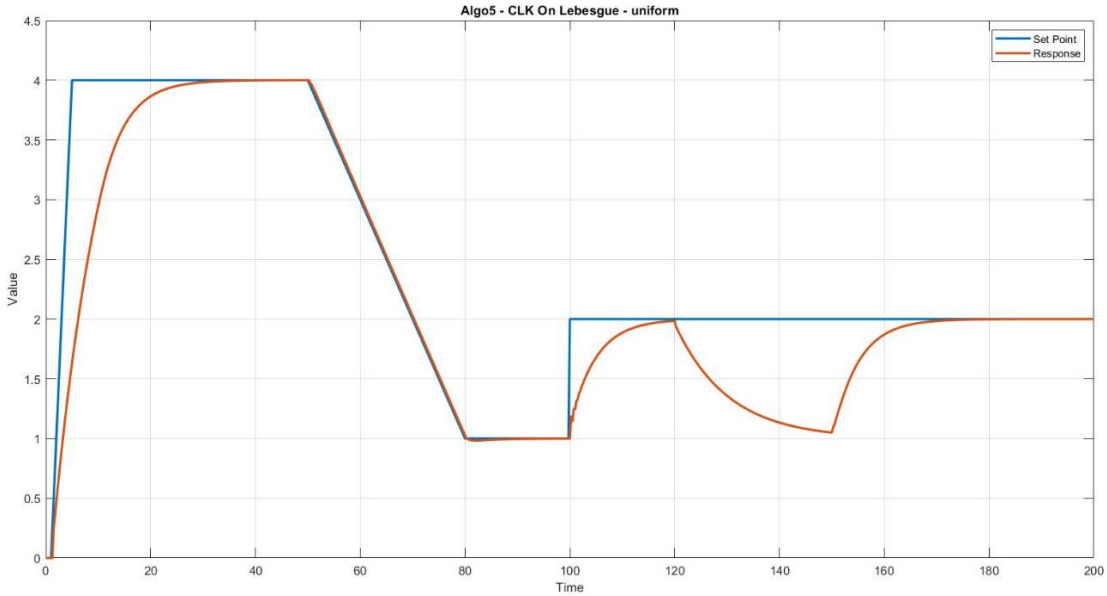

Figure 33 algo5 response by PID

OBSERVATION

Analyzing the time responses, we can conclude that the PID control algorithm is correct: the profile of the response is smooth and the settling time small. The behavior of the second form of the control algorithm, the one described in Chapter 4, is now studied keeping the same process structure of the tests shown before and the same sampling techniques.

CLK_ON_DELTA USING e-PID

Here the sampling method is the "send on delta one" and the control algorithm is executed only when an event is triggered.

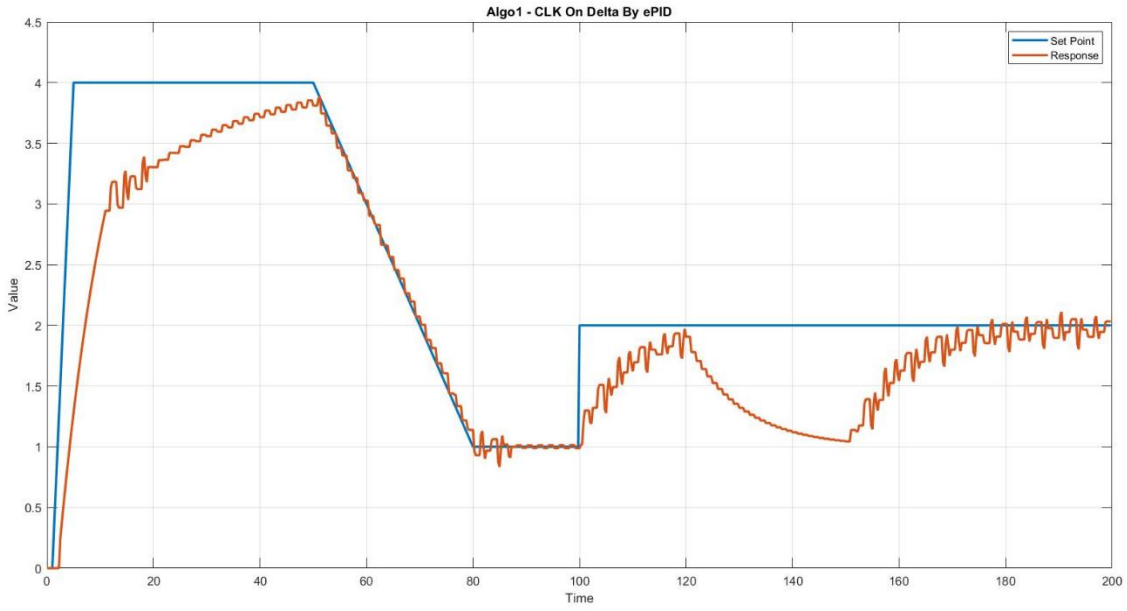

Figure 34 algo1 response with e-PID

With respect to the previous case, here the closed-loop system's response is not very satisfactory. The controller tries to reach the set-point, but there is some oscillatory behavior of the controlled variable and a greater settling time.

From this emerges that the tuning of the regulator has to be slightly modified in order to accommodate for the differences in its execution. The re-tuned controller, then, has been tested again with the results shown in Figure 38. In this case, the controller was effectively active for 462-time instants, while its fixed-step counterpart remained active for 800-time instants.

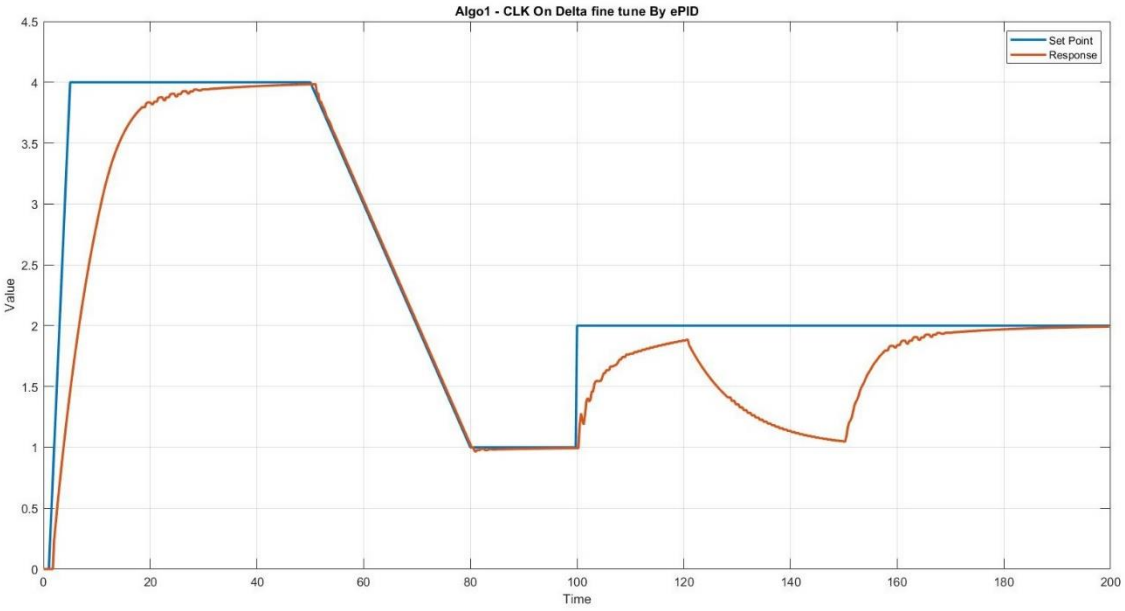

Figure 35 algo1 fine-tuned response by e-PID

CLK_INTANDFIRE_ABS USING e-PID

Here the sampling method is the "integrate and fire" with the integral computed on the absolute value of the error and the control algorithm is executed only when an event is triggered.

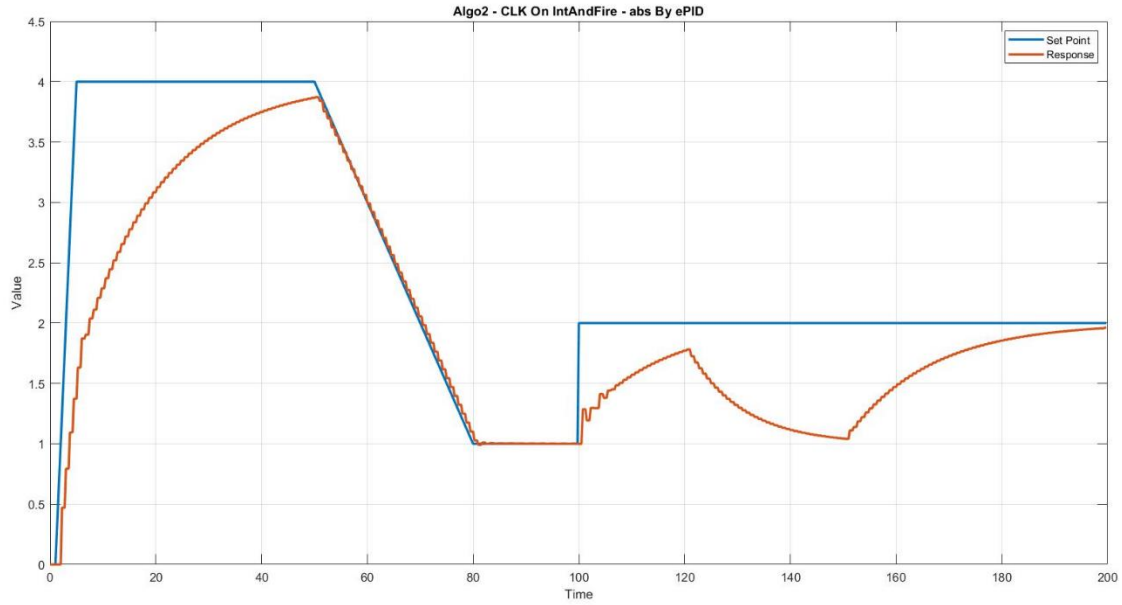

Figure 36 algo2 response by e-PID

As in the previous case, the on-demand execution of the controller lead to oscillatory response and an enlarged response time. After a proper re-tuning, whose results are shown in Figure 40, the

behavior of the closed-loop system is satisfactory. In this case, the controller code was executed for 548-time instants, with respect to the 800 taken by the fixed-rate implementation.

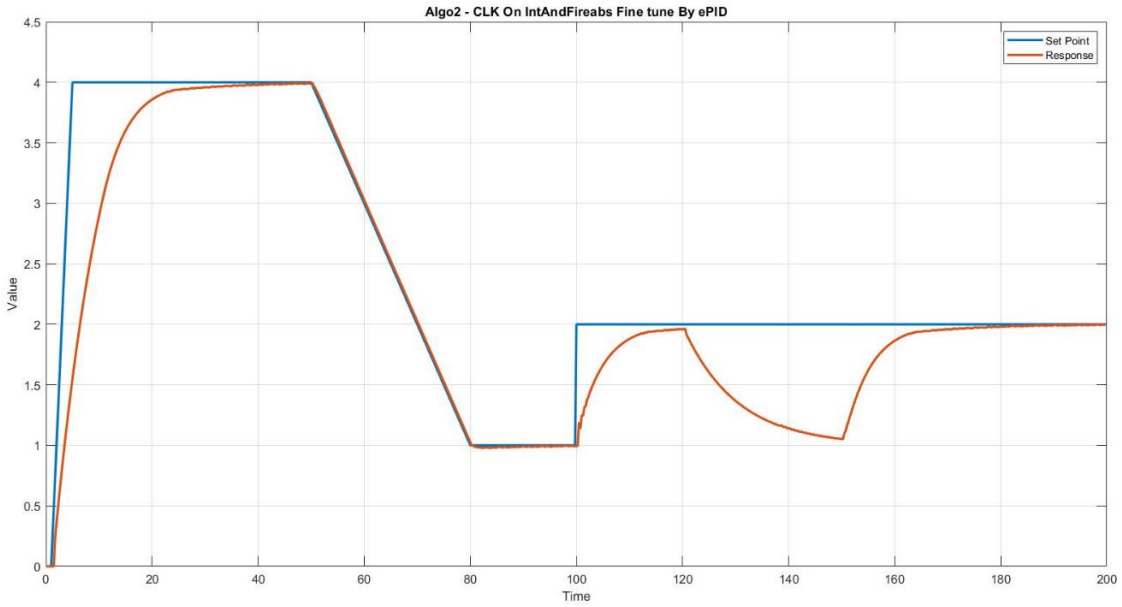

Figure 37 algo2 fine-tuned response by e-PID

CLK_INTANDFIRE_SQU USING e-PID

Here the sampling method is the "integrate and fire" with the integral computed on the squared value of the error and the control algorithm is executed only when an event is triggered.

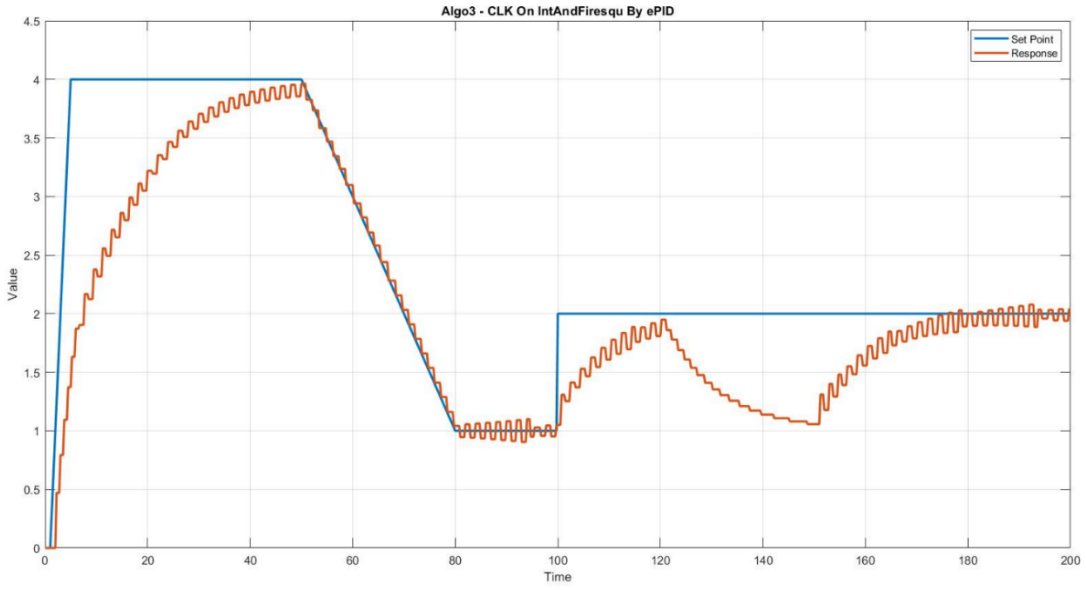

Figure 38 algo3 response by e-PID

As in the previous case, the on-demand execution of the controller lead to oscillatory response and an enlarged response time. After a proper re-tuning, whose results are shown in Figure 42, the behavior of the closed-loop system is satisfactory. In this case, the controller code was executed for 472-time instants, with respect to the 800 taken by the fixed-rate implementation.

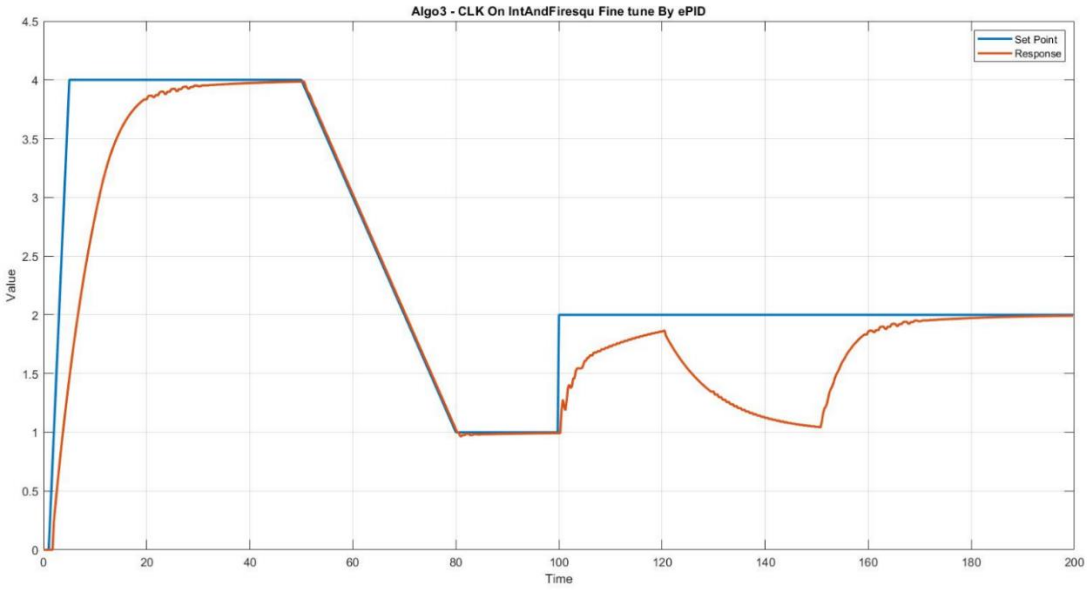

Figure 39 algo3 fine-tuned response by e-PID

CLK_DELTA_TIMEOUT USING e-PID

Here the sampling method used is the "send on delta" complemented with a timeout mechanism and the control algorithm is executed only when an event is triggered.

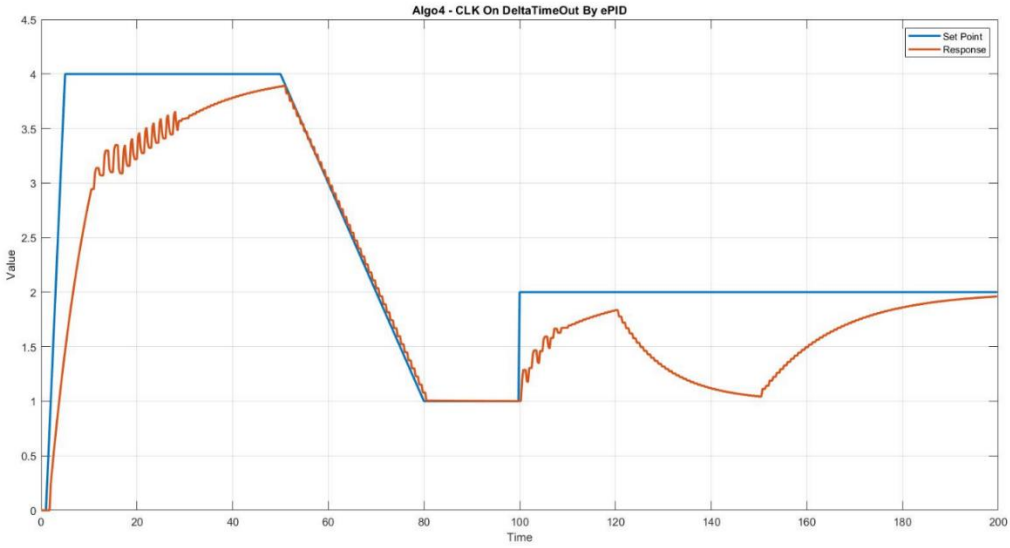

Figure 40 algo4 response by e-PID

As in the previous case, the on-demand execution of the controller lead to oscillatory response and an enlarged response time. After a proper re-tuning, whose results are shown in Figure 44, the behavior of the closed-loop system is satisfactory. In this case, the controller code was executed for 462-time instants, with respect to the 800 taken by the fixed-rate implementation.

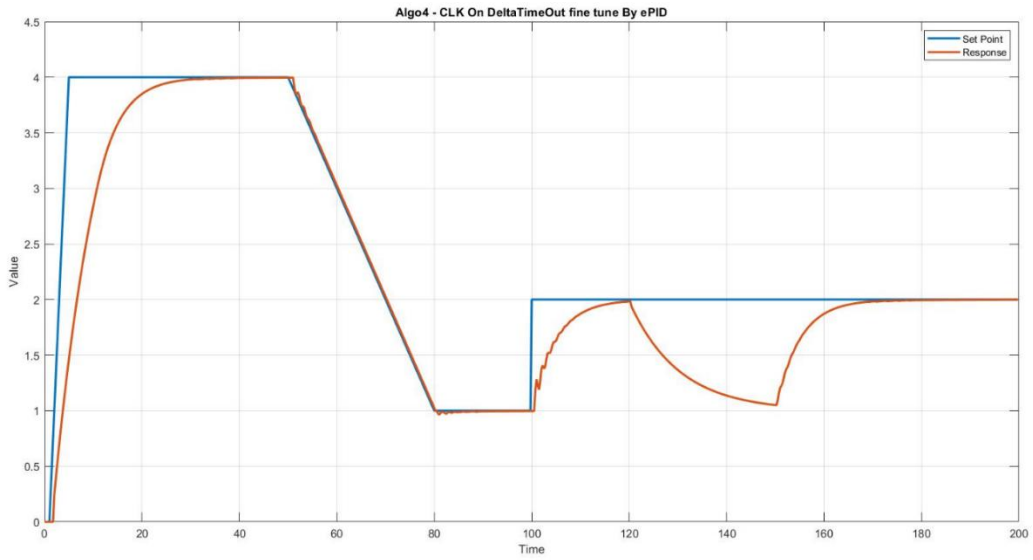

Figure 41 algo4 fine-tuned response by e-PID

CLK_LEBESGUE_UNIFORM USING e-PID

In this case the "Lebesgue sampling" method is used, and the control algorithm is executed only when an event is triggered.

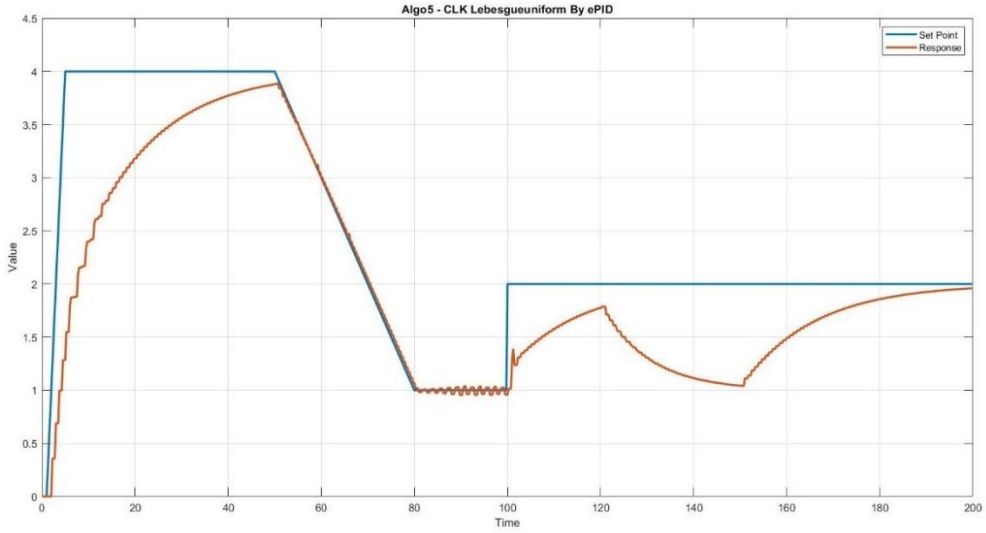

Figure 42 algo5 response by e-PID

Also, in this case the system has an enlarged response time, however the oscillatory behavior is reduced. After a proper re-tuning, whose results are shown in Figure 46, the behavior of the closed-loop system is satisfactory. In this case, the controller code was executed for 554-time instants, with respect to the 800 taken by the fixed-rate implementation.

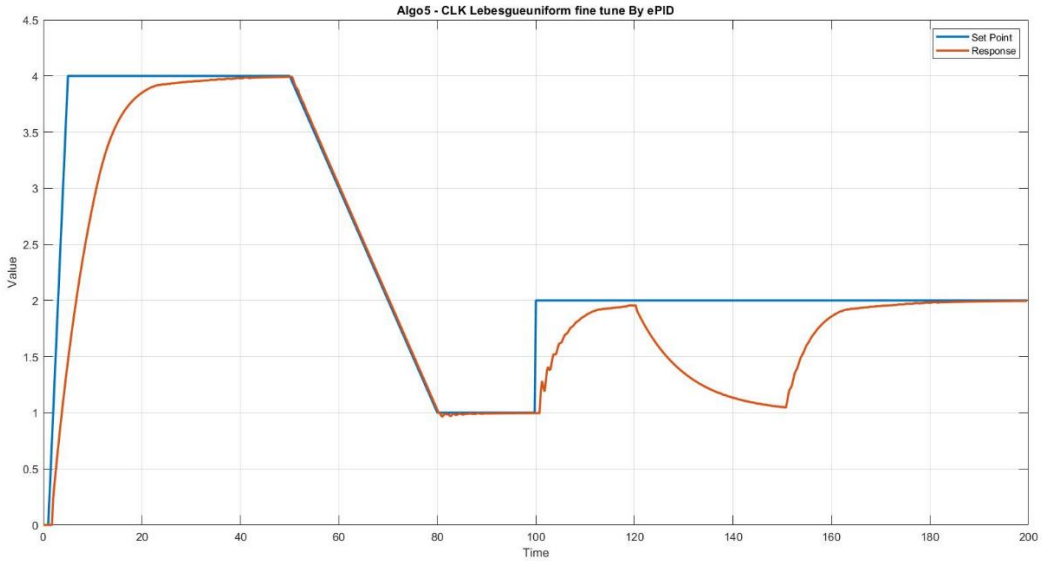

Figure 43 algo5 fine-tuned response by e-PID

OBSERVATION

Analyzing the results, it can be concluded that the on-demand control computation took roughly one half of the time steps when compared to the fixed-step control technique. As pointed out in the previous chapters, this allows for relevant savings both in terms of energy and transmission bandwidth.

COMPARISON

A side-by-side comparison between the two techniques for control computation is now presented with the aim of fining which one performs better in terms of performance and power efficiency.

In all the comparison graphs the blue line represents the set-point, the red dashed one the output response of the fixed-rate closed-loop system, the yellow line the output response of the event-based control system and, finally, the purple one the output response of the event-based control system after the PID controller tuning has been refined.

Clk_on_delta using PID vs e-PID

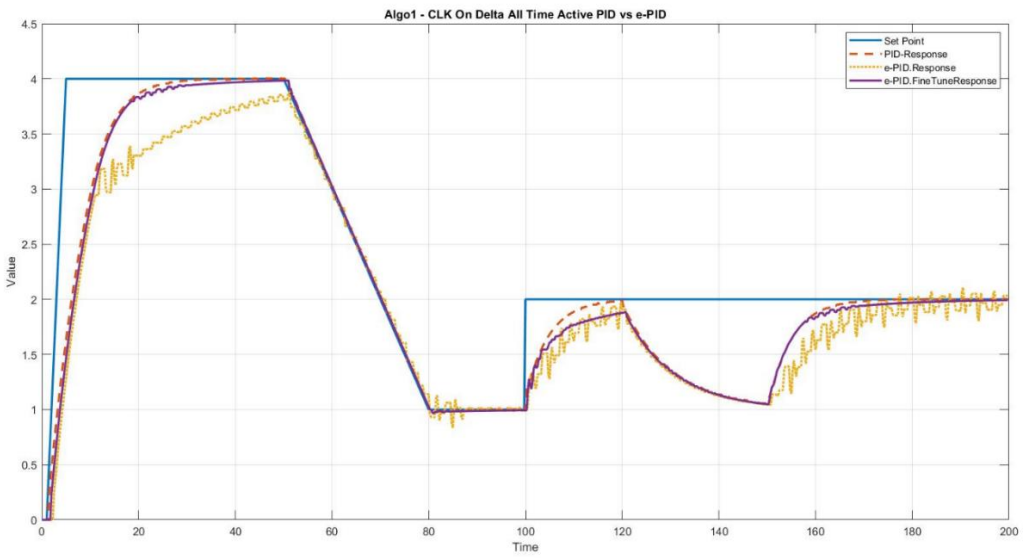

Figure 44 algo1 response comparison PID VS e-PID

In this case the fixed-rate controller gave the best performance. The fine-tuned event-based realization, however, has a response profile very similar but consumes roughly half the CPU time. In a tradeoff perspective among power and performance, this latter technique allows to achieve both a good closed-loop response and important power savings.

Clk_IntANDFire_abs using PID vs e-PID

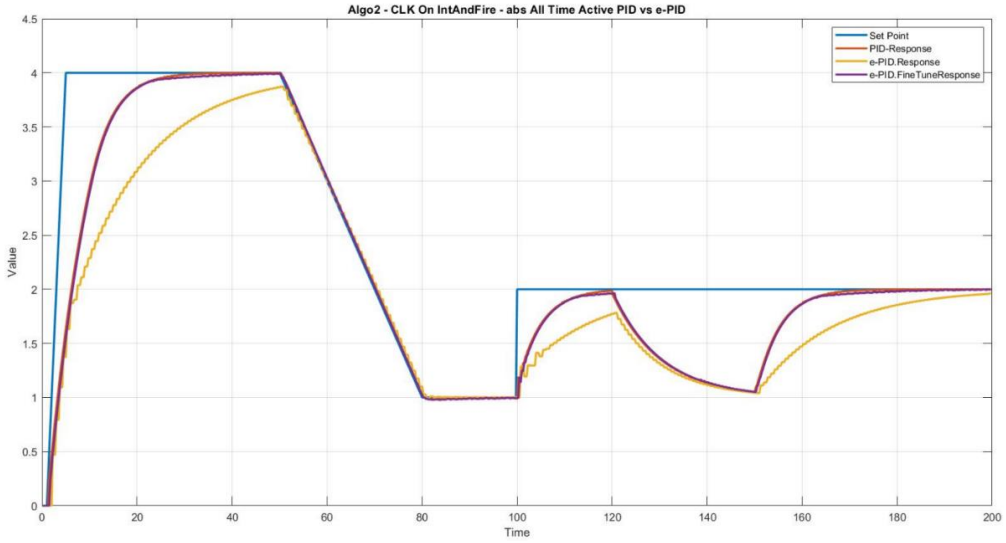

Figure 45 algo2 response comparison PID vs e-PID

In this case the closed-loop response is almost equal for the fixed-rate and event-based realizations. The event-based one, however, requires roughly one half of the time steps used by the fixed-rate one.

Clk_IntANDFire_squ using PID vs e-PID

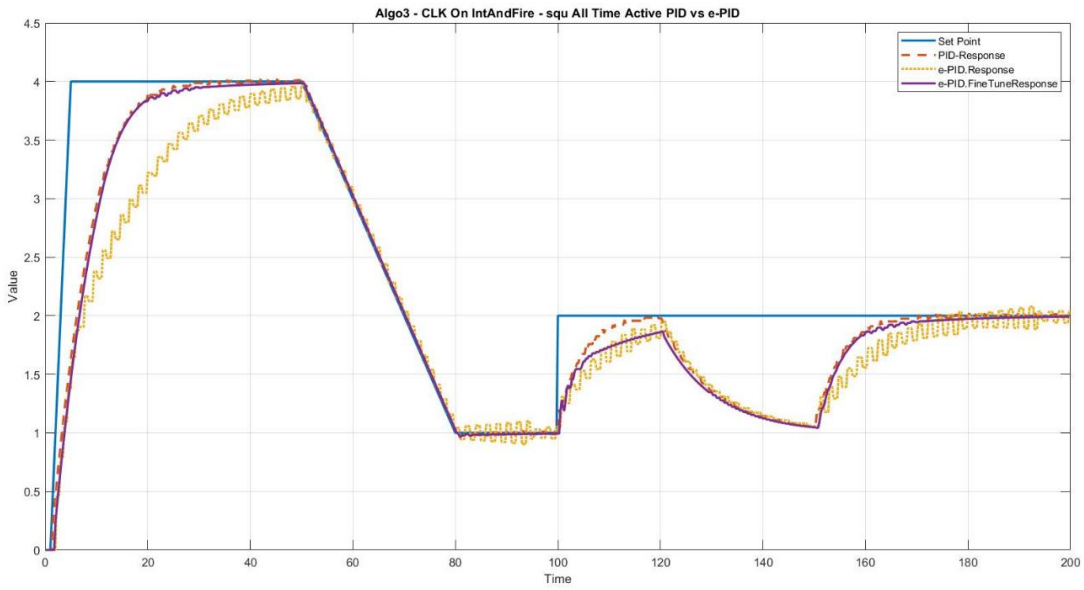

Figure 46 algo3 response comparison PID vs e-PID

In this case the closed-loop response is almost equal for the fixed-rate and event-based realizations. The event-based one, however, requires roughly one half of the time steps used by the fixed-rate one.

Clk_Delta_TimeOut using PID vs e-PID

Also, here the closed-loop response is almost equal for the fixed-rate and event-based realizations. The event-based one, however, requires roughly one half of the time steps used by the fixed-rate one.

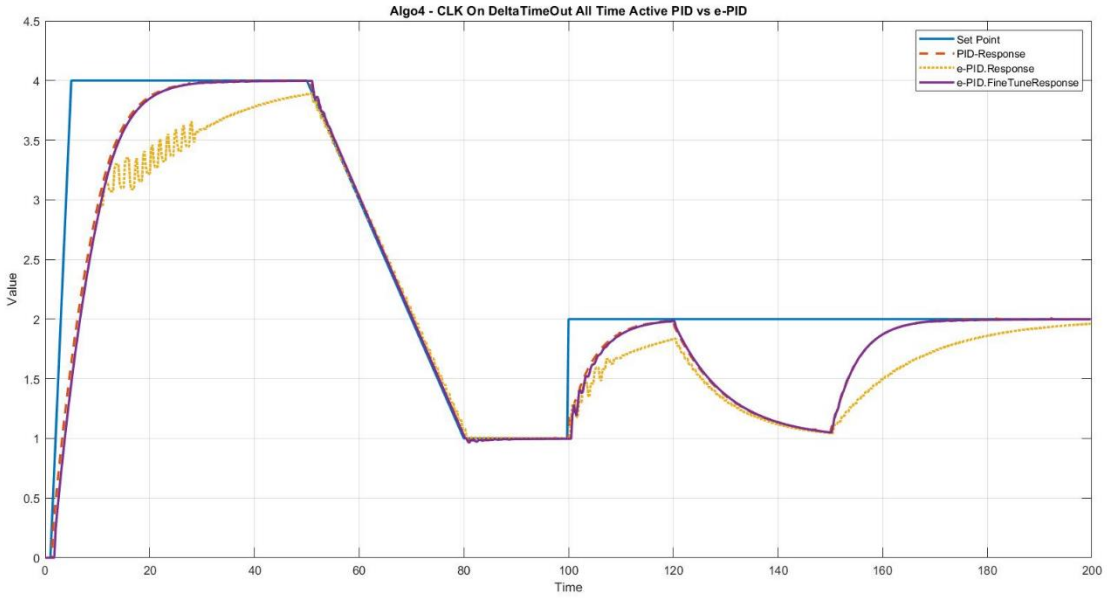

Figure 47 algo4 response comparison PID vs e-PID

Clk_Lebesgue_Uniform using PID vs e-PID

The usage of "Lebesgue sampling" method resulted to have a little impact on the performance of the event-based control algorithm, which gave a closed-loop response comparable to the one of the fixed-step one.

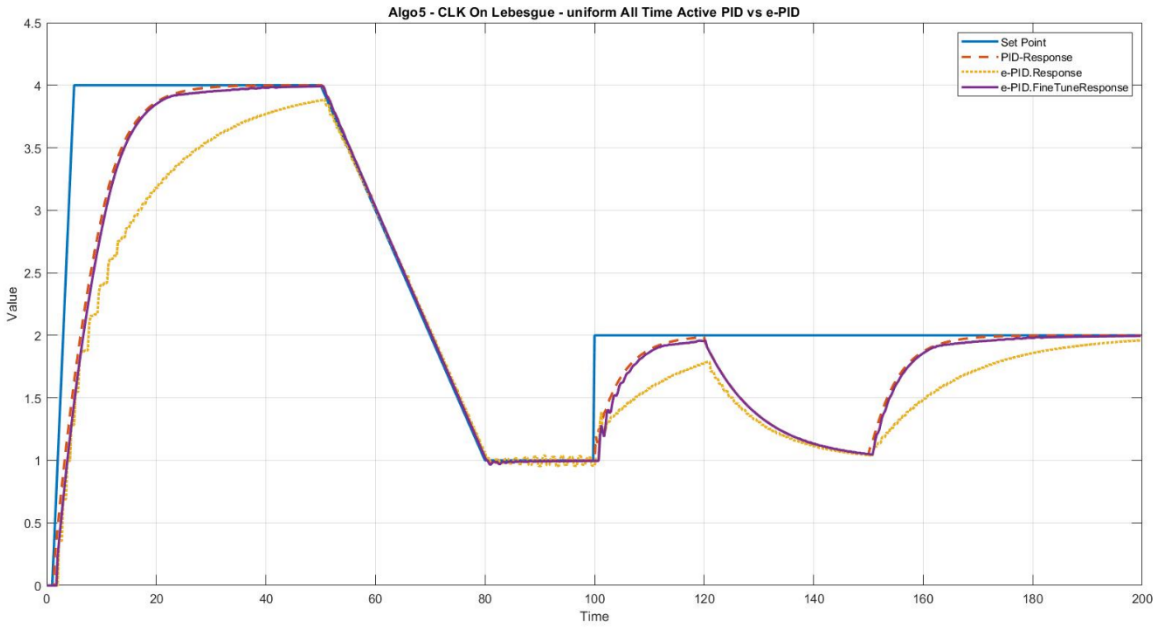

Figure 48 algo5 response comparison PID vs e-PID

Overall Comparison

In all the five cases, the fine-tuned event-based control technique gave results comparable to the ones given by the fixed-rate one, if not equal. The former, however, brings in significant advantages in terms of processor time and, consequently, drawn power.

The overall comparison of each algorithm is shown below.

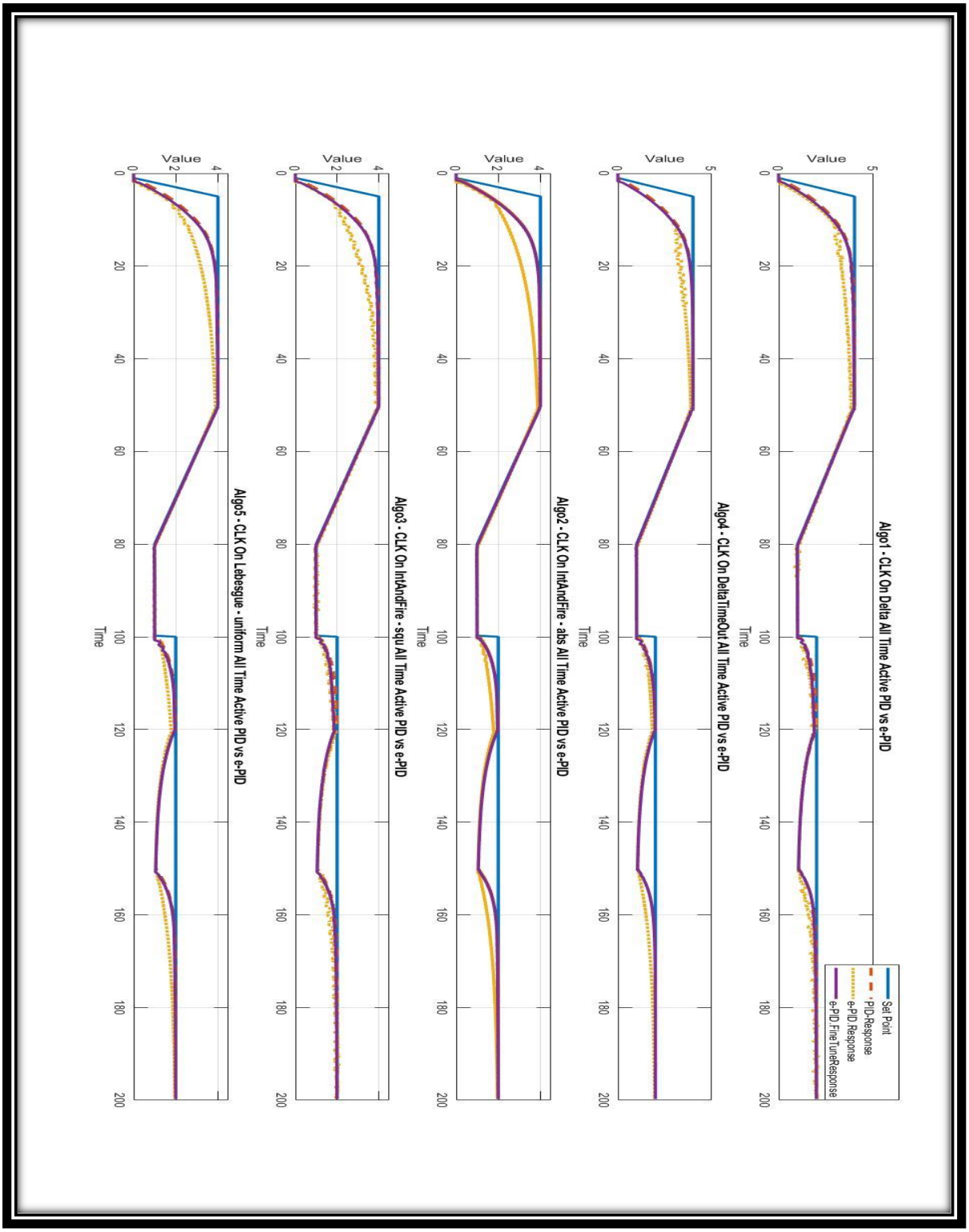

Figure 49 overall comparison b/w PID & e-PID

By observing the responses in the plot, we can say that the response of all-time active PID and e-PID is almost same.

Chapter 6

Conclusion and Future work

INTRODUCTION

In this chapter we obtained results are analyzed and discussed basing on control performance and power consumption, the latter being a key parameter in a low-power environment.

POWER CONSUMPTION

Due to the fact that the power requirements of embedded systems vary depending upon both hardware and software specifications, let us define a general criterion to evaluate power requirements of proposed system with respect to present solutions.

Basing on the assumption that the microcontroller is normally kept in a stand-by or power-off mode and activated only when an event is triggered, we can consider power consumption to be directly proportional to the number of events generated Hence, we can use the number of triggered events as the evaluation parameter when making comparisons.

If we make a power consumption chart according to the value of computation steps taken by each controller, we can easily find the most energy efficient algorithm.

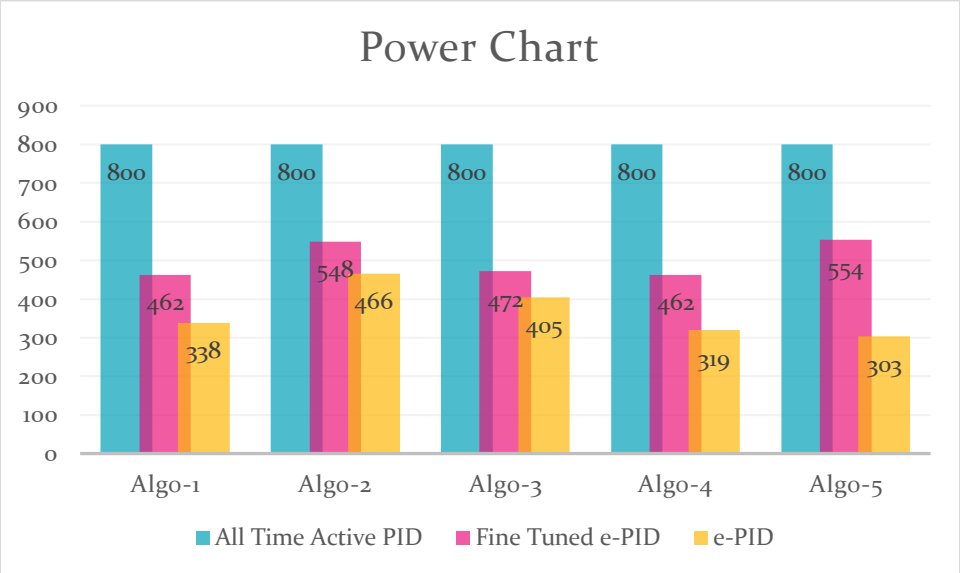

Figure 50 Power chart

According to the results of the previous chapter and to the power chart above, the fixed-rate PID controller has the maximum power consumption, but it also provides the best response. On the other hand, fine-tuned event-based controllers consume less power but provide as well as good

response. Finally, the non-optimized version of the event-based controllers have low power consumption but also bad closed-loop responses.

FINAL REMARKS & FUTURE WORK

In this thesis work, a PID control library has been written following two different approaches, detailed in Chapters 3 and 4. Then, a deep analysis of their behavior has been performed in order to determine which approach among the two gives the better closed-loop response and allows for significant energy savings.

From the simulations emerged that the fixed-rate PID controller gives the best control response but it also implies a huge power consumption by requiring the host microcontroller to be always active. Event-based methods led to significant power savings but come at the cost of a degraded closed-loop response. More in detail, an enlargement of the threshold for event triggering leads to lower power consumption but at the cost of a poor response. Increasing the threshold allows to have better response profiles, however it implies a large power consumption.

For future work one can implement advanced control techniques and analyze their consumed power-vs-response trade-off model. If a good trade-off exists, one can obtain an advanced control system with reduced power consumption.

Bibliography

[1] SEVA, S. (2019). Periodic event-based control with past measurements transmission and multiple control computations. Design, analysis, and FPGA implementation.

[2] El-Sharif, I., Hareb, F., & Zerek, A. (2014). Design of discrete-time PID controller. In International Conference on Control, Engineering & Information Technology (CEIT'14) (pp. 110-115).

[3] Valenti, C. (2004). Implementing a pid controller using a pic18 mcu. Microchip Technology.

[4] Åström, K. J. (2002). Control system design lecture notes for me 155a. Department of Mechanical and Environmental Engineering University of California Santa Barbara, 333.

[5] Holberg, A. M., & Saetre, A. (2006). Innovative techniques for extremely low power consumption with 8-bit microcontrollers. Atmel White Paper.

[6] Scilab.org. 2020. Discrete-Time PID Controller Implementation | Www.Scilab.Org. [online] Available at: <https://www.scilab.org/discrete-time-pid-controller-implementation>.

[7] Shenton, A. T., & Shafiei, Z. (1994). Relative stability for control systems with adjustable parameters. Journal of guidance, control, and dynamics, 17(2), 304-310.

[8] Åström, K. J., & Hägglund, T. (1984). Automatic tuning of simple regulators with specifications on phase and amplitude margins. Automatica, 20(5), 645-651.

[9] Ziegler, J. G., & Nichols, N. B. (1942). Optimum settings for automatic controllers. trans. ASME, 64(11).

[10] Ho, W. K., Hang, C. C., & Zhou, J. (1997). Self-tuning PID control of a plant with under-damped response with specifications on gain and phase margins. IEEE Transactions on Control Systems Technology, 5(4), 446-452.

Index