Universidade de Aveiro
Departamento de Eletrónica,
Telecomunicações e Informática
2016

Pedro Miguel Jesus Gonçalves

Controlador de Tempo-Real baseado em MATLAB e Raspberry Pi

Real-Time controller based on MATLAB and Raspberry Pi

Universidade de Aveiro
2016

Departamento de Eletrónica,
Telecomunicações e Informática

Pedro Miguel Jesus Gonçalves

Controlador de Tempo-Real baseado em MATLAB e Raspberry Pi

Real-Time controller based on MATLAB and Raspberry Pi

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor Alexandre Manuel Moutela Nunes da Mota, Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Paulo Bacelar Reis Pedreiras, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor José Alberto Gouveia Fonseca
Professor Associado da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Frederico Miguel do Céu Marques dos Santos
Professor Adjunto do Instituto Superior de Engenharia de Coimbra

Professor Doutor Paulo Bacelar Reis Pedreiras
Professor Auxiliar da Universidade de Aveiro

agradecimentos /
acknowledgements

Quero agradecer ao meus pais, Albertina Gonçalves e Helder Gonçalves, pela oportunidade concedida para que eu pudesse continuar com os meus estudos.

Agradeço aos meus professores Alexandre Manuel Moutela Nunes da Mota, Paulo Bacelar Reis Pedreiras e Rómulo Antão pela sua disponibilidade, sugestões, bem como todas as discussão pertinentes que contribuiram para a realização desta dissertação.

Por fim, quero agradecer ao meus amigos que me acompanham desde o início do percurso académico, pela sua camaradagem e por terem tornado esta experiência mais enriquecedora.

Palavras Chave Controlo, MATLAB, PID, RST, Raspberry Pi, Tempo Real, Xenomai.

Resumo

Para esta dissertação é proposto o desenvolvimento de um controlador de tempo-real baseado na plataforma computacional *Raspberry Pi*, munidos de *ADCs* e *DACs* para interagir com sistemas físicos. Os algoritmos de controlo deverão poder ser desenvolvidos e testados em *MATLAB*, sendo "traduzidos", de forma automática, para linguagem C capaz de ser compilada e executada na plataforma *Raspberry Pi*.

Keywords Control, MATLAB, PID, RST, Raspberry Pi, Real-Time, Xenomai.

Abstract

For this dissertation is proposed the development of a real-time controller based on the computer platform Raspberry Pi, fitted with ADCs and DACs to interact with physical systems. The control algorithms must be able to be developed and tested in MATLAB, then "translated" automatically to C programming language to be compiled and run on the Raspberry Pi platform.

Contents

List of Figures

List of Tables

Glossary

ABS	Anti-lock Braking System
ADC	Analog to Digital Converter
ADEOS	Adaptive Domain Environment for Operating Systems
API	Application Programming Interface
ARM	Advanced RISC Machine
ARX	AutoRegressive with eXogenous terms
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
CS	Chip Select
DAC	Digital to Analog Converter
DAQ	Data AcQuisition
DETI	Departamento de Eletrónica, Telecomunicações e Informática
DI	Digital Input
DO	Digital Output
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GPIO	General-Purose Input/Output
GPI	General Purpose Input
GPO	General Purpose Output
GPU	Graphics Processing Unit
HAL	Hardware Abstract Layer
I-PD	Integral - Proportional Derivative
I/O	Input/Output
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IP	Internet Protocol
IRQ	Interrupt ReQuest
JFET	junction gate Field-Effect Transistor
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
LAN	Local Area Network
LSB	Least Significant Bit
LS	Least Squares
MATLAB	MATrix LABoratory
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
OS	Operative System
PCB	Printed Circuit Board
PC	Personal Computer
PDIP	Plastic Dual-In-line Package
PID	Proportional Integral Derivative
PI	Proportional Integral
PWM	Pulse-Width Modulation
P	Proportional
RAM	Random-Access Memory
RLS	Recursive Least Square
RST	Reference Signal Tracking
RTDM	Real-Time Driver Model
RTOS	Real-Time Operative System
RT	Real-Time
SAH	Sample And Hold
SCLK	Signal Clock
SCL	Serial Clock
SDA	SDASerial Data

SOC	System On Chip	**UART**	Universal Asynchronous Receiver/Transmitter
SPI	Serial Peripheral Interface	**UDP**	User Datagram Protocol
TCP	Transmission Control Protocol	**USB**	Universal Serial Bus

CHAPTER 1

INTRODUCTION

Developing a Real-Time (RT) control system is a challenge that involves the production of software that not only has to meet functional requirements, but must also meet the non functional requirements, in particular in the temporal level. There are several solutions that are able to meet this requirements based on RT Linux. Related to function levels it is known that MATrix LABoratory (MATLAB) provides a wide range of solutions, and therefore will be an integral part of the project.

1.1 MOTIVATION

Control Systems are all around. There are many examples that we use daily and we are not always aware, such as the air conditioning, the car's Anti-lock Braking System (ABS) or ourselves. Our body temperature regulation is an example of control system that I will enter in more detail. Our body regulates its internal temperature to be around 37º. If the temperature goes up, the body will act to lower the temperature, using vasodilation or sweating. In case of the temperature goes down, the body will start to do vasoconstriction, piloerection or shivering to rise the temperature [1].

Nowadays, with the price of digital computers dropping, most controllers start to be implemented in software. The price is not the only reason to do the implementation. Flexibility, adaptable controllers and error detection are other reasons to use digital controllers. With tools available like MATLAB, it is possible to test and to tune complex and expensive systems without the risk of damaging it. After having the controller developed, would be great to validate it with the physical system and test it without leaving MATLAB. Digital controller theory is based on periodicity of sampling and actuation on system. For this, it is mandatory to have real-time systems that ensure that the time requirements are met. Therefore, it is necessary to have the control loop running in a real-time embedded system. Having a minimal effort of code adaptation would help to focus in the development of system and lead to a better comprehension of the control subject.

1.2 GOALS

In this dissertation it is intended to develop a digital and analog interface for MATLAB. This interface is to be used in an educational environment to let students improve their knowledge in control systems. Making the bridge between an informatic system and the physical world is not the only intent of this system. The developed interface has to be able to work on its own, and to execute the algorithm that would be running on MATLAB.

To accomplish the goals, a set number of steps need to be accomplished:

- Preliminary study of the problem;
- Development of interfaces ADC and DAC for the Raspberry Pi.
- Integration of a RT kernel on the Raspberry Pi.
- Development of a RT manager for Raspberry Pi.
- Development of control algorithms and translation process from MATLAB to C programming language.
- Development of a demonstrator.
- Testing and Validation.

1.3 DOCUMENT STRUCTURE

The structure of this document is:

- **Chapter 1**: presents the motivation and goals of this thesis as well this document structure.
- **Chapter 2**: presents the tools available at the market and the fundamental concepts required to the devolpment of this work.
- **Chapter 3**: presents the development of the hardware and the requirements to be meet.
- **Chapter 4**: presents the configurations made to the hardware, the protocol created for real-time communications and the program to translate MATLAB code to Raspberry Pi.
- **Chapter 5**: presents the tests made to validate the entire system, specifically testing control algorithms and perform electrical and time tests for each part of the developed system.
- **Chapter 6**: presents an analysis of the work performed and it is given suggestions for future work.
- **Appendix**: with various information about the protocol, a guide to install Xenomai and results of tests made.

CHAPTER 2

Background

This chapter presents general concepts about real-time, control systems and serial protocols used during the development of the work. It also presents a survey of products already on market with similar functionality respecting the work herein presented.

2.1 MATLAB

MATLAB was created in the early 1970s by Cleve Molar. It was initially written in FORTRAN with the aim to compute the eigenvalues of matrices and solve systems of linear equations. It had only one data type, matrix of complex doubles, and a collection of 80 functions.

Jack Little saw the potential of MATLAB in fields like signal processing and control, and with Steve Bangert they developed MATLAB for Personal Computer (PC) by porting the Molar's code to C. They also add the possibility to create user functions, which made MATLAB a programing language rather then a calculator. In 1984, Mathworks was founded and PC MATLAB was its first product [2].

From the beginning until today, MATLAB has increase its scope, adding capabilities in areas such as optimization, signal and image processing, fuzzy logic, splines, wavelets, statistics, partial differential equations, bioinformatics, mathematical finance and control [2].

2.1.1 DATA ACQUISITION WITH MATLAB

MATLAB is optimized for solving engineering and scientific problems. In this platform is possible to simulate systems and test control algorithms. It is not new the idea of using MATLAB to interact with physical systems, as there are already some tools available in the market for this propose, which are briefly presented in the following sections.

LABJACK CORPORATION

Labjack Corporation makes Data AcQuisition (DAQ) hardware and software to connect the physical world to computers and the Internet [3]. The company has a wide range of hardware with ADCs from 12 to 24 bits of resolution, wide inputs range and selectable gains. Still in the analog domain, they have DACs with different resolutions of 10 or 12 bits, going from 0V to 5V. Other functionalities include multiple digital Input/Output (I/O), counters, timers with PWM output and quadrature inputs. Their products are compatible with multiple environments such as: Laboratory Virtual Instrument Engineering Workbench (LabVIEW), MATLAB, C and Python. More detail about the hardware can be found on Table 2.1.

The series T7 supports scripting in LUA, a programming language designed for embedded systems. "A lua script can be used to collect data without a host computer or to perform complex tasks producing simple results that a host can read." [4] and "While running a Lua script, the T7 and T7-Pro can operate without computer involvement. Basically, user-specified operations (feedback loops, logging, PID loops) can be conducted via on-board script" [5] (from their website).

	U12	U3-LV	U3-HV	U6	U6-Pro	UE9	UE9-Pro	T7	T7-Pro
Analog Input Voltage	±10V	0-2.4V	-10V to 20V	±10V	±10V	±5V	±5V	±10V	±10V
Analog Inputs	8	16	16	14	14	14	14	14	14
ADC Effective Resolution	12 bits	12 bits	12 bits	16 bits	22 bits	12 bits	20 bits	16 bits	22 bits
Analog Output Voltage	0-5V	0-5V	0-5V	0-5V	0-5V	0-5V	0-5V	0-5V	0-5V
Analog Outputs	2	2	2	2	2	2	2	2	2
DAC Resolution	10 bits	10 bits	10 bits	12 bits	12 bits	12 bits	12 bits	12 bits	12 bits
Digital I/O	20	20	16	20	20	23	23	23	23
Logical Level	5V	3.3V	3.3V	3.3V	3.3V	3.3V	3.3V	3.3V	3.3V
Counters	1	Up to 2	Up to 2	Up to 2	Up to 2	Up to 2	Up to 2	Up to 10	Up to 10
USB	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Internal Temp Sensor	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ethernet	No	No	No	No	No	Yes	Yes	Yes	Yes
Wireless	No	No	No	No	No	No	No	No	Yes
Modbus TCP	No	No	No	No	No	Yes	Yes	Yes	Yes
Scripting (Lua)	No	No	No	No	No	No	No	Yes	Yes
MATLAB Compatible	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Price	$139	$108	$114	$299	$369	$479	$579	$399	$499

Table 2.1: LabJack Product Comparison

NATIONAL INSTRUMENTS

NI has multiple products to perform data acquisition, embedded control and systems monitoring. Their products are available in different form factors, from single board to reconfigurable platforms, with extend I/O options.

Focusing in single board solutions, one of the many available is sbRIO-9627, a dual core solution with a Field Programmable Gate Array (FPGA). It has four 3.3V digital channels, sixteen channels with 16bits ADC with multiple ranges capable of 200k samples per second and four DAC channels with 16 bits of resolution working from -10V to 10V. This product runs a NI Linux RTOS programmable with LabVIEW or C. Price was not stated. The board myRIO-1950, follows the same concept, a dual

core with FPGA. Multiple digital I/O with 8 ADC channels, 4 DAC channels, but with more modest analog characteristics. The target price is $901 but for academic use it costs $400.

USB-6001 is a low cost multifunction DAQ, with eight analog inputs of 14 bits of resolution capable of doing 20k samples per second. It features also two analog output channels with 14 bits of resolution and 5k samples per channel per second. Both analog ranges can go from -10V to 10V. It also has 13 multipurpose digital I/O ports. It is compatible with C, LabVIEW and the newer versions of MATLAB. It costs $189. USB-6211 and PCI-6010 have the same philosophy. They are multifunction DAQ with better specifications.

More detailed information about these five products, can be found on Table 2.2.

	USB-6001	USB-6211	PCI-6010	sbRIO-9627	myRIO-1950
Analog Input Voltage	±10V	±10V	±10V	±10V	0 to 5V
Analog Inputs	8	16	16	16	8
ADC Resolution	14 bits	16 bits	16 bits	16 bits	12 bits
Analog Output Voltage	±10V	±10V	±10V	±10V	0 to 5V
Analog Outputs	2	2	2	4	4
DAC Resolution	14 bits	16 bits	16 bits	16 bits	12 bits
Digital I/O	13	8	24	96	32
Logical Level	3.3V	5V	5V	3.3V	5V
Counters	1	2	2	N.A.	N.A.
USB	Yes	Yes	Yes	Yes	Yes
Internal Temp Sensor	No	No	No	No	No
Ethernet	No	No	No	Yes	No
Programmable	No	No	No	Yes	Yes
MATLAB Compatible	Yes	Yes	Yes	No	No
Price	$189	$808	$677	N.A.	$400[1]or $901

Table 2.2: LabJack Product Comparison

DIGILENT

Digilent, a National Instruments company, has "Analog Discovery 100MSPS USB Oscilloscope & Logic Analyzer"[6] featuring a two channels oscilloscope, a two channels function generator and 16 channels in multiple modes, which can be configured as digital I/O, logical analyzer or pattern generator. Other functionalities include digital bus analyzer (SPI, I2C...), voltmeter and power supply. Currently is supported by MATLAB and communicates using Universal Serial Bus (USB). This product cost $259.

[1]price only for academic use

MATHWORKS

Mathworks supports a varied number of systems, like Beagleboard, Raspberry Pi and Arduino to work as an interface to the physical world. For the above systems, MATLAB provides a set number of commands to interface with the peripherals I2C, SPI, GPIO [7] [8] [9] and much others supported by the platforms. For Beagleboard and Raspberry Pi it is also possible to have access to Linux system shell, which allows to accomplish more powerful functions with more control. In the several examples that are possible to find on their website, it is demonstrated how the system can be used for analog data acquisition. In the case of the Raspberry Pi [10], it is used an external 10bits ADC with eight channels, MCP3008 [11], for Beagleboard [12] and Arduino [13] it is used the internal ADC.

2.1.2 CODE GENERATOR FOR EMBEDDED SYSTEMS

This platform allows simulating systems and test control algorithms. After performing all tests and arrive to a suitable control algorithm, it would be useful to have a tool to generate automatically the code to an embedded system. This task can be done with the MATLAB Coder [14], a commercial software from Mathworks available to MATLAB. This tool converts code from MATLAB language to readable C code to be integrated as source code on projects.

2.1.3 COMPARING SOLUTIONS

The commercial offer of DAQ systems is abundant. Selecting U12 from LabJack or USB-6001 from National Instruments would be a good choice for the price range. Both can handle a wide input voltage in the analog side, with eight channels available. In the analog output the USB-6001 has a wider output voltage than the U12, but both have two channels. In the digital domain, they have more than 10 I/O channels 5V compatible. For $189, USB-6001 is a good choice, with built-in support to MATLAB, and for $139 the U12, with support given by LabJack.

However, these boards do not perform so well for closed-loop control. The U12, takes 20ms [15] to execute and respond to a command so it is not suitable for fast controller loops. For better performance, the U6 would be a faster alternative, doing in the same request command four ADC reads and two DAC writes in 8.28ms [16], but it costs $299. The same analysis can not be performed on USB-6001 as there is no data available in National instrument website. In any case, these tools are not suitable for our use, as it is required that they work independently, without a PC. The only tool present here that is capable to work as a self contained controller running LUA scripting and work as a MATLAB digital and analog interface is T7 from LabJack. In average it is capable to read four Analog to Digital Converters (ADCs) and write on two Digital to Analog Converters (DACs) in 5ms [17], but it sells for $399.

Therefore, it is advantageous to develop our own hardware and software. The first advantage is economical: it is possible to build a less expensive tool, that can work in both modes of operation. The second is the possibility to add more functionalities in software without having to buy a new equipment. And last, as the equipment is to be used in a academic environment, the probability of it being misused or damaged is high; so having the schematic and using more general purpose ICs is a great help to troubleshooting and repair.

2.2 REAL-TIME OPERATIVE SYSTEM

There are several types of Operative Systems (OSs); those of general purpose, which are design to have the best average throughput, such as Microsoft Windows, and Real-Time Operative Systems (RTOSs), that will be covered in this section. A RTOS is a type of computational system where the correct answer is not enough for the correctness of the job, the time to get the answer is equally important. These types of systems are projected to attend events or carry out periodic activities, which have timeliness constraints. For example, to attend events there is the airbag case, where is not possible to know when the system will actuate, but when it is needed the system has to function well and with precise timing. On the other hand, a closed control loop is a type of system that requires periodicity to read the sensors and calculate the signal to the actuators; not actuating on time can lead to an unreliable control. Both systems have time constrains.

When talking about RTOS, it is common to assume that the latency will decrease, but this is not precise. What will decrease is the maximum latency. The goal is to have a predictable and deterministic system.

RT systems can be classified in two types: soft and hard. Soft is when a system usually performs an operation on time, but even after the time limit, often called the deadline, the result of operation maintains some of the utility. However, for hard RT the OS must guarantee in every case that the operations are perform on time. If the deadlines are not respected the operation loses all usefulnesses and may cause catastrophic outcomes.

Modern OSs are capable to handle more than one process at a time, concurrently. This ability is called multitasking, and it is possible thanks to a unit responsible for the scheduling task, the scheduler. It determines the order of each task executed, with the aim to complete the tasks before their deadline. The scheduling algorithm can be classified [18]:

- Preemptive vs non-preemptive: preemptive is when the Central Processing Unit (CPU) is executing a task, and its execution can be interrupted by another task of higher priority. Non-preemptive, is when the task will be running until the end of its completion, without interruption.
- Static vs dynamic priorities: In static, the priority of the task remains constant. In dynamic, the priority can be higher or lower than the initially defined to let the system meet the temporal requirements easier.
- Off-line vs on-line: On-line, the order of execution of the task is decided during the normal system operation; in this systems it possible to start and end tasks at run time. Off-line, when the order of execution of the tasks are defined before the system start to run.

2.2.1 PREEMPT_RT

PREEMPT_RT [19] are a set of patches to the Linux kernel, some of which are already integrated on the OS. Linux is preemptible in user level, which means that a task in this domain will be interrupted by another of higher priority. The goal of these patches is to change the kernel to preemptable. To accomplish this, it must be changed some operating mechanisms; two of them will be presented next. The first is to convert all Interrupt ReQuest (IRQ) to thread interrupts; when it is received a interruption it is performed only the necessary work and it is wake-up the corresponding thread. This

thread is scheduled by the kernel depending on its priority. The second is the replacement of spinlocks in kernel by mutex; spinlock is a busy wait mechanism to access critical sections with little overhead, used on kernel space. On the other hand the mutex will put the thread in sleep until the resource is free again. More changes are applied, but these two are considerer the most important ones.

2.2.2 XENOMAI

Xenomai is a micro-kernel that adds hard RT properties to Linux. The main idea is to have all RT threads executed on the RT kernel and when there is no work to do it is given the opportunity to Linux to run, as if it was a low priority thread.

When the hardware makes an interruption, Xenomai has to be capable to arrive first than the Linux Kernel, or it can be blocked for an unbounded time. To accomplish this is used the Adaptive Domain Environment for Operating Systems (ADEOS) interrupt pipe, to send all hardware interruptions to Xenomai first. If Xenomai has an RT handler to the interrupts, it will execute it, if not it will pass to the Linux kernel.

When a Xenomai thread is created, it usually starts in primary mode, the Xenomai domain. If it invokes a non-RT syscall, the thread will transit to the secondary mode, the Linux domain, continue to run in the real-time services of Linux, maintaining its priority. In this mode, Xenomai benefits from the improvements made by the PREEMPT_RT project to have a lower latency.

2.3 COMMUNICATION PROTOCOLS

2.3.1 ETHERNET

Ethernet is mostly used in Local Area Networks (LANs) because of its big bandwidth, cheap hardware and is supported in most OS. It is one of the link layer protocols in the Internet Protocol suite, which is divided in four layers: the link, the Internet, the transport and the application layer [20]. The Figure 2.1 represents its multiple layers.

- Link: it is the lowest layer specified on the Internet Protocol suite. This layer is responsible to frame the packets to be send to the physical medium. An example is the Ethernet frame specified in IEEE 802.3-2012 [21].
- Internet: in this layer is defined the addressing and routing strategy. It is responsible to move the IP data to the next network until it arrives to the destination.
- Transport: here is implemented the congestion and error control, segmentation and application addressing. Examples of protocols are: TCP and UDP.
- Application: provides applications a way to access the services of the other layers, defining protocols that applications use to exchange data.

The first Ethernet standard was able of transmission rates of 10Mb/s. Since its standardization in 1985 it has evolved to 100Gb/s [21]. Ethernet can operate on multiple physical layers such as coaxial, twisted-pair or optical fiber, depending of the intended speed.

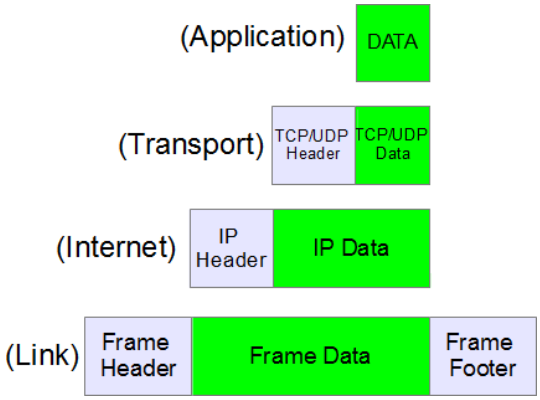


Figure 2.1: Layers on Internet Protocol suite model

TCP

Transmission Control Protocol (TCP) is a connection oriented protocol part of the transport layer. To begin the communication the sender has to perform a three-way handshake. This consists in sending to the server a connection request then the server will answer to the client with a connection request acknowledge and the client will do the last step, sending the final acknowledge. To close a connection the steps are similar.

As TCP is connection oriented it has implemented the following features: TCP ensures that all the packets are delivery to the source. To accomplish this, the packets have to be answered with an acknowledge. If during a predefining amount of time the sender does not receive acknowledge packets, it will assume that the packets failed to arrive to the receiver and it will retransmit them. The protocol delivers the packets to the application in-order, even when they arrive out of order. To accomplish this, the TCP has a 32 bits sequence number in the header. In the header there is also included a 16 bits checksum for error checking. TCP also features a congestion control algorithm that delays the transmission when the network is busy.

UDP

User Datagram Protocol (UDP) is a message oriented lightweight protocol, also part of the transport layer. There is no session, as in TCP, therefore the application can start to talk with the server immediately. For being a lightweight protocol, it does not have the features of the TCP. There is no acknowledge, so the sender does not know if the packets fail to arrive to the destination; and the packets can be delivered to the application out of order. UDP does not have implemented any type of congestion control, so the packets continue to be send even when the network is overloaded. Another propriety of the protocol is that the message boundaries are preserved.

With this saying, UDP is more suitable for applications that requires low latency, and can deal with some packets losses, such as audio and video streaming, because the retransmission mechanism of TCP is temporally unpredictable.

2.3.2 SPI

Serial Peripheral Interface (SPI) [22] is a full-duplex synchronous serial communication interface used for short distance, initially designed by Motorola. There are multiple versions of the protocol, but we will only talk about the four wire: SCLK is the clock generated by the master device; MOSI, master output slave input, is where the master device write the data to the slave; MISO, master input slave output, is where the master reads the data sent by the slave, and finally the CS, chip select, is the port responsible to for selecting the slave to communicate with.

2.3.3 I2C

Inter-Integrated Circuit (I2C) [23] is a half-duplex synchronous serial protocol develop by Philips Semiconductor. The bus is consisting of two channels, the Serial Clock (SCL) and SDA (SDA). SCL is the clock signal line and SDA is the line in which data is transmitted in both directions. This protocol was develop to have multiple slaves in the same bus. When a communications starts, it first sends the slave address followed by a read/write bit.

2.4 CONTROL SYSTEMS

Controller is a system that manipulates one or more elements of another system to set the desired behavior of this last. There is two forms of control: open and closed-loop. The first, open-loop, does not use feedback, which means the output of the system will not have influence in the control signal. The opposite is the closed-loop controller, which uses information of the output signal of the system to manipulate the control signal. In this section will be presented the closed-loop controllers ON-OFF, Proportional Integral Derivative (PID) and its variants, and also Reference Signal Tracking (RST). The controllers will serve to test the functional and temporal performance of the developed platform.

2.4.1 ON-OFF

The controller ON-OFF is the simplest to implement. This controller only has two states, it outputs the maximum value in case the difference between the output system and the setpoint is positive, or the minimum in case the difference is negative. The disadvantage of using this controller is that leads to a oscillatory output of the system, as the controller is oscillator.

If the output of the system is contaminated with noise it should be add hysteresis to the controller.

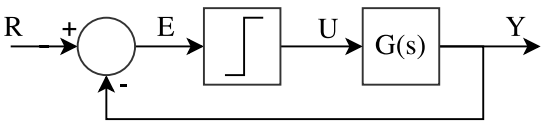


Figure 2.2: ON-OFF controller

The control equation is:

$$u(k) = \begin{cases} U_{max} \text{ if } e(k) > 0 \\ U_{min} \text{ if } e(k) < 0 \end{cases} \tag{2.1}$$

,where $e(k) = r(k) - y(k)$, U_{max} and U_{min} are the maximum and minimum of the output of the control signal.

2.4.2 PID

In this subsection it will be presented the PID controller, one of the oldest strategies still in use. According to a survey fulfilled in 1989, more then 90% of the controllers used in the industry were PID type [24]. Nowadays the PID controller continue to be widely used [25].

PROPORTIONAL

To better understand the PID controller, first it should be explained the behavior of a simple proportional controller. As the name implies, the output of the controller is proportional to the difference of the output system and the setpoint, $e(k)$ (Figure 2.3). In practice, this is only valid until the output of the controller saturates, or in another words, when it reaches the maximum and minimum values of the control signal.

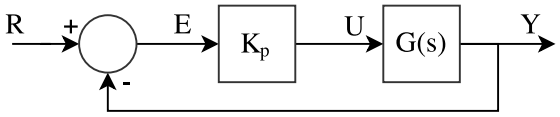


Figure 2.3: P controller

The output signal of the controller is:

$$u(k) = \begin{cases} U_{max} & \text{if} \quad e(k) > +e_o \\ u_0 + K_p e(k) & \text{if} \quad -e_o \leq e(k) \leq +e_o \\ U_{min} & \text{if} \quad e(k) < -e_o \end{cases} \tag{2.2}$$

Yet, this controller is not ideal as the output of the system not always reaches the setpoint. In the next subsubsection is presented a controller that intends to solve this problem.

PROPORTIONAL INTEGRAL

The Proportional Integral (PI) controller is consisted of two adjustable parameters, one of which is the K_p proportional value and the other T_i integration time (Figure 2.4).

The control signal is the sum of the proportional value of the error and the integral of this same error times the factor $\frac{1}{T_i}$ (2.3). With this controller, the control signal is always rising when the error has positive signal, or decreasing when the error signal is negative [26]. This allows us to conclude that when the error signal is constant the output of the system is equal to the setpoint.

$$G_c(s) = K_p(1 + \frac{1}{sT_i}) \tag{2.3}$$

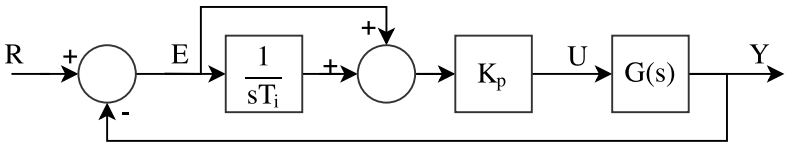


Figure 2.4: PI controller

Using the Equation 2.3 to obtain the Equation 2.4 in discrete time, approximating the derivate operator by the backward difference, where the h is the sampling interval.

$$G_c(q^{-1}) = \frac{s_0 + s_1 q^{-1}}{1 - q^{-1}} \tag{2.4}$$

$$\begin{cases} s_0 = K_p(1 + \frac{h}{T_i}) \\ s_1 = -K_p \end{cases} \tag{2.5}$$

The Equation 2.6 is the control signal derived from Equation 2.5 [26].

$$u(k) = u(k-1) + s_0 r(k) + s_1 r(k-1) - s_0 y(k) - s_1 y(k-1) \tag{2.6}$$

PROPORTIONAL INTEGRAL DERIVATIVE

The PID controller has three parameters to adjust. The proportional K_p, which was presented earlier; the integral T_i, which leads to no error in stationary state; and the derivate T_d, which causes a reduction of overshoot.

Figure 2.5 shows the diagram block of the controller and its equation at 2.7 is in Laplace domain. As in this thesis the controller will be used in discrete time, it was obtained the Equation 2.8 by approximating the derivate and integral operator by the backward difference.

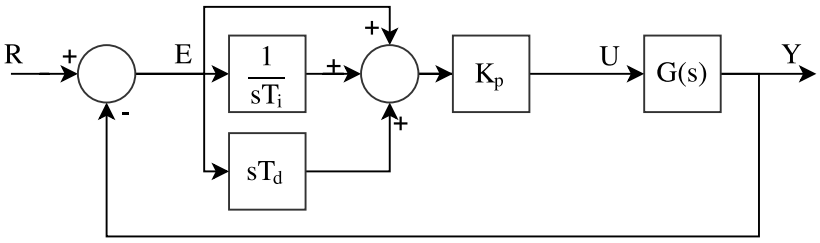


Figure 2.5: PID controller

$$G_c(s) = K_p(1 + \frac{1}{sT_i} + sT_d) \tag{2.7}$$

$$G_c(q^{-1}) = K_p(1 + \frac{h}{T_i}\frac{1}{1 - q^{-1}} + \frac{T_d}{h}(1 - q^{-1})) \tag{2.8}$$

$$G_c(q^{-1}) = K_p\frac{1 + \frac{h}{T_i} + \frac{T_d}{h} - (1 + \frac{2T_d}{h})q^{-1} + \frac{T_d}{h}q^{-2}}{1 - q^{-1}} \tag{2.9}$$

The Equation 2.10 is the expression of the controller with the parameters s_0, s_1 and s_2 on 2.11.

$$G_c(q^{-1}) = \frac{s_0 + s_1 q^{-1} + s_e q^{-2}}{1 - q^{-1}} \tag{2.10}$$

$$\begin{cases} s_0 = K_p(1 + \frac{h}{T_i} + \frac{T_d}{h}) \\ s_1 = -K_p(1 + \frac{2T_d}{h}) \\ s_2 = \frac{K_p T_d}{h} \end{cases} \tag{2.11}$$

From the analysis of the Equation 2.10 it is possible to conclude that the controller has a pole in the origin and two zeros.

The following Equation 2.12 is the control signal derived from the Equation 2.10 [26].

$$u(k) = u(k-1) + s_0 r(k) + s_1 r(k-1) + s_2 r(k-2) - s_0 y(k) - s_1 y(k-1) - s_2(k-2) \tag{2.12}$$

The advantage of using PID is that it allows to control the plant adequately without knowing the transfer functions. This tuning can be achieved manually by trial and error or automatically. More information about the tuning techniques can be found on Subsection 2.4.3.

Due to the derivative component, the quality of the control can be degraded in the presence of noise either on output of the system or at the reference signal.

INTEGRAL - PROPORTIONAL DERIVATIVE

The Integral - Proportional Derivative (I-PD) controller is a variant of the PID in which the derivative components is applied only to the output of the system, so this controller tends to reduce the initial overshoot that happens when the reference point value is changed, but it takes more time to converge to the reference point [27].

Figure 2.6 presents the block diagram of the I-PD controller.

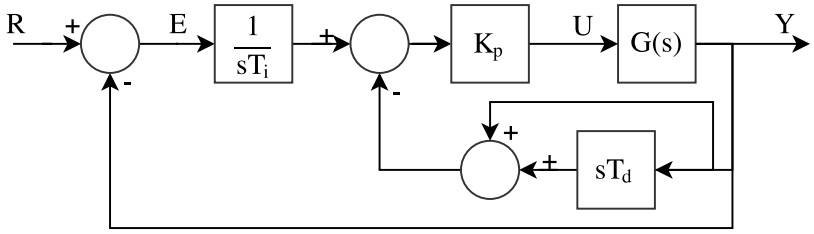


Figure 2.6: I-PD controller

From the block diagram above it was obtained the control signal in Laplace domain (Equation 2.13). From this equation it was derived the control signal in the discrete domain (Equation 2.14)s.

$$U(S) = K_p \frac{1}{sT_i}(R(S) - Y(S)) - K_p(1 + sT_d)Y(S) \tag{2.13}$$

$$u(k) = v_1(k) + v_2(k) + v_3(k) \tag{2.14}$$

$$\begin{cases} v_1(k) = v_1(k-1) + K_p \frac{h}{T_i} e(k) \\ v_2(k) = -K_p y(k) \\ v_3(k) = -K_p \frac{T_d}{h}(y(k) - y(k-1)) \end{cases} \tag{2.15}$$

2.4.3 TUNING METHODS

The tuning methods presented next were proposed by Ziegler-Nichols in 1942 for tune P, PI and PID controllers.

ZIEGLER-NICHOLS OPEN LOOP

One of the methods is performed with the system in open-loop. This can only be employed to a system that can be approximated to a first order system with some dead time. If this condition is met, it is applied a unitary step to the plant and it is observed the output signal. Figure 2.7 is an example of a typical response; from this it is taken the dead time L and the T time constant. So by using this two parameters, it is possible to get parameters for the controller by using the Table 2.3.

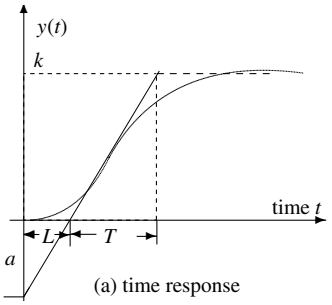


Figure 2.7: Open loop step response

Controller type	K_p	T_i	T_d
P	$\frac{T}{kL}$		
PI	$\frac{0.9T}{kL}$	$3L$	
PID	$\frac{1.2T}{kL}$	$2L$	$\frac{L}{2}$

Table 2.3: Ziegler-Nichols tuning formula in open loop

ZIEGLER-NICHOLS CLOSED-LOOP

The other suggested method is in closed-loop. The plant is put under a P controller and it is possible to put in oscillation with constant amplitude and frequency when the gain K_p increases. For this to be possible the plant has to have excess of poles in relation to zeros by three [26] .

This method intends to obtain the period P_{cr} and the gain K_{cr} of the controller that leads to oscillation. The parameters for the controller P, PI, and PID can be set using the Table 2.4.

Controller type	K_p	T_i	T_d
P	$0.5K_{cr}$		
PI	$0.45K_{cr}$	$\frac{P_{cr}}{1.2}$	
PID	$0.6K_{cr}$	$\frac{P_{cr}}{2}$	$\frac{P_{cr}}{8}$

Table 2.4: Ziegler-Nichols tuning formula in closed-loop

RELAY FEEDBACK METHOD

In 1984 it was presented an auto-tuning method for the PID controller. The method proposed by Åström and Hägglund uses two types of controllers [26] as described in this subsection. First the plant is put under control by an ON-OFF controller, with the intent to put it in oscillation. With this it is intended to obtain the period of oscillation, which is an approximation of the critical period P_{cr} and the amplitude of oscillation a. Knowing the a value it is possible to calculate the K_{cr} with the Equation 2.16, where d is the total amplitude of control signal. Consulting the Table 2.4 it is obtained the parameter for the PID controller. The next step is to switch the ON-OFF controller to the PID controller with the parameters already tunned. The idea of this method is presented on Figure 2.8.

$$K_{cr} = \frac{4d}{\pi a} \tag{2.16}$$

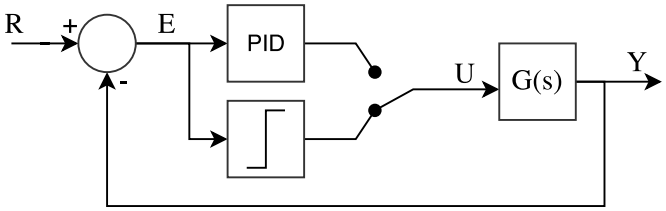


Figure 2.8: Relay feedback auto tuning method

2.4.4 IDENTIFICATION METHOD

The control methods presented earlier can be applied by treating the system as a black box, or in other words, without knowing the composition of the system. In some control techniques can be desirable to know more about the system and it can be theoretical deduced by a mathematical model or by using a system identification technique. Still, when using a mathematical model is not always possible to find all the coefficient of the system and for this reason the system identification techniques are important.

The Least Squares (LS) and it recursive implementation is a method that tries to obtain the coefficient of the model that minimizes the square error of the difference between this model and the system. LS is a parametric regression method, which means it is necessary to know the structure of the model as well as its order. The goal of the method LS is to minimize the function cost of Equation 2.18, using AutoRegressive with eXogenous terms (ARX) model with no noise (Equation 2.17).

$$G(q^{-1}) = \frac{b_1 q^{-1} + ... + b_n q^{-n}}{1 + a_1 q^{-1} + ... + a_n q^{-n}} \tag{2.17}$$

$$J(\hat{\boldsymbol{\theta}}, N) = \frac{1}{2} \sum_{i=1}^{N} \varepsilon^2(i) \tag{2.18}$$

$$\varepsilon(i) = y(i) - \boldsymbol{\varphi}^T \hat{\boldsymbol{\theta}} \tag{2.19}$$

On Equation 2.19 the $y(i)$ is the current output of the system, $\boldsymbol{\varphi}$ are the regressors that usually are a delayed version of the input and output, and $\hat{\boldsymbol{\theta}}$ are the unknown parameters b_i and a_i from the

model ARX. The resolution of Equation 2.18 [28] leads to the equation 2.20, which is only possible if $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ has a determinant not null, with $\boldsymbol{\Phi}$ being the matrix column of all $\hat{\varphi}$ and $\boldsymbol{Y}$ the vector of all Y.

$$\hat{\boldsymbol{\theta}} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{Y} \tag{2.20}$$

The recursive variation of the LS, RLS is more practical when using the identification method for real-time control or when the system is time-variant. The advantages are: less memory required to record the φ (part of the input and output signal), and simpler computational complexity to update the $\hat{\boldsymbol{\theta}}$ (estimated model coefficients).

The solution for the RLS [29] (Equation 2.21), takes the following steps: build the φ_{N+1} with the new incoming data, calculate the weighting factor K_{N+1}, calculate the new error of the model ε_{N+1} and for last update the $\hat{\boldsymbol{\theta}}_{N+1}$. The variable $\boldsymbol{P_N}$ expresses the confidence level in the model; if it is small it means that there is an high confidence level in the $\hat{\boldsymbol{\theta}}_N$. For time-variant systems, one of the techniques used to let the RLS continues to update its $\hat{\boldsymbol{\theta}}$ is to add a forgetting factor coefficient. To accomplish this is only necessary to replace the last row of the Equation 2.21 with the Equation 2.22. Usually the value λ should be only a little lower than one.

$$\begin{cases} \hat{\theta}_{N+1} = \hat{\theta}_N + K_{N+1}\varepsilon_{N+1} \\ \varepsilon_{N+1} = y(N+1) - \varphi_{N+1}\hat{\theta}_N \\ K_{N+1} = P_N\varphi_{N+1}^T(1 + \varphi_{N+1}P_N\varphi_{N+1}^T) \\ P_{N+1} = (I - K_{N+1}\varphi_{N+1})P_N \end{cases} \tag{2.21}$$

$$P_{N+1} = \frac{1}{\lambda}(I - K_{N+1}\varphi_{N+1})P_N \tag{2.22}$$

2.4.5 REFERENCE SIGNAL TRACKING

RST controller is a pole placement control technique to impose the behavior of the system in a closed-loop. It is a controller with two degrees of freedom, feed-forward and feedback actions, in opposition to the controller presented earlier that has only one degree. By knowing the plant, through theoretical equations or some identification system method, it is possible to set the closed-loop poles by changing the RST parameters. Figure 2.9 presents the structure of the controller.

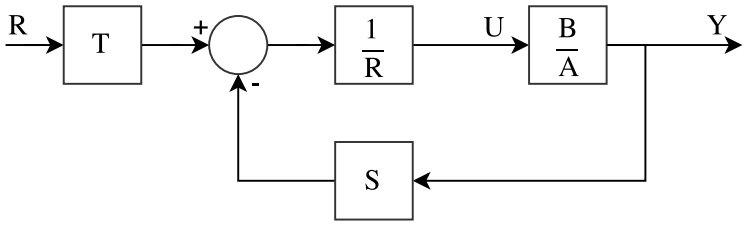


Figure 2.9: RST controller structure

Using the model ARX, the closed-loop model must have the number of poles in Laplace domain greater or equal to the number of zeros; also the poles have to be in the left side of the half s-plane.

$$\frac{BT}{AR + BS} = \frac{B_m}{A_m} \tag{2.23}$$

Equation 2.23 not always let us choose the wanted solution and for this reason is normal to introduce one more degree of freedom, the auxiliary polynomial A_{obs}. This polynomial will not interfere in the output response in relation to the input (Equation 2.24), but will influence the behavior of the system with perturbations (Equation 2.26), when considering the ARX model with noise (Equation 2.25).

$$\frac{BT}{AR + BS} = \frac{A_{obs}B_m}{A_{obs}A_m} \tag{2.24}$$

$$y = \frac{B}{A}u + \xi \tag{2.25}$$

$$y = \frac{B_m}{A_m}u + \frac{R}{A_{obs}A_m}\xi \tag{2.26}$$

The new polynomial A_{obs} should be understand as one parameter more of the controller, and likewise must follow a set of rules:

- $deg(R(q)) \geq deg(T(q))$, to be causal
- $deg(R(q)) \geq deg(S(q))$, to be causal
- It is usual to choose the polynomial R and S with the same degree of A. In this case A_{obs} has to be: $deg(A_{abs}) = 2deg(A) - deg(A_m)$

From Equation 2.24, the solution for polynomial T is direct (Equation 2.27). Then, it is left to solve the Equation 2.28 that gives R and S polynomials. To have no steady state error is needed to have a high gain to low frequencies, which is accomplish by having integral action on the regulator. For this it is used the polynomial R in the form $R = (q-1)R_1$. This leads to the Equation 2.29, usually known as Diophantine equation [30].

$$T(q) = \frac{A(q)_{obs}B_m(q)}{B(q)} \tag{2.27}$$

$$A(q)R(q) + B(q)S(q) = A_m(q) \tag{2.28}$$

$$A(q)R_1(q) * (q-1) + B(q)S(q) = A_m(q) \tag{2.29}$$

Following the rules for a first order system in closed-loop, we reach the statements:

- $A(q) = q + a$
- $B(q) = b$
- $A_m(q) = q + a_m$
- $A_{obs}(q) = 1 + a_o$
- $B_m(q) = b_m$
- $R_1(q) = 1$
- $S(q) = qS_0 + s_1$

In this case, s_0 and s_1 are the only parameters left to define, and can be found by the Equation 2.30.

$$\begin{cases} s_0 = \frac{1-a+a_m+a_0}{b} \\ s_1 = \frac{a_m a_0+a}{b} \end{cases} \tag{2.30}$$

For a second order system in closed-loop it is known that:

- $A(q) = q^2 + qa_1 + a_2$
- $B(q) = qb_1 + b_2$
- $A_m(q) = q^2 + qa_{m1} + a_{m2}$
- $A_{obs}(q) = q^2 + qa_{obs1} + a_{obs2}$
- $B_m(q) = qb_{m1} + b_{m2}$
- $R_1(q) = q + r_1$
- $S(q) = q^2 s_0 + qs_1 + s_2$

In this case, is needed to determine r_1, s_0, s_1 and s_2. The solution for this parameters can be found at Equation 2.31, where P_0, P_1, P_2, P_3, P_4, P_5 and P_6 are defined at 2.32.

$$\begin{cases} r_1 = P_0 - b_1 s_0 \\ s_0 = -\frac{P_1+b_1 s_1}{P_2} \\ s_1 = -\frac{b_1(P_5 P_2 - P_1 a_2 b_1)+b_2 P_6}{b_2(P_4 b_1 - P_2 b_2)-a_2 b_1^3} \\ s_2 = \frac{P_6+(P_4 b_1 - P_2 b_2)s_1}{P_2 b_1} \end{cases} \tag{2.31}$$

$$\begin{aligned} P_0 &= a_{m1} + a_{aobs1} + 1 - a_1 \\ P_1 &= a_2 + a_1 P_0 - a_{m1} - a_{m2} - a_{obs2} - a_{obs1}(1 + a_{m1}) - 1 \\ P_2 &= b_1(1 - a_1) + b_2 \\ P_3 &= a_2(P_0 - 1) - a_1 P_0 - a_{m1} a_{obs2} - a_{m2} a_{obs1} \\ P_4 &= b_1(a_1 - a_2) \\ P_5 &= -a_2 P_0 - a_{m2} a_{obs2} \\ P_6 &= -P_3 P_2 + P_4 P_1 \end{aligned} \tag{2.32}$$

With the polynomials $S(q)$,$R(q)$ and $T(q)$ calculated, it is only missing the control function, Equation 2.33.

$$u(k) = \frac{T(q)r(k) - S(q)y(k)}{R(q)} \tag{2.33}$$

CHAPTER 3

Hardware Development

3.1 REQUIREMENTS

The objective of this work is to build an embedded system to be used in a educational environment, helping students to learn and test control algorithms in a physical system. This system will have two operation modes, shown on Figure 3.1. The first mode allows a fast development of the control algorithm in a familiar programmable environment for the user, MATLAB. The developed system will communicate with MATLAB by the Ethernet interface, and will only work as digital and analog interface to the physical system. In the second mode, the embedded system has to work without any assistance of an external computer. Thus, it must realize all needed calculations, read and actuate on time in the physical system. Before proceeding to the selection of the hardware and describing the developed system, it is necessary to define the requirements. The hardware in development has to be capable to read and operate in a mix of digital and analog system, with at least four Digital Input (DI) and four Digital Output (DO) compatible with 5 Volt Complementary Metal-Oxide-Semiconductor (CMOS) technology, capable of supplying more than 20mA per channel and featuring input protection. Furthermore, in the digital side is needed four Pulse-Width Modulation (PWM) channels with clock frequency up to 20kHz. On the analog side, it is necessary at least four ADCs able to measure from -5V to 5V with an input resistance of more than 1GΩ, 5mV resolution and sample frequency of 1kHz, and four DACs channels with the same voltage range, resolution and frequency characteristics and with an output current of 20mA and short-circuit protection. The board has to be powered from an unique supply voltage of 5V and generate internally all the voltages needed. Figure 3.2 represents the block diagram of the target system.

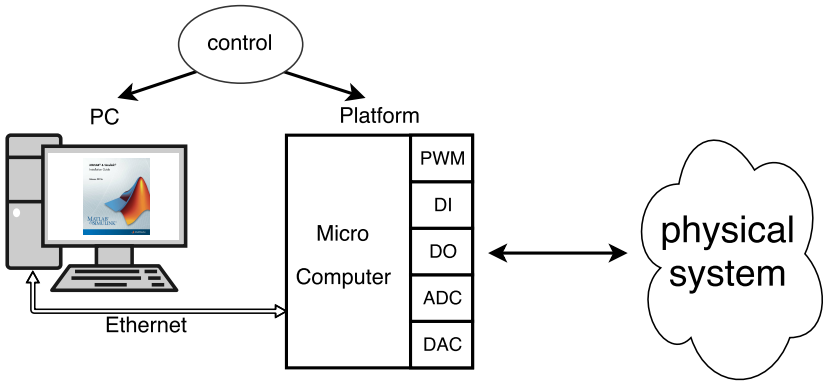


Figure 3.1: Multiple working modes

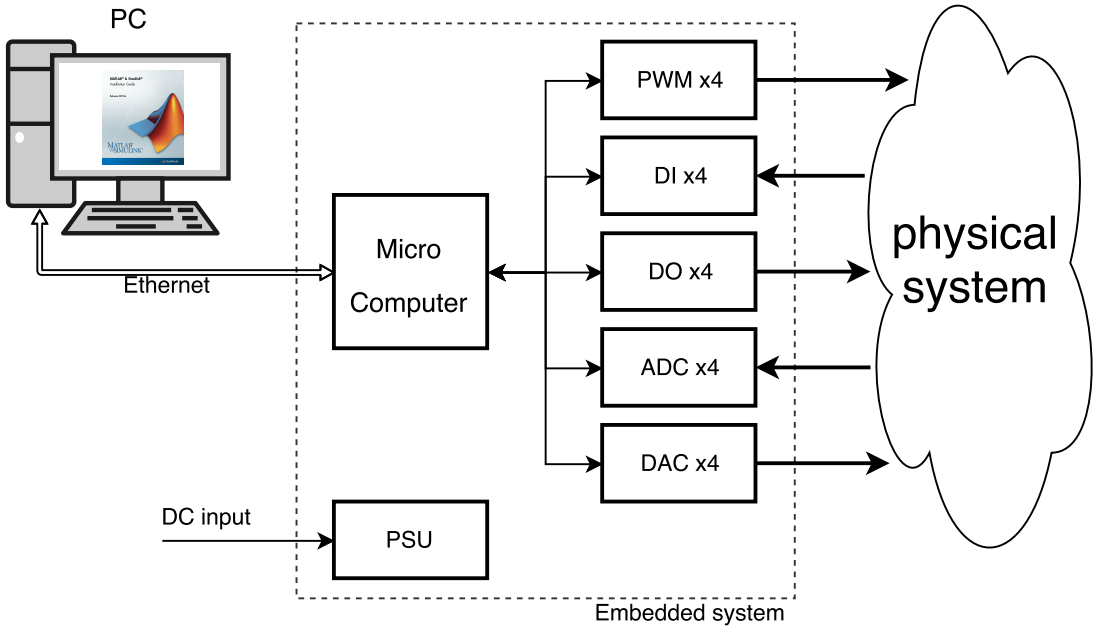


Figure 3.2: Block diagram

3.2 PROCESSOR UNIT

For this work it is necessary an embedded system to handle Ethernet communications while working alongside MATLAB and to be capable to execute the control loop code when working in independent mode. It also has to be able to interact with the physical system or at least has to have protocols to talk with the Integrated Circuits (ICs) responsible to read and actuate in the system. Taking in account the intended use of the embedded system, it was made an analysis of systems like Raspberry Pi, Beaglebone and others. The choice of the system fell upon the Raspberry Pi 1, model B, because it is the most affordable option, is available at the university and was already used in previous projects. This is a small credit-card size single micro-computer board with the purpose to allow children to have access to a computer at home. It features a Broadcom BCM2835 System On Chip (SOC) containing a single core ARM11 processor with floating point running at 700MHz clock speed, an integrated Graphics Processing Unit (GPU) [31] with 512MB Random-Access Memory (RAM) distributed for

both processors. The SOC connects to a USB hub which provides a 100Mb/s Ethernet connection and two USB 2.0 ports. It provides a 34 pin connector with General-Purose Input/Output (GPIO) ports, common protocol communications as Universal Asynchronous Receiver/Transmitter (UART), SPI, I2C and also some peripherals as PWM. Figure 3.3 shows the placement of components and connectors of Raspberry Pi.

This computer runs a free OS based on Debian, a well known Linux distribution [32]. Every OS has a learning curve to allow users to understand how the system is designed and to know how to program using its Application Programming Interface (API). It is here that Linux presents more advantages over any other system, with many books, on-line tutorials and a big community ready to help. Linux is an open source, which means the code is available to anyone to read it, modify it and contribute with their one modifications, which leads to faster and multiple solutions for one problem.

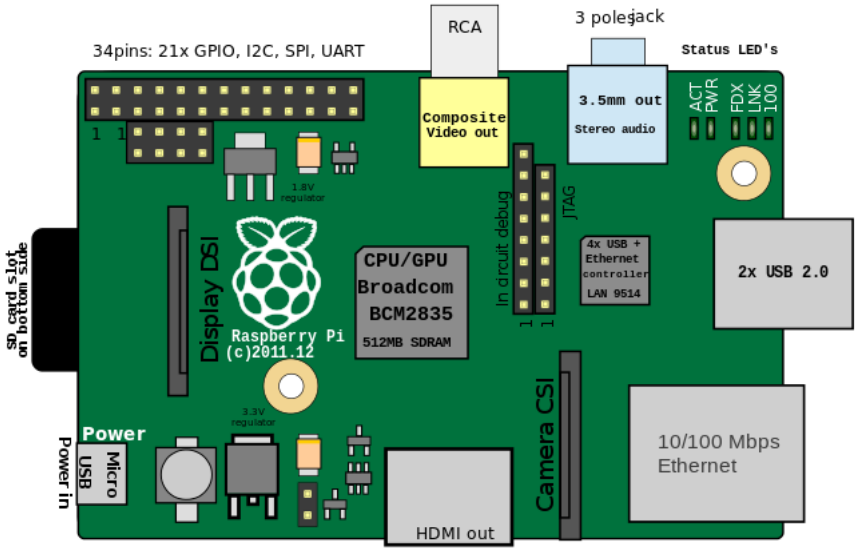


Figure 3.3: Raspberry Pi 1 model B plant

3.3 HARDWARE IMPLEMENTATION

Although the Raspberry Pi already provides some of the important features required, it does not fulfill all the electrical requirements defined on Section 3.1. This section will address the main ICs used to complement the functionalities missed by Raspberry PI, namely the analog and digital I/O, and the PWM functionality. The additional hardware was selected taking in consideration the size, price, functionality and past knowledge.

3.3.1 COMPLEMENTARY INTEGRATED CIRCUITS

MAX11300

The most important complementary IC used is the MAX11300 [33] from Maxim Integrated. It was used because of its price, multiple functionalities, availability at the university and because it was already used in older projects. It has 20 I/O ports, each one configurable as ADC, DAC or GPIO, with ports capable of being configured as logic level translators or analog switches, and with a internal voltage reference. The ADCs and DACs have a 12 bits resolution, with selectable input and output voltage ranges from -10V to +10V.

The advantages of using this "Swiss army knife", as the EETimes calls it [34], are being: a very capable and configurable IC that attends most of our needs; allowing to set the sample rate of the ADC, the number of samples before sending the ADC value, the threshold for the digital input ports and the high level voltage for a digital output ports. Other way to meet the requirements fulfilled by the IC could be using multiple dedicated ICs to perform each function, but this would lead to a more expensive and complex solution.

MAX11300 features a high speed SPI and a digital core, which can be used to configure the device, read or write values in the ports, and two more special pins, one to trigger an ADC conversion and another that can be configured to signal the CPU about multiple events as an ADC conversion done, DAC over-current or the internal temperature.

A more complete description of the IC can be found in Table 3.1

Parameter	Min	Typical	Max
ADC range	-10V		10V
ADC resolution	12 bits		
ADC offset error	-5LSB		5LSB
ADC sample rate	200kps		400kps
ADC sample number	1		128
ADC input resistance	70kΩ	100kΩ	130kΩ
DAC range	-10V		10V
DAC resolution	12 bits		
DAC offset error	-20LSB		20LSB
DAC refresh time			40us
DAC output current			25mA
GPI programmable threshold	0.3V		2.5V
GPO programmable High	0V		10V
Voltage reference	2.494V	2.5V	2.506V
SPI clock speed			20MHz

Table 3.1: Electrical and functional description of MAX11300

MAX31790

Most of the functional requirements were fulfilled by the previous IC, missing only PWM and I/O protection. Although Raspberry Pi has PWM built in, it only has two channels available and on Section 3.1 it was state the need of four channels. For this reason it is used an external IC to accomplish

Parameter	Value
PWM resolution	9bits
PWM frequency	25Hz, 30Hz, 35Hz, 100Hz, 149.7Hz, 1.25kHz, 1.47kHz, 3.57kHz, 5kHz, 12.5kHz, 25kHz
PWM frequency accuracy	+-6%
PWM maximum voltage	5.5V
PWM maximum current	5mA
TACH count resolution	11bits
I2C clock speed	400kHz

Table 3.2: Parameters of MAX31790

this functionality. The IC responsible for the PWM is the MAX31790 [35], from Maxim Integrated, for the following reasons: the first and more important is the existence of a kernel device-driver for Linux, which will be of much help at making the porting to Xenomai Real-Time Driver Model (RTDM). Also the IC has six PWM channels with Open-Drain outputs and six tachometer channels, both tolerating up to 5.5V. The PWM has 12 programmable oscillating frequencies shared in groups of three channels, and the communications are performed by I2C protocol. More details can be found at the Table 3.2.

INPUT AND OUTPUT PROTECTION

It can not be neglected that this work aims the educational environment, where it is likely for the equipment to be misused. For this reason it is necessary to have electrical protection for the more expensive ICs, and the protection circuit has to be easily replaced when permanent damage happens.

For the I/O digital channels configured on MAX11300 it is used the 74HC244 [36] in Plastic Dual-In-line Package (PDIP) package. The 74HC244 is a very common IC easy to find on retailers. It has eight buffers in one package with clamp diodes for excessive voltage protection in the input. A more complete description of this device can be found in Table 3.3.

Parameter	Value
HIGH-level input voltage	4.2V
LOW-level input voltage	1.35V
HIGH-level output voltage	4.4V
LOW-level output voltage	0.1V
Maximum output current	35mA

Table 3.3: Electrical characteristic of 74HC244 from -40°C to 80°C at 5V

For protecting the MAX11300's analog channels was selected a TL071 [37] Op-Amp on PDIP package, for the same reasons described above. It is a low-noise JFET-input Op-Amp, which means high input resistance with input offset compensation, high slew rate, and with an output capable of being in a continuous short-circuit for an undefined amount of time. If a better input offset characteristics without the need of offset compensation is desired, the OP27 [38] can be used as replacement, with the disadvantage of being more expensive. For an easier comparison between the two, consult Table 3.4.

The MAX31790 PWM ports are Open-Drain, which means that they can be short circuited to ground without any damage and pulled as high as 5.5V safely if the current limit is respected. The tachometer ports are 5.5V tolerant, and for most case scenarios this value is enough. For those reasons and to save space on the Printed Circuit Board (PCB), it was not implemented any protection circuit for this IC.

Parameter	TL071	OP27
Supply voltage	+-18v	+-22V
Input offset voltage	13mV	50uV
Input resistance	1TΩ	3GΩ
Output maximum current	25mA	35mA

Table 3.4: Electrical characteristics of Op-Amp TL071 and OP27 from -25°C to 85°C

3.3.2 POWER SYSTEM

The system should be powered by a single supply of 5V. It is used 5V coming from the GPIO connector of the Raspberry Pi. As the MAX11300 and the Op-Amp need -7.5V and 7.5V, it is used a dual switch regulator, LT3471 [39] from Linear Technology, followed by two linear regulators for noise improvement. Two more linear regulators were used to supply the 5V and 3.3V that the IC needs. These supplies are not only used to power the internal IC, but will also be available to power any other circuit outside the board.

It is the responsibility of the user to respect all the limitations of each power supply source. The maximum current on each rail can be found on table 3.5.

Supply voltage	Maximum Current
-7.5V	-50mA
3.3V	100mA[1]
5V	100mA[1]
7.5V	100mA[1]

Table 3.5: Maximum current available in each rail

3.4 PRINTED CIRCUIT BOARD

In the scope of this thesis it was developed a prototype board named RasP:IO that contains all the required electronic component and interfaces. This board was essential to test and validate the project. The first step was to make the schematic that can be found on Appendix A with all the IC mentioned in this chapter. Afterwards was designed a four layers board; one of the middle layers is a ground plane to reduce the noise on analog side and the rest of the layers are used to route all other

[1]Note the current in 3.3V, 5V and 7.5V rails can not exceed 100mA combined.

electrical signals. The dimensions are 10 by 10cm, the maximum dimension possible that respects the target price to manufacture.

On Figure 3.4 can be found a 3D rendering of the developed PCB with the components marked:

- Left side - power output connector: The first four ones are ground, followed by -7.5V, 3.3V, 5V and 7.5V. Next is the input power connector. In the first contact can be connect the 5V and the second GND. The last is a connector compatible with the Raspberry Pi GPIO port.
- Bottom side - analog I/O: The first connector is the analog input with four grounds and four analog inputs interleaved. In the same pattern is the second connector, the analog output, with four grounds and four analog outputs.
- Top side - PWM & Tachometer connector with the Digital I/O. The first has four tachometer inputs intercalated with four PWM channels. The second has four outputs followed by four inputs, both digital.

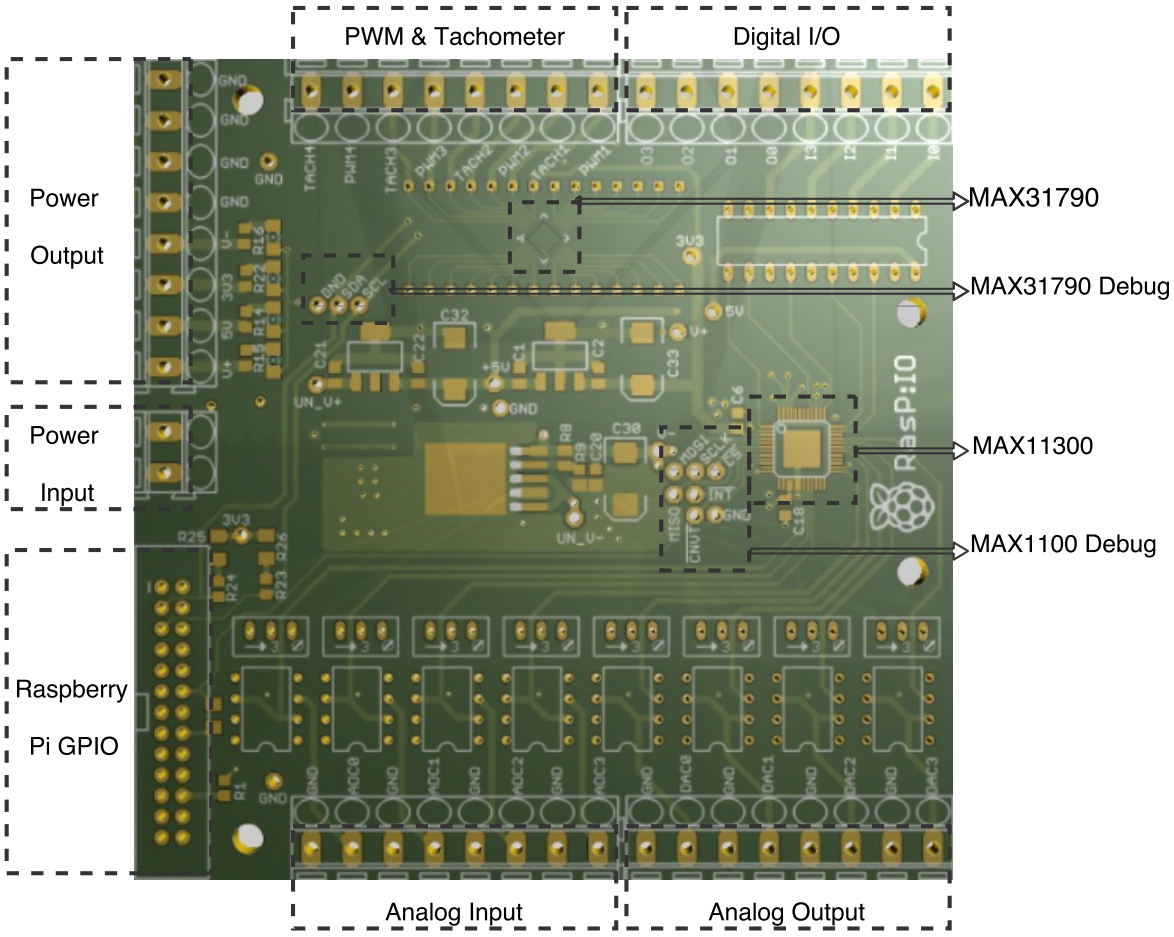


Figure 3.4: Plant of the developed PCB

3.5 SUMMARY

In this chapter it was stated the requirements of the project, the selection of the ICs and the CPU.

For the CPU, it was chosen Raspberry Pi 1 model B for its Ethernet interface, ability to run a Linux distribution, the multiple GPIO and CPU.

As Raspberry Pi does not have all the need functions, it was added two ICs, MAX11300 for the analog and digital I/O and the MAX31790 to generate the PWM.

Finally, it was developed a PCB. This chapter shows the location of the most important ICs and connectors, as well the interfaces.

CHAPTER 4

Software and Firmware

This chapter presents the firmware that supports the communication with the multiple ICs on the board and the Hardware Abstract Layer (HAL) to interface the hardware with the higher level software without worrying about the technical details of the hardware. Then it is described the Ethernet based protocol that was implemented to allow the communications between the Raspberry Pi and the PC. Finally, it will be presented the software that allows interacting with the Raspberry Pi and RasP:IO board as an I/O interface as well as turning them on a standalone controller.

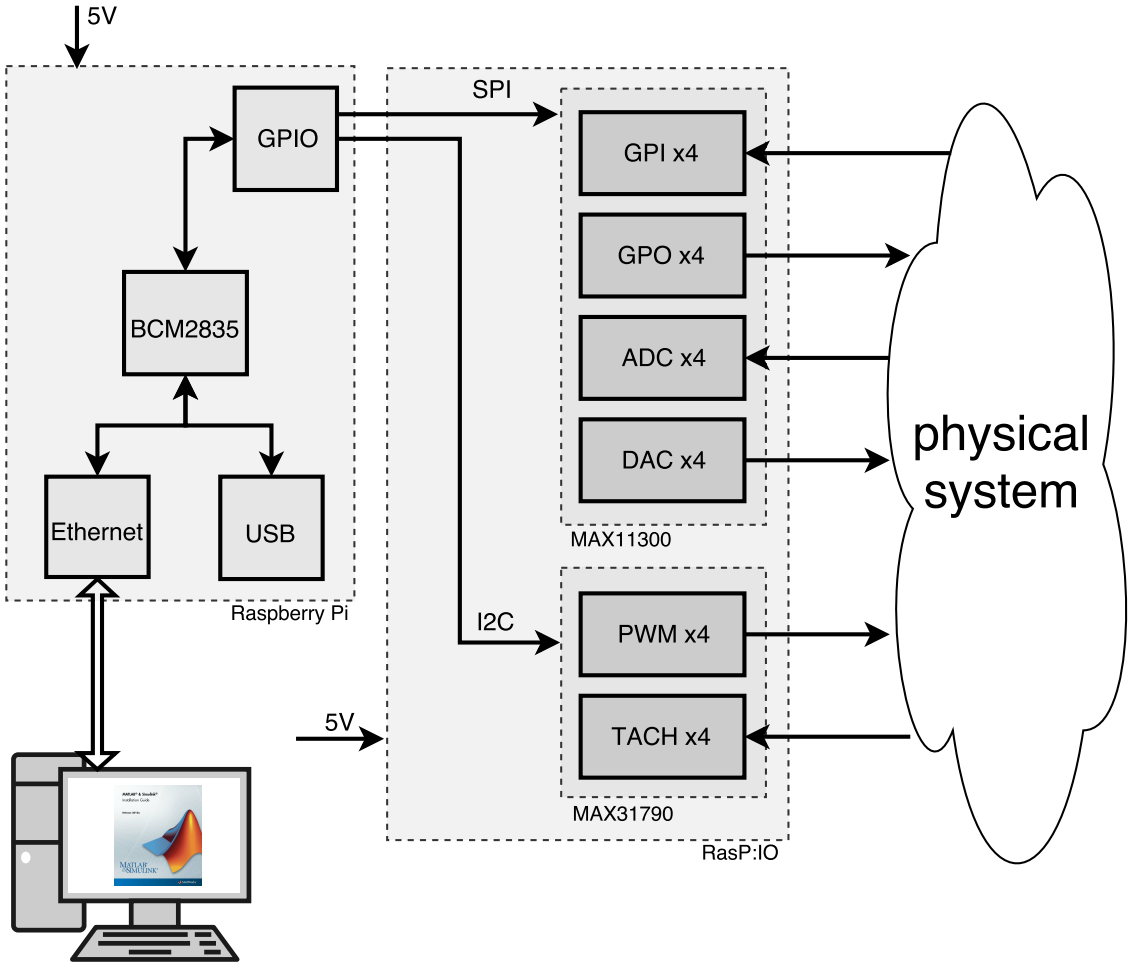


Figure 4.1: System block diagram

4.1 FIRMWARE

4.1.1 DEVICE DRIVER

Raspberry Pi is running Raspbian version of May, 2015, a distribution of Debian, that came with kernel on version 3.18 . To install the version 2.6 of Xenomai, was necessary to revert the Linux kernel to the version 3.10, in which device drivers like SPI were not working correctly. Installing Xenomai on Raspberry Pi was not a trivial task, and after having it working it was created a guide with the steps taken. The guide can be consulted on Appendix B.

Raspberry PI features SPI with two FIFOs of 16 bytes, one to store the data to send and the other the received data. Most of the operations carried out by the system imply the use of the SPI interface. Therefore it is important to choose an efficient device driver. For example, one of the ICs used on the board is the MAX11300, which has an SPI interface to communicate with Raspberry Pi.

There are multiple options for the device driver from SPIDEV, a library working in user-land and written in C, implemented on Linux in Kernel Space but only accessible on user-land [40], or more preferable a real-time device driver working on Xenomai. When the software was developed and tested there was no RTDM for SPI on Xenomai available, and implementing one is a difficult and time consuming task. SPIDEV suffers from latency [41], and for this reason it was discarded. Therefore, the

choice of the driver fell back on BCM2835, an Open Source library written in C [42]. This library was modified to transmit in polling mode a block of data with no more than 16 bytes. This way it takes advantage of the Hardware's FIFO and the transfer occurs in a predictable time. Another modification done is the inclusion of a timeout mechanism.

4.1.2 HARDWARE ABSTRACT LAYER

Having the SPI device driver decided, it is time to start communicating with MAX11300. The address space is of 7 bits and the registers are 16 bits. For this reason to do any type of request or communication, it is always needed to send the address number followed by the read/write bit, and then read or write in multiples of two bytes, of which the first is the most significant one. Figure 4.2 shows an example of a read operation of two bytes. Before doing any operation involving the ports, the IC must be configured. Maxim Integrated has available configuration software [43] and it generates all code in C for the initialization of the IC, defines the channels type, the sample frequency, number of samples per read, maximum and minimum voltage of the ADC and DAC, decision level of the digital input, output logical one level, and more. All important configurations can be found on Table 4.1.

Description	register value	value meaning
ADC conversation mode	3	continuous sweep
ADC conversation rate	0	200kps
ADC sample average	3	8 samples
ADC voltage range	2	-5V to 5V
ADC voltage reference	0	internal
DAC voltage range	2	-5V to 5V
DAC voltage reference	0	internal
GPI threshold	4095	2.5V
GPO logical one level	2047	5V
Temperature monitor	1	internal
Temperature sample average	1	8 samples

Table 4.1: Configuration parameters for MAX11300

The ADC is 12 bits and it works in continuous mode. It is always necessary to read two bytes of data to get the analog value on each channel. It was selected the conversation rate of 200kps, because

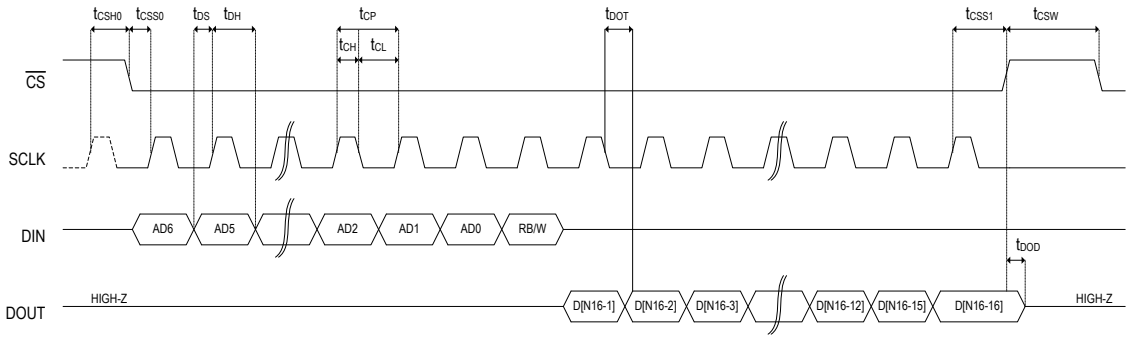


Figure 4.2: SPI reading

it provides the maximum acquisition time (3.5us) available on the IC. With a longer acquisition time is possible to decrease the offset DC voltage on the hold phase of the Sample And Hold (SAH) circuit [44]. As there are four channels configured as analog inputs, from channel 0 to 3, and the internal temperature monitor also uses the ADC, all doing eight sample averages, it leads to a maximum theoretical refresh rate of 5000Hz. It was selected the highest number of sample averages with care to have a sample time of five times less [45] than the minimum period of the control loop. Figure 4.3 shows the total number of samples done on one sweep, as well as the acquisition time.

To get the analog value of one of the pins configured as analog input first, it is read a 16 bits register to check if there is a new analog conversion available. If yes, it is read a 32 bits register to check in which pin is available the new conversion. If the desired pin has a new conversion, it is read a 16 bits register associated with the chosen pin and in the end it is erased the first 16 bits register. If any of this conditions fails, there is a wait time of a full sweep conversion and then it is made a new attempt. Figure 4.2 shows the situation where the initial four bits are zeros, followed by 12 bits with the analog value. The procedure to read the four ADC channels is the same. It is read a register to check if there are analog conversions ready, and if it is true then it is read the second register to verify if the four pins have new conversions available. Then it sends the address of the first channel and waits for the return of eight bytes. This is possible to do in one SPI transaction because the ADC channels are mapped in sequence. If any of the check flags fail, it is perform the same behavior describe before.

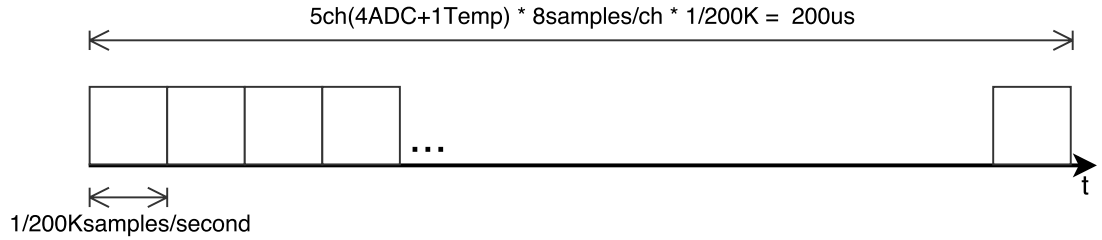


Figure 4.3: Number of samples per ADC sweep

The DAC has a 12 bits resolution being responsible to set the analog output, define the General Purpose Input (GPI) threshold voltage between the digital levels, and also define the voltage of the logical one level of General Purpose Output (GPO). So in total there are 12 channels depending of the internal DAC in the IC, if every channel needs 40us, the minimum refreshment rate is 1887Hz, taking in consideration the introduction of multiples of 80us that will be explain later. To set one value on analog output, it is necessary to send the address of associated register port with the write bit, followed by 16 bits that contains the 12 bits value aligned to the right. The analog outputs have two working modes; the first is the sequential update mode. In this mode the DAC would be updating the 12 DAC depending pins all sequentially when it receives or not a new value to be set. This would introduce a variable time of actuation from 0 to 440us (11channels*40us). The second mode, which is used here, makes the update immediately when is set a new value in one of the analog output channels. The disadvantage of using this mode is that successive writes to the DAC register must be separated by 80us as can be seen in Figure 4.4. So in this mode, when it is wanted to set the four DAC channels, there is a delay of 360us between the first and fourth channel, against the 120us of the first mode.

Digital inputs use the DAC to set the threshold between high and low level, which can be any value from 0.3V to 2.5V. It was chosen 2.5V, because it is the medium value of the output of the protection circuit (Table 3.3). The GPI bits of the channels are split in two register of 16 bits. On software the registers were concatenated in one 32 bits register, to have a direct match between the

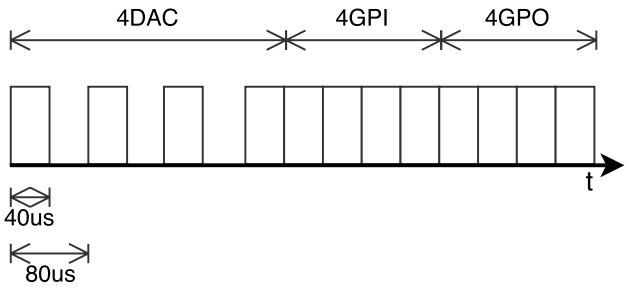


Figure 4.4: Worst case time for DAC sweep

number of the pin and the position of the digital value on the register. In order to read the digital value of a pin, it is just needed to read the bit(s) number of the wanted pin(s).

Digital outputs use the DAC to set the logical one level and it can be any value from 0V to 5V. The value used was the maximum available, 5V, to have the best margin for the input protection circuit, (Table 3.3). The write operation in the digital port is similar to the read one, with the difference that is necessary to do a read-modify-write operation to change only the wanted bit(s) and preserve the remaining ones.

In 'MAX11300Hex.h', the file generated by the configuration software, there is the first HAL function that initializes the MAX11300 with the values set on table 4.1. The function is:

Sintax:	void MAX11300init()
Arguments:	None
Return:	None

In the file 'raspio_max11300.h' it is possible to find two sets of functions, the private ones to access the hardware in low level, and the public that implement the HAL. These are:

- The function getADC is used to get only a new read from ADC.
  Sintax: uint8_t getADC(uint8_t channel, float *voltage)
  Arguments: channel: unsigned byte, value from 0 to 3
  voltage: float pointer, value came in SI unit
  Return: different value from 0 in case of error

- The function getADCAll is used to get all analog values at the same time from the channels configured as analog inputs.
  Sintax: uint8_t getADCAll(float *voltage)
  Arguments: voltage: float array of 4 elements, values came in SI unit
  Return: different value from 0 in case of error

- The function setDAC is used to set the output voltage of one of the DAC channels.
  Sintax: uint8_t setDAC(uint8_t channel, float voltage)
  Arguments: channel: unsigned byte, value from 0 to 3
  voltage: float, value from -5 to 5 in SI unit
  Return: different value from 0 in case of error

- The function setDACAll is used to set all output voltages of all the channels configured as analog outputs.
  Sintax: uint8_t setDACAll(float *voltage)
  Arguments: voltage: float array of 4 elements, values from -5 to 5 in SI unit
  Return: different value from 0 in case of error

- The function getIN is used to get a new digital value from one of the channels.
  Sintax: uint8_t getIN(uint8_t channel, uint8_t *boolean)
  Arguments: channel: unsigned byte, value from 0 to 3
  boolean: unsigned byte pointer, 0 for low level or 1 to high
  Return: different value from 0 in case of error

- The function getINAll is used to get all the digital values from the channels configured as digital inputs.
  Sintax: uint8_t getINAll(uint8_t *boolean)
  Arguments: boolean: unsigned byte array of 4 elements, 0 for low level or 1 to high
  Return: different value from 0 in case of error

- The function setOUT is used to set a new digital value in one of the digital outputs.
  Sintax: uint8_t setOUT(uint8_t channel, uint8_t boolean)
  Arguments: channel: unsigned byte, value from 0 to 3
  boolean: unsigned byte, 0 for low level or 1 to high
  Return: different value from 0 in case of error

- The function setOUTAll is used to set all digital values in all channels configured as digital outputs.
  Sintax: uint8_t setOUTAll(uint8_t *boolean)
  Arguments: boolean: unsigned byte array of 4 elements, 0 for low level or 1 to high
  Return: different value from 0 in case of error

4.2 COMMUNICATION PROTOCOL FOR MATLAB AND RASPBERRY PI

4.2.1 PROTOCOL OVERVIEW

To have MATLAB doing all data processing running in a PC with Microsoft Windows, OSX from Apple or Linux and use the Raspberry Pi as an interface is necessary to choose the mode of communication between both. Raspberry Pi supports multiple serial protocols: UART, I2C, SPI and Ethernet. The choice fell back on Ethernet because it is available in most modern PC, so it does not require additional hardware except for a cable. Ethernet has a wide bandwidth, it is supported by most modern OS without the need to do any software installation; in one cable is possible to run multiple services and can work in ranges up to 100 meters [46].

UDP is more suitable than TCP for network applications with low latency goals, because the latter one has a non-deterministic behavior, due to the retransmission and congestion mechanisms employed. Although UDP has this characteristics, it has one unwanted behavior, that is the lack of guarantees

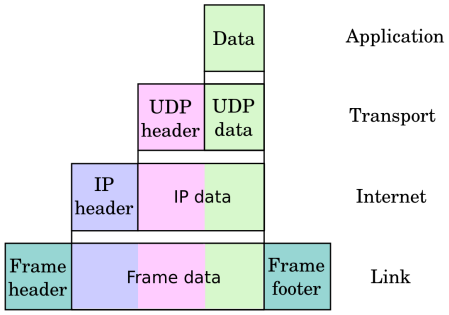


Figure 4.5: Internet protocol suite

of packet delivery. As this application is sensitive to packets losses, it is necessary to implement complementary mechanisms to ensure packet delivery. To achieve this, several mechanisms have been implemented. First, the server will respond to every message with an acknowledge or appropriate answer. There is a sequence number in the beginning of every message to make sure that it has not received messages out of order and a checksum number in the end of the messages to check if the content was not modified in the transmission or in the copy to the UDP socket.

In this protocol is implemented the client-server model type. Raspberry Pi is the server with known Internet Protocol (IP) address and port capable to attend only one client at time and the client will be the computer running MATLAB. The client is responsible to initiate and end the communication, and has the task of incrementing the sequence number on every new sent message and wait for the response. In case it passes more than the defined time-out value after the message was sent, and it was not received any answer, it will be send again the same message. If the arrival keeps failing it will be tried one more time, and if this behavior is maintained, the user will be informed with a time-out error. Another possible error is if the sequence number received does not match the sent one, or the checksum is wrong; in this situation it tries to read again the socket for two more times. If no valid message has arrived, the message is retransmitted in the same conditions as explained before.

Although this protocol is created for real-time application, neither MATLAB nor the operative systems are real-time, so there are not guarantees that the time-out value will be always respected, what can lead to longer time of execution.

The value chosen for the time-out was 8ms. The value was adjusted to get a failure rate lower than 0.01% and it was obtained exchanging messages between MATLAB and Raspberry PI, using the implemented protocol.

4.2.2 NETWORK COMMANDS

This subsection presents the commands created to interacted with the HAL on Raspberry PI. Most commands shown here use the same name of the ones presents in the HAL on Subsection 4.1.2, but it is to notice that the parameters and return value can differ. In the end of this subsection there are two more commands related to PWM functionality but there is no hardware supporting them.

START CONNECTION

The client has to initiate the communication with the server, which is done with the start command.

Syntax:	start( )
Arguments:	None
Return:	0 or NaN (Not a Number) in case of error

END CONNECTION

When the client does not need to do any more operations with the server, it must send the end command.

Syntax:	end( )
Arguments:	None
Return:	0 or NaN in case of error

GET A NEW ADC VALUE

The command getADC is used to get a new read from the ADC.

Syntax:	getADC(channel)
Arguments:	Channel: int value from 0 to 3
Return:	Voltage, float beteween -5 or 5, or NaN in case of error

GET ALL ADC VALUES

The command getADCAll is used to get all the analog values at the same time from the ADC.

Syntax:	getADCAll( )
Arguments:	None
Return:	Voltage, float' array with 4 elements between -5 and 5 volt, or NaN in the position(s) with error(s)

SET A NEW DAC VALUE

The command setDAC is used to set the output voltage of one of the DAC channels.

Syntax:	setDAC(channel, voltage)
Arguments:	Channel: int value from 0 to 3
	Voltage: float between -5 and 5 volts
Return:	0 or NaN in case of error

SET ALL DAC VALUES

The command setDACAll is used to set the output voltage of all the DAC channels.

Syntax:	setDACAll(voltage[])
Arguments:	Voltage: float' array with 4 elements between -5 and 5 volts, or NaN on the position(s) to not set any
Return:	0 or NaN in case of error

GET A NEW DIGITAL INPUT

The command getIN returns the digital value from one of the digital inputs.

Syntax:	getIN(channel)
Arguments:	Channel: int value from 0 to 3
Return:	Boolean 0, 1 or NaN in case of error

GET ALL DIGITAL INPUTS

The command getINAll returns the digital values from the digital inputs.

Syntax:	getINAll( )
Arguments:	None
Return:	Boolean' array with 4 elements of 0, 1 or NaN in the position(s) with error(s)

SET A NEW DIGITAL OUTPUT

The command setOUT sets a digital outputs to one logical level.

Syntax:	setOUT(channel, boolean)
Arguments:	Channel: int value from 0 to 3
	Boolean: 0 or 1
Return:	0 or NaN in case of error

SET ALL DIGITAL OUTPUTS

The command setOUTAll sets all the digital outputs to the given logical level.

Syntax:	setOUTAll(boolean[])
Arguments:	Boolean' array with 4 elements of 0, 1 or NaN in the position(s) to not set any value(s)
Return:	0 or NaN in case of error

SET PWM FREQUENCY VALUE

The command setPWMFreq sets the frequency of the PWM channel. In case the requested frequency is not supported by the hardware, it is applied the next lower value, and returned in the answer message. It is important to notice that in the Rasp:IO, the first three channels share the same clock, so any change in frequency in one channel will be applied to the other two. For more information about this limitation, consult Section 3.3.1 on Chapter 3.

Syntax:	setPWMFreq(channel, frequency)
Arguments:	Channel: int value from 0 to 3
	Frequency: 25, 30, 35, 100, 125, 150, 1250, 1470, 3570, 5000, 12500 or 25000 in Hz
Return:	Frequency or NaN in case of error

SET PWM DUTY-CYCLE VALUE

The command setPWMDuty sets the duty-cycle value of one of the PWM channels.

Syntax:	setPWMDuty(channel, duty-cycle)	
Arguments:	Channel:	int value from 0 to 3
	Duty-cycle:	float between 0 and 100
Return:	0 or NaN in case of error	

4.2.3 REGISTERS DESCRIPTION

In the last subsection was presented the commands available from MATLAB. The messages exchanged in this protocol are in the Appendix C.1 and the format of each one in the C.2. Here it is only explained the meaning, size and type of every register present in the messages.

- Seq Num: Sequential Number to identify the number of the message. Size 1 byte, type unsigned; it is the first register in all messages.
- ID: Identification command code. The corresponding value to the message request can be found on Appendix C.3. Size 1 byte, type unsigned; it is only used in the messages send by the client and appears always next to Seq Num.
- Checksum: The method used is the weighted sum of all bytes in the message in modulo of 256. To detect more errors the coprimes of 256 must be used for the weight value. The chosen values were {3, 5, 19, 61, 97, 127, 211, 251}. This method detects all single byte errors, and some transposition errors. Size 1 byte, type unsigned; it is the last register in all messages.
- Channel: Number of the channel of the requested operation. The channel count will start at 0. Size 1 byte, type unsigned.
- Voltage: Applied only to analog commands, is the voltage applied to DAC or received from ADC. Type floating point and follows the IEEE Standard 754 for single precision. Size 4 bytes.
- Boolean: Applied only for digital commands, 0 for LOW and 1 for HIGH. Size 1 byte, type unsigned.
- Frequency: Frequency in Hz of PWM to be set. The channel 0, 1 and 2 share the same timer, so any change in any of this channels will affect the frequency at the others. Channel 3 has an independent timer. Size 2 byte, type unsigned.
- Duty-cycle: Duty-cycle in percentage of PWM to be set. Type floating point and follows the IEEE Standard 754 for single precision. Size 4 bytes.
- Error: Number to describe the type of error. If zero, the operation was successful. For more information consult Appendix C.4. Size 1 byte, type unsigned.

4.2.4 PROTOCOL ON MATLAB AND RASPBERRY PI

MATLAB IMPLEMENTATION

MATLAB has built-in UDP protocols but it is not the most efficient implementation, and for this reason it was used the UDP library [47], with the advantage of being modifiable. In both implementations, when a message is sent, it is performed a socket open before and close after. This is not advantageous for the intended use, because opening and closing the socket adds time. So it was used a slightly modified version made by Rómulo Antão, PhD student of DETI [48] to keep the socket open between messages.

On the MATLAB side it was created a socket for the UDP communications. There are a set of parameters that are unknown for the programmer, and they have to be set by the user. So it was created the function 'udp_message()' that allows to configure the Remote IP and Port of the server, and the Port and time-out value which MATLAB will use. The syntax of this function is:

Syntax:	raspio = udp_message(Local Port, Remote IP, Remote Port, Time Out)	
Arguments:	Local Port:	Port on the side of Matlab, type int
	Remote IP:	IP address of Raspberry Pi, type string
	Remote Port:	Open port on Raspberry Pi for the UDP socket, type int
	Time Out:	Time waiting in milliseconds for receive a UDP package, type int
Return:	Object	

All the functions specified on Subsection 4.2.2 use the function 'transmit()' to handle the following errors: Sequence Number mismatch, wrong checksum, time-out and also to do the retransmissions. With the creation of this function, whose flowchart is in Figure 4.6, it is much easier to debug the communications and to extend the list of commands, as it is only needed to construct the buffer with the information, call this function and after that check if there were no errors and send to the user the information decoded.

Syntax:	transmit(obj, check_pos_send, check_pos_recv)	
Arguments:	obj:	inside of the object there is multiple information: from the buffer to the socket.
	check_pos_send:	index where the function has to add the checksum number in the buffer to send. It starts counting from 1.
	check_pos_recv:	index of the checksum in the received buffer to validate the content. It starts counting from 1.
Return:	0 for successful, other number for error)	

Function 'transmit()' has as arguments the object, the position of the checksum in the buffer to send and of the received buffer, in this order. The object contains multiple information such as: a pre-version of the buffer to send, current sequence number, UDP socket, number of tries to read and transmit. The function returns a value different of zero if there was any error; for more information about the error consult the Appendix C.4.

When the function 'transmit()' is called by another function such as 'setOUT()', in the beginning it has to increment the sequence number by one, fill it in the first position of the buffer, calculate the checksum and add to buffer in the position set by the second argument. Next will try to send the buffer; if there is no error it will try to read the incoming buffer, compare the sequence number and do the checksum. If every step taken does not give any error, the received buffer will be copied to the same place of the sent buffer, and the function returns zero. If when reading there is a time-out error, the function will do a retransmission; if the sequence number or checksum mismatches the function will try to do two more reads before doing a retransmission. After it performs a total of three transmissions and were all unsuccessful, the function returns the appropriated error. In the end, the caller function just have to check the value returned to see if there is no error, and retrieve the information from the buffer. The described behavior of the transmit function is show on Figure 4.6.

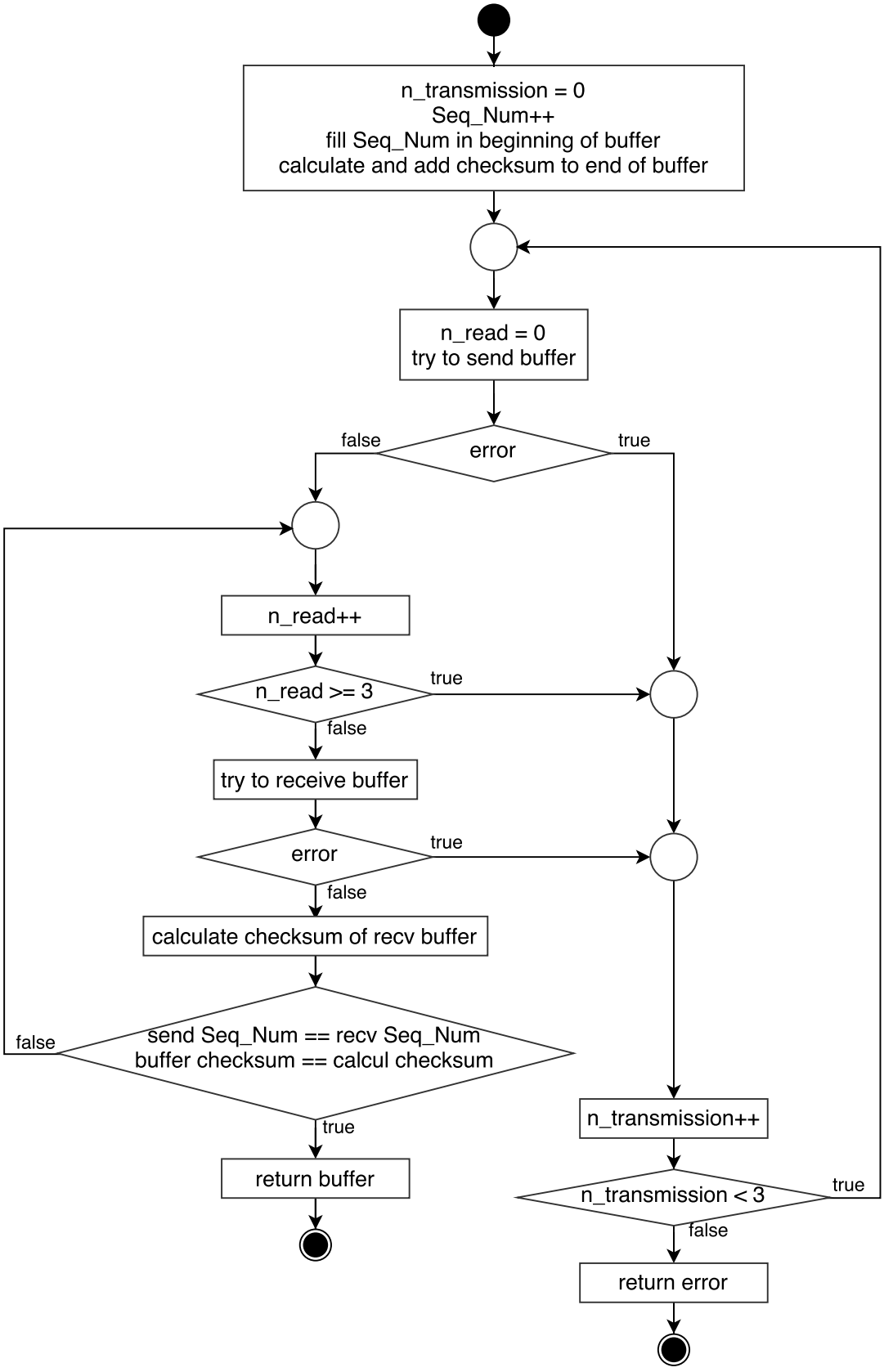


Figure 4.6: UDP transmission handler

RASPBERRY PI IMPLEMENTATION

In Raspberry Pi it was used a socket in Linux running on a Xenomai task, with priority of 98. The first step is to create the socket, and for that is configured the address family to IPv4, to service type datagram, and protocol UDP. The second step is to bind a address to the socket, this means choose the port number in the IP network, where the client will connect. After that, the socket is ready to start receiving messages. The function used to receive the messages is of the blocking type. This means the program will not continue the execution until it receives a message or until an error occurs.

The first message that the server expects to receive is the start command. If it receives any other command before, it sends a message with an error code to inform the client. After the start command the server will send the appropriated message following the guidelines on Subsection 4.2.1. Then the server will attend all the requests until the stop command is received.

When a message is received, is compared if the sequence number is greater or equal than the previous valid message. Next it decodes the ID and if it is valid, is used to know the position of the checksum, to see if the it matches the content of the message. If any of these conditions fail, the server will discard the message. If the sequence number, the ID and checksum are equal to the last previous valid message, the server knows that it has already processed the command. To speed up the process, the command will not be executed again and will only rebuilt the buffer with the information of the last call and sent it to the client. In case the sequence number is greater then the previous valid message, it will use the ID to decode the rest of the fields of the message, will execute the command, and with the result of the command (being successful or an error), will built the message to send to the client. Figure 4.7 presents a flowchart that describes the algorithm. Note that not all mechanisms to deal with errors are present in the figure.

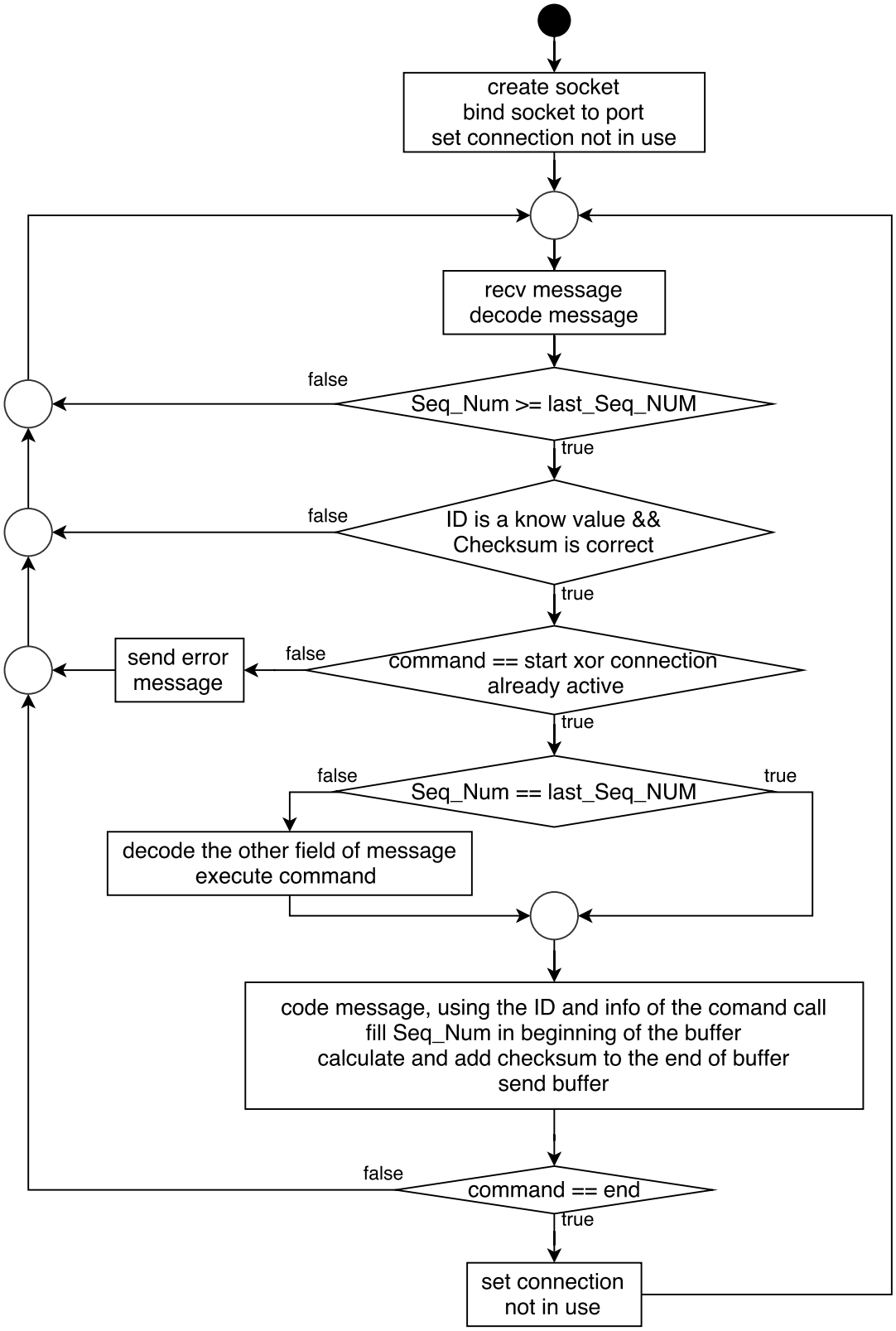


Figure 4.7: UDP server task

4.3 MATLAB CODE TRANSLATOR TO C

As mentioned before, the objective of this work is to be able to develop control algorithms in MATLAB and then allow its execution on the RasP:IO platform. To automate this process it was implemented a toolchain that translates the MATLAB algorithms to C code and carries out its compilation. This section describes the main components of this toolchain.

4.3.1 IMPLEMENTATIONS

As it was mention before, MATLAB is a multi-platform tool. Therefore, to make the code translator available to all OS, it was decided to make the code translator in MATLAB programing language.

The function responsible to translate the code is 'translate2C()'; in the arguments there must be a string with the name of the file with its termination. The function will try to find the file and open it. If successful it will do a first read to learn the name of the variables and arrays inside of the given .m file. Afterwards, it will do a second pass for accomplish two things: to translate the name of the functions to C and insert the parameters and to correct the syntax.

The function responsible to translate and do the match from MATLAB to C is in the file 'fixFunction.m'. Here there is a structure with one function to match per line and at least five fields to fill. The first two fields regard to the function in MATLAB; first if the function returns something it has to be field with 'ret' if not it must be let empty ' '; second is the name. The third, fourth, and fifth parameters regard to the equivalent function in C; the third is 'ret' or empty ' ', depending if the function returns something, the fourth is the name and the fifth is the argument. Inside the argument string, the code will try to find the expression 'arg' with a number from 1 to 9; this means that in the function translated to C there will be the argument number, found in the string with the rest of the expression. If any more arguments are needed, there can be added following the example on Figure 4.9. For example, to convert the expression 'set_periodic(0.01)' to make a loop with a period of 10ms, the function resulting from translation would be 'rt_task_set_periodic(NULL, TM_NOW, 0.01*1000000000)'.

The second point to correct, as it was said before, is the syntax, and in this case is necessary to correct the parenthesis of the index of arrays, making them beginning in zero and not one as MATLAB does; correct the control flow statements and treat the copy and fill of arrays. Figure 4.8 presents an high level flowchart of the translation of code.

In the end, the program will print the code translated in the MATLAB's console with the appropriate warnings when it was not capable to do the translation, and it will print a file in the same directory. The file is opened automatically in a text editor to give the opportunity to the user to correct the code. Afterwards, it is inquired in the console if the user wants to send the C code to Raspberry Pi using a ssh/scp session, and if he wants to compile the code. If there are no errors in the compiling process, it is prompted to run or not the code, if there are errors or warnings during compilation it will be shown.

On the Raspberry Pi was created a program that initializes the SPI communications, creates and starts a Xenomai thread with the name 'Control Loop' and priority 97. In the beginning of the launched thread, is verified if the SPI device driver is working and if the IC used to give the digital and analog I/O is connected. This is done by verifying if it is possible to read the ID of the IC and if it matches with the value in its datasheet. After performing the verification, and if it was successful, it

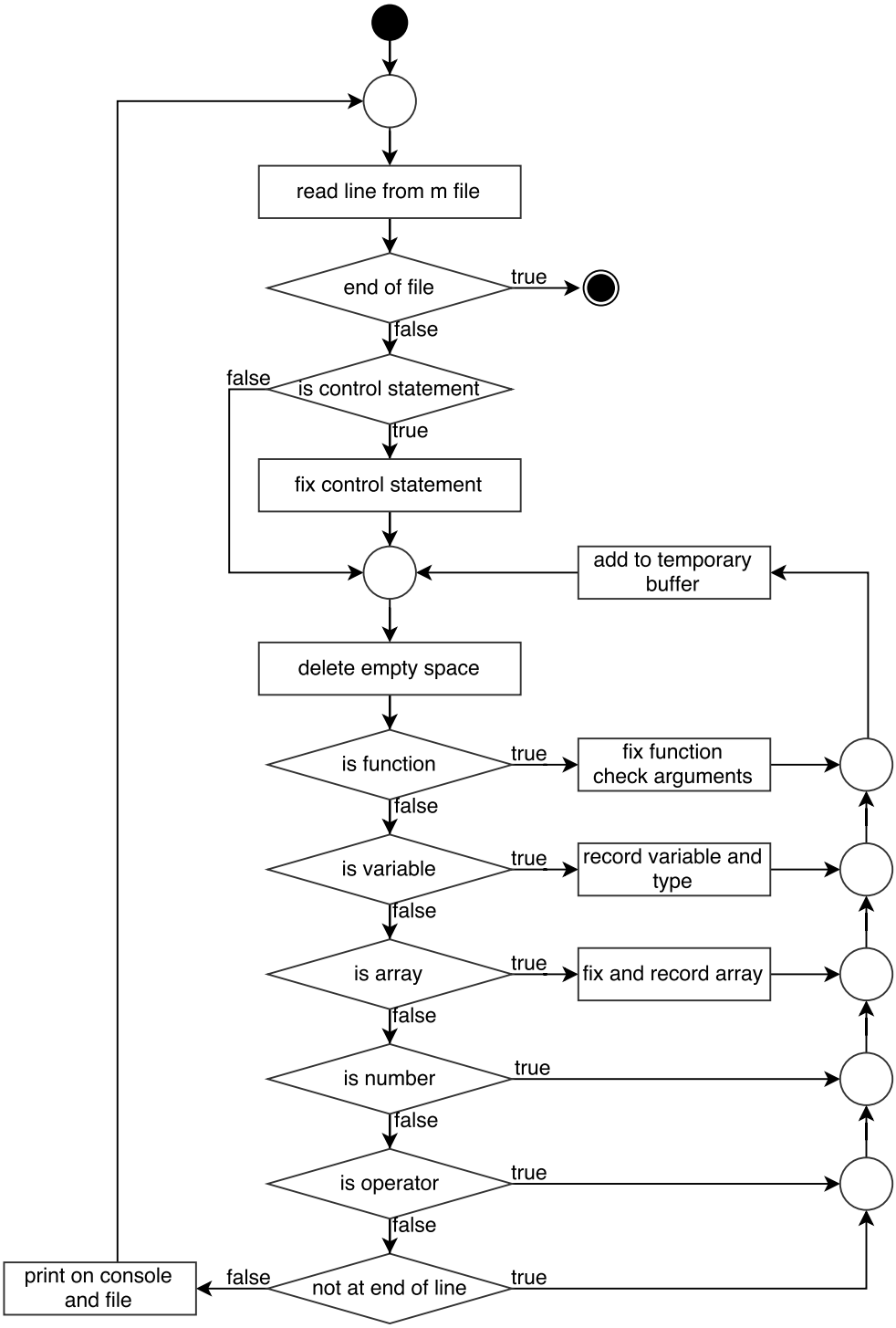


Figure 4.8: Code translator flowchart

is called inside the Xenomai thread the function 'control_task_body()' that is prototyped on the file 'control.h' generated by the code translator developed in this section. And is ready to compile and run.

Sintax:	translate2C(file_name)
Arguments:	file_name: string with the file name and the termination
Return:	None

MATLAB FUNCTION		C FUNCTION				
return	function_name	return	function_name	arg1	arg2	arg3
'ret'	'obj.getADC'	' '	'getADC'	'(uint8_t)arg1'	'&ret'	
' '	'obj.setDAC'	' '	'setDAC'	'(uint8_t)arg1'	'arg2'	
' '	'set_periodic'	' '	'rt_task_set_periodic'	'NULL'	'TM_NOW'	'arg1* 1000000000'

Figure 4.9: Structure to convert functions. The complete structure can be found on Appendix F

4.3.2 RULES TO USE

With the time available it was not possible to do a more complete code translator to work in all the cases. For this reason some guidelines must be followed:

- First, all lines of code before the instruction:

  "udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT)" will be ignored.

- It has to exist one and only one 'start()' and 'stop()' command in the m file.
- To make a function periodic, the function "set_periodic(period_s)" has to be used with the time in seconds, in conjugation with the function "wait_period()", normally inside the loop.

The program has a set number of limitations that can be found in the next list:

- The code translator will not translate the inside of the functions. Instead it will try to find an equivalent function to C.
- It is only allowed one instruction per line.
- A function can not be called in one of the arguments of another function.
- Control flow statements allowed: if, else, while and for.
- It is not allowed any arithmetic operation with matrices and arrays.
- Operations allowed with arrays: only 'zeros()'.
- The arrays have to maintain the same size in run-time.
- The exponential operator must be avoid.

Note: When the code translator when declares the variable in C it will try to get the values from the MATLAB's workspace. This can be useful to define the initial value to variables where there is no other way to set the value in C language.

4.3.3 CODE EXECUTION ON THE RASPBERRY PI

The translator code inquires the user with multiple questions, to send, compile and run the code. To perform these operations is used a ssh/scp session [49] executed on MATLAB. Although the user has some control when it runs the function code translator, it was created the function 'ssh_action()' to give more options to the user. Above this code was created an interface with all the options needed. It is possible to send a file, compile, run or stop the control code, and run or stop the server protocol.

Here are the multiple options for using the function 'ssh_action()':

- To send a file to Raspberry Pi, which can be the file generated by the code translator or other. Argument FILE NAME is a string containing the file name with the extension. Nothing is returned.

  ssh_action('SEND FILE' , FILE NAME)

- To compile the file "control.c" in conjugation with the body of the control loop sent by MATLAB. Returns the warnings or errors resulting from the compilation.

  ssh_action('COMPILE')

- To run the compiled code from the option describe above. The function will only be concluded when the control execution finishes. Returns everything is written to the Linux console by the program.

  ssh_action('RUN CONTROL')

- To force the code to stop running. Returns a string confirming that the program has stopped.

  ssh_action('STOP CONTROL')

- To run the UDP server task. Nothing is returned.

  ssh_action('RUN SERVER')

- To stop the UDP server task. Nothing is returned.

  ssh_action('STOP SERVER')

- To shutdown the server. Nothing is returned.

  ssh_action('SHUTDOWN')

4.4 SUMMARY

In this chapter about software and firmware was present the SPI device driver and the reasons that justified the option for the BCM2835 open-source device driver. Next it was presented the HAL of MAX11300, the IC responsible for the analog and digital I/O and the parameters to configure this IC. In the second section, was defined the UDP based communication protocol for allowing the

communication between the IC and the Raspberry Pi, as well the library used and functions created. At last, was present the code translator from MATLAB environment to C, the set of rules needed to use, its limitations and the functions to operate the Raspberry Pi.

CHAPTER 5

Experimental Results

In this chapter it will be presented the tests performed to the platform. First it is presented the more technical tests where it is shown the results of the function and temporal tests to the communication protocol and to the RasP:IO interface.

Afterwards it will be shown a RST controller with RLS for identification of the system. With this controller it is intended to show the performance of RasP:IO working as an interface to MATLAB and also as an independent embedded system. As consequence, the communication protocol is also tested with a real example, and the same happens to the code translator. The next test is performed with a PID controller with relay feedback, where it is important to measure the time correctly for the calculation of the PID parameters.

In the end it is performed a simple test to the code translator. This test also intends to show the good practices when making use of the translator.

The following tests were performed with MATLAB running on Windows 10 64 bits, in a computer with 4Gb of RAM and the CPU i3-330UM. The results may vary in a different computer.

5.1 Measurements and validation tests

5.1.1 Performance of communication protocol

To test the protocol implemented between MATLAB and Raspberry Pi was created multiple scripts in MATLAB to test each command 200000 times. Each command inside the scripts is executed at maximum speed permitted by MATLAB without any delay between them. The scripts for the tests can be found in Appendix D, but briefly follow these rules:

- the channels are generated randomly inside the valid values, when applied;
- in the analog commands the value sent or received has to describe a ramp from -5V to 5V and back to -5V;
- in the digital commands the sent or received values are the or operation of the least significant bit between the value of the channel and the sequence number of the communication protocol.

For the tests, it was used the following settings:

- Local Port: random value between 8000 and 50000;
- Server IP: 'raspberrypi.mshome.net';
- Server Port: 10000;
- Time-out: 8ms.

Function Name	Mean time	Standard deviation	Maximum time	Number of errors
getADC	2.36ms	0.52ms	23.28ms	6
getADCALL	2.63ms	0.39ms	16.56ms	0
setDAC	2.34ms	0.56ms	25.96ms	4
setDACALL	2.47ms	0.58ms	22.82ms	8
getIN	2.35ms	0.53ms	24.23ms	4
getINALL	2.61ms	0.65ms	35.62ms	5
setOUT	2.34ms	0.54ms	19.88ms	5
setOUTALL	3.03ms	0.85ms	39.79ms	4
setPWMFreq	2.44ms	0.52ms	24.56ms	4
setPWMDuty	2.36ms	0.55ms	24.55ms	4

Table 5.1: Execution time of each function on MATLAB

Most of the error communications that happened were in the MATLAB side when it was waiting to receive a message or a acknowledge. On Raspberry Pi it was not detected any missed reception, neither any error on sending the message. The only way to ensure that the error registered is MATLAB/OS's fault would be putting a switch in the middle of the communications and check every message. As the errors are not so frequent this can be hard to realize and was not done as the results were satisfactory.

5.1.2 TEMPORAL ANALYSIS OF DIGITAL AND ANALOG I/O

To test the time of execution of each function created to read and to write digital and analog values on the interface, it was created a periodic Xenomai thread running at a frequency of 1000Hz, that executed each function sequentially each time for 200000 times and registered the time in each call. For more information about the meaning of each function consult Subsection 4.1.2.

With the times registered, it was calculated the mean, standard deviation and maximum values that are presented on Table 5.2. A short analysis of the time measured of each command was performed, to ensure that its execution was correct. Command getIN and getINALL were taken as base values, for being the ones that only perform one transaction on SPI.

- The setOUT and setOUTALL take twice the time, because it is made a 32 bits read operation and then a 32 bits write operation on SPI.
- The setDAC only does a 16 bits write and has the value higher than the other operations. The possible reason is that it has to first perform some floating point operations to convert the voltage to a binary value.

- The highest value is from setDACALL, and the reason is that it has to perform four times the floating point conversion and the 16 bits write to SPI, with 80us delay between every write.
- The getADC has the mean value within the expected value, as it also performs floating point conversions, reads 16 bits and 32 bits registers, and in the end it reads the actual value, which is a 16 bits read operation.
- The getADCALL has the same steps, with the difference that it has to do four floating point conversions, and in the end four times 16 bits read, but as the channels are in raw, they are all sent in the same operation, which does not add much time.

Function Name	Mean time	Standard deviation	Maximum time
getADC	83us	9us	143us
getADCALL	87us	9us	152us
setDAC	77us	7us	120us
setDACALL	326us	7us	372us
getIN	26us	3us	56us
getINALL	26us	3us	53us
setOUT	45us	5us	80us
setOUTALL	46us	5us	84us

Table 5.2: Execution time of each function on Raspberry Pi

During the execution of these tests, it was not registered any errors. The maximum value takes around twice the time of the mean value, and could only be improved with real-time drivers for Xenomai.

5.1.3 FUNCTIONAL ANALYSIS OF ANALOG I/O

In this subsection it was tested the accuracy of the DAC and ADC. With the channel 0 of the DAC connect to the channel 0 of ADC, it was measured the voltage from the DAC with the multimeter FLUKE 287 and compared the voltage with the value read from ADC. The multimeter in use as an accuracy of 1.25mV with in the range -5V and 5V [50], better than the resolution of the MAX11300 that is of 2.44mV.

The test starts with the DAC set to -5V and were made increments of 200LSBs (about 488mV) until 5V. In every step it is made 1000 read with the ADC, but before starting to read it is waited one second.

Figure 5.1 has the difference between the set voltage on DAC and the read voltage of multimeter. The mean value of the difference is 4.61mV, which means that the DAC has a systematic error of about 2LSBs.

Figure 5.2 shows the difference between the value measured by the multimeter Fluke 287, which is being used as reference, and the mean voltage get from the 1000 reads of ADC. From this 1000 reads it was obtained the maximum and minimum value (Figure 5.3), and it shows a dispersion of about 10mV or 4LSBs. When only considering the voltage range between -4.5V and 4.5V it is possible to

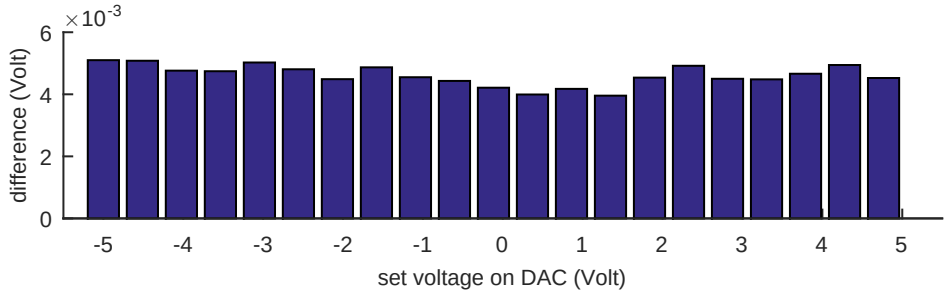


Figure 5.1: DAC offset voltage

deduce that the ADC suffers from a gain error, in other words, the linear gain is slightly different from one. At -5V the ADC registers an offset of 5mV with opposite signal. From the datasheet [33] these offset voltages are within specifications.

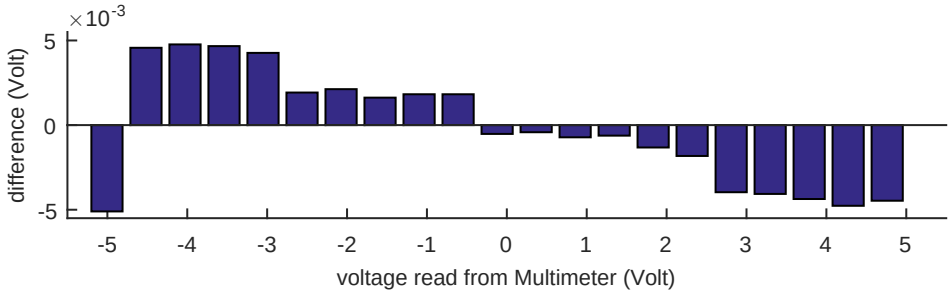


Figure 5.2: ADC offset voltage

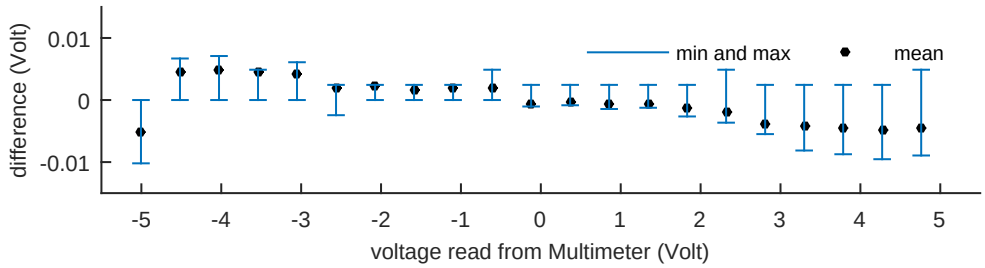


Figure 5.3: Error bar graph with minimum, maximum and mean value difference to the multimeter

Figure 5.4 is a more interesting graph for control applications. It shows the difference between the ADC and DAC, as this is the offset voltage that the controller will have. In the case that the closed-loop has a high gain, this offset voltage will not be significant. At -5V the difference between the ADC and DAC is zero, but in the next recorded value it jumps to 10mV, about 4LSBs. Afterwards, it continues to decrease until the difference reaches 0V, at about 4.76V.

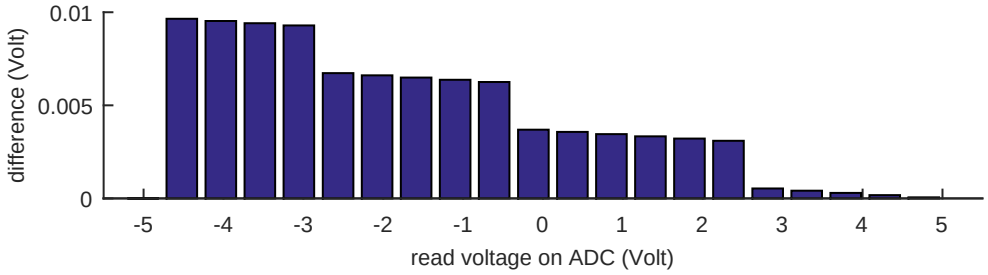


Figure 5.4: Difference between set voltage on DAC and read on ADC

5.2 RST CONTROLLER

In this section it is used the RST controller with the Recursive Least Square (RLS) algorithm to make the system identification. The plant in use has to have two inputs, one for the input and the other to emulate load on the system, and one output. It is intended to apply the stated algorithms in a first and second order plant. To meet the set requirements, the OKAWA Electric Design [51] was used to generate a second order Sallen-key low pass filter with a damping ratio approximately of 0.4, rise time of 0.5 seconds and unitary gain. The electronic circuit is presented on Figure 5.5 with the addition of two more op-amps, the first in a summing configuration to have the two inputs needed. As this configuration has a gain of -1, it was added in last an op-amp in inverting configuration to make the gain of the plant equal to 1. The components values are stated in the figure.

The input of the circuit is connected to the channel 0 of the DAC, the load to channel 1 of the DAC, and the output to the channel 0 of ADC. In the tests made where the load is not used or mentioned, it was left connected to the DAC set to 0V.

The RLS algorithm is initialized in all tests, both in MATLAB and Raspberry Pi, with all the estimated coefficients $\hat{\theta}$ with 1 and the covariance matrix P with the matrix identity multiplied by 100. By doing so, it is not given any confidence to the set $\hat{\theta}$, but it will converge quickly to the right value. Both algorithms, identification and control, are running at a frequency of 20Hz.

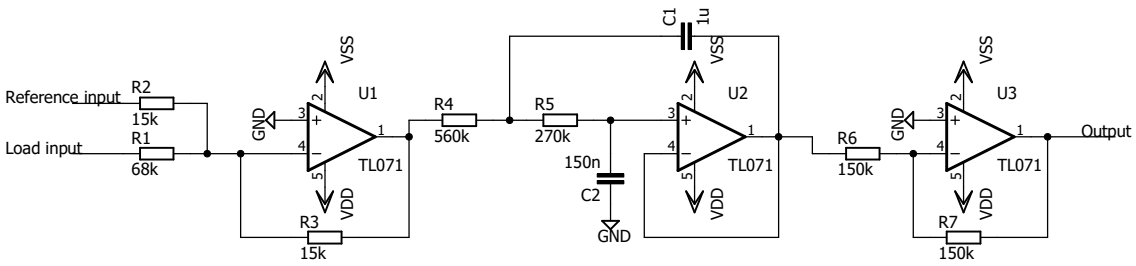


Figure 5.5: Sallen-key second order low pass filter

5.2.1 FIRST ORDER SYSTEM

As first order plant it was used the circuit present in Figure 5.5 with the C1 replaced by an open circuit.

RLS IDENTIFICATION METHOD

To excite the system it was used as reference signal a square wave with amplitude of 2V centering in 0V and period of 6 seconds. At the same time it was running the algorithm of identification and it was recording both the output signal and the evaluation of the estimated model coefficients, that are represent on Figure 5.6.

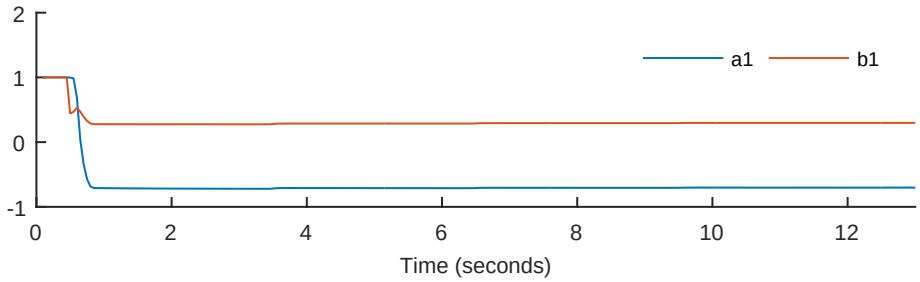


Figure 5.6: Estimated coefficients using RLS on first order system

With the steady values of the estimator it was found the equation of the model of the system 5.1, exciting the model with the signal and compared against the physical one. The result can be found on Figure 5.7.

$$G(q^{-1}) = \frac{0.297q^{-1}}{1 - 0.7031q^{-1}} \tag{5.1}$$

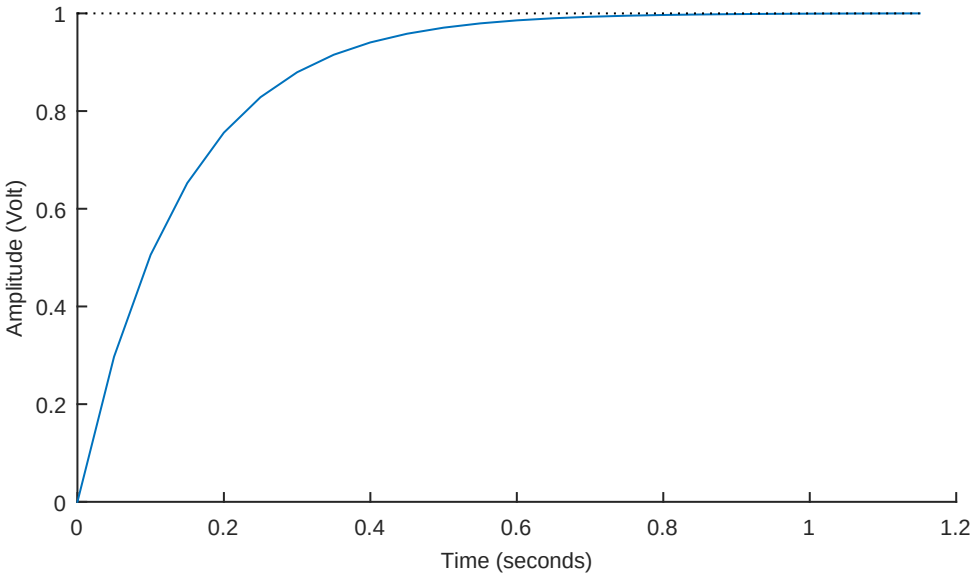


Figure 5.7: Step signal of the first order model

Taking in account the time constant of the plant and the step response of the model, it is likely that the RLS algorithm is working correctly. But there is not enough data to ensure the identification is 100% correct.

RST CONTROLLER ON MATLAB

The reference signal used runs multiple times between 2V and -2V ending in 0V, to prove the correctness of the controller. Approximately at the 14^{th} second was added load to the input of the system to see its immunity to perturbations. The behavior of the closed-loop system was selected to have a system with unitary gain and a rise time of 0.42 seconds, and it was obtained by choosing the system, represented in the Equation 5.2. The observer pole was set at -14, to be two times faster than the pole in closed-loop. In Figure 5.8 there is the performance of the RST controller with RLS identification at run time, executed in MATLAB.

$$G_m(s) = \frac{7}{s+7} \tag{5.2}$$

At the first second the output of the system overshoots the setpoint as the estimated coefficients of the model by the RLS algorithm are only correct and stable after one second, already demonstrated in the Figure 5.6. The closed-loop tracked correctly the reference signal and was quick to converge to the setpoint again when it suffered the perturbation at the 14^{th} second, around 0.9 seconds.

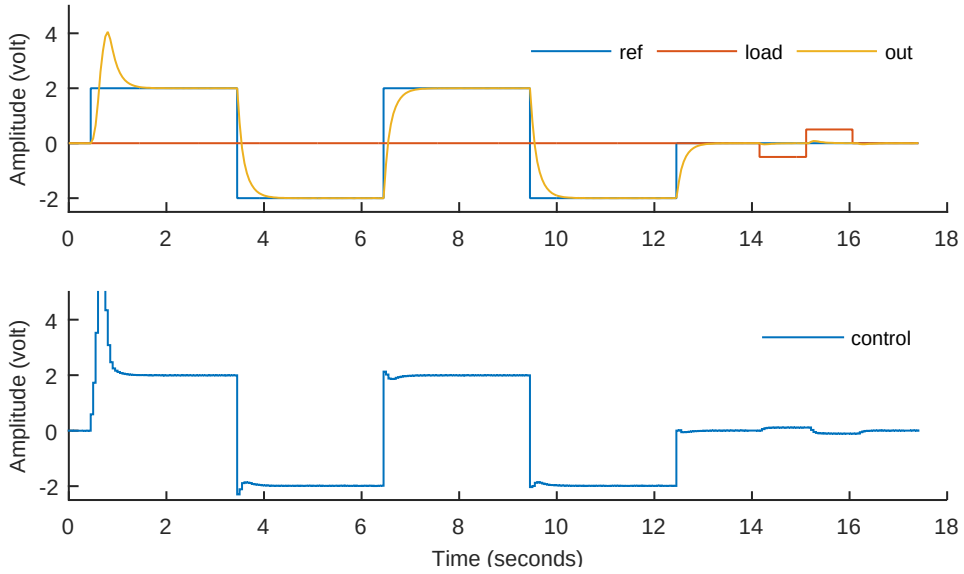


Figure 5.8: Close loop first order with RST controller and RLS identification on MATLAB

During the execution of the control loop, it was recorded the time between each period, the duration of the functions 'getADC()' and 'setDAC()' and the time took to perform all the calculations, using the functions 'tic()' and 'toc()' from MATLAB. The times measured in each cycle of the control loop can be seen on Figure 5.9. From the analysis of this figure, it is visible that in the beginning of the control algorithm, first three iterations, the time measured was more than two times the mean value. As this happens in beginning it is possible to reduce the impact of this in the performance of the control loop just by starting to execute the control later. For this reason it was calculated the mean, standard deviation and maximum values, starting from iteration number four. The result are at Table 5.3.

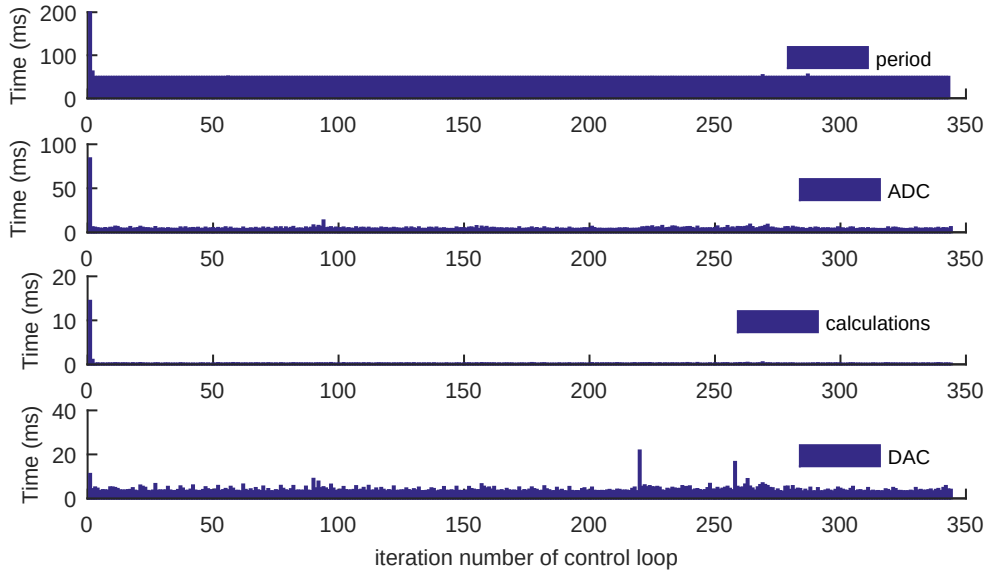


Figure 5.9: Measured time on each iteration of the control loop on MATLAB

	Mean time	Standard deviation	Maximum time
Period	50.035ms	0.354ms	55.251ms
ADC	4.806ms	0.999ms	13.588ms
Calculations	0.227ms	0.040ms	0.480ms
DAC	4.263ms	1.474ms	21.800ms

Table 5.3: Execution times of first order plant with RST and RLS on MATLAB

RST CONTROLLER ON RASPBERRY PI

Using the code translator, the code was translated, sent, compiled and run on Raspberry Pi. Therefore the procedure continues the same from the last subsubsection. It was used the function 'rt_printf()', the equivalent to 'printf()' for Xenomai, to send in the end of the execution the data visible on Figure 5.10 and Table 5.4. For register the time was use one special function from Xenomai, 'rt_time_read()'.

	Mean time	Standard deviation	Maximum time
Period	50.000ms	0.004ms	50.022ms
ADC	0.131ms	0.007ms	0.141ms
Calculations	0.042ms	0.001ms	0.047ms
DAC	0.095ms	0.010ms	0.106ms

Table 5.4: Execution times of first order plant with RST and RLS on Raspberry Pi

From the comparison of the output signal of Figure 5.8 and 5.10 it is possible to notice the initial delay that only happens in the beginning of the MATLAB execution. As stated before this delays are due to the initial long time of the three first iterations of the control loop on MATLAB. As it only

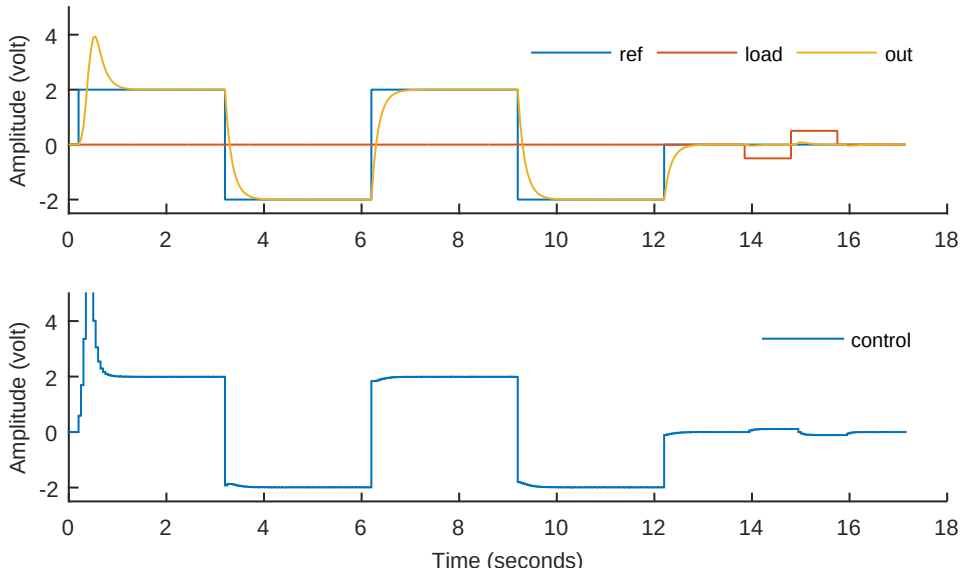


Figure 5.10: Close loop first order with RST controller and RLS on Raspberry Pi

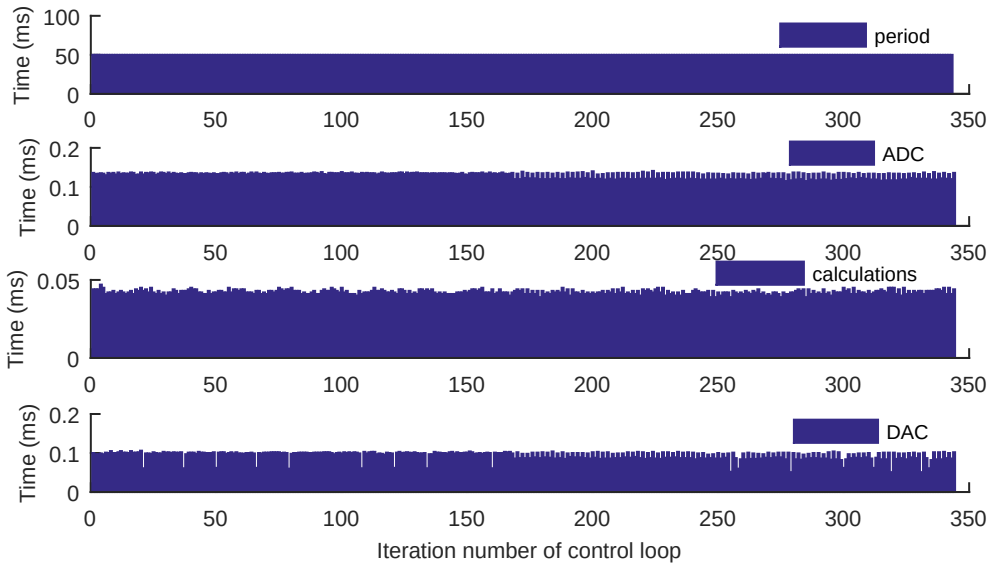


Figure 5.11: Measured time on each iteration of the control loop on Raspberry Pi

happens at the beginning of the performance of the controller, it is not significantly affected. In any case, the behavior observed on MATLAB is very similar to the one observed with Raspberry Pi. On the same figures, the difference between running in best effort system and in real-time system is more visible on the control signal, as it appears more smooth in the last. From the analysis of the Tables 5.3 and 5.4 is possible to verify this as well, as on MATLAB the standard deviations of each section of the code are much worse than on Raspberry Pi, and from the Figures 5.9 and 5.11 is also shown that the times on Raspberry Pi are more constant and shorter.

5.2.2 SECOND ORDER SYSTEM

For the test with a second order plant it was reinstalled the capacitor C1 on the electric circuit, as it is shown in Figure 5.5.

RLS IDENTIFICATION METHOD

Like before, in the reference input there is a square signal with amplitude of 2V. It was plotted the evolution of the estimated parameters using the RLS algorithm (Figure 5.12).

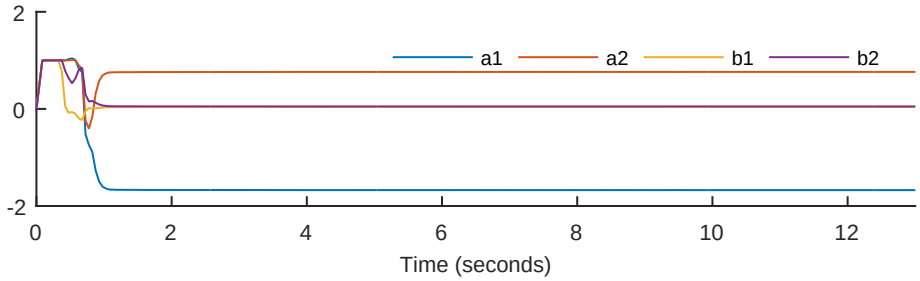


Figure 5.12: Estimated coefficients using RLS on second order system

From the steady values of the estimator it was calculated the equation of the model (Equation 5.3), and compared the model response of the physical system (Figure 5.13).

$$G(q^{-1}) = \frac{0.03887q^{-1} + 0.05084q^{-2}}{1 - 1.673q^{-1} + 0.7631q^{-2}} \tag{5.3}$$

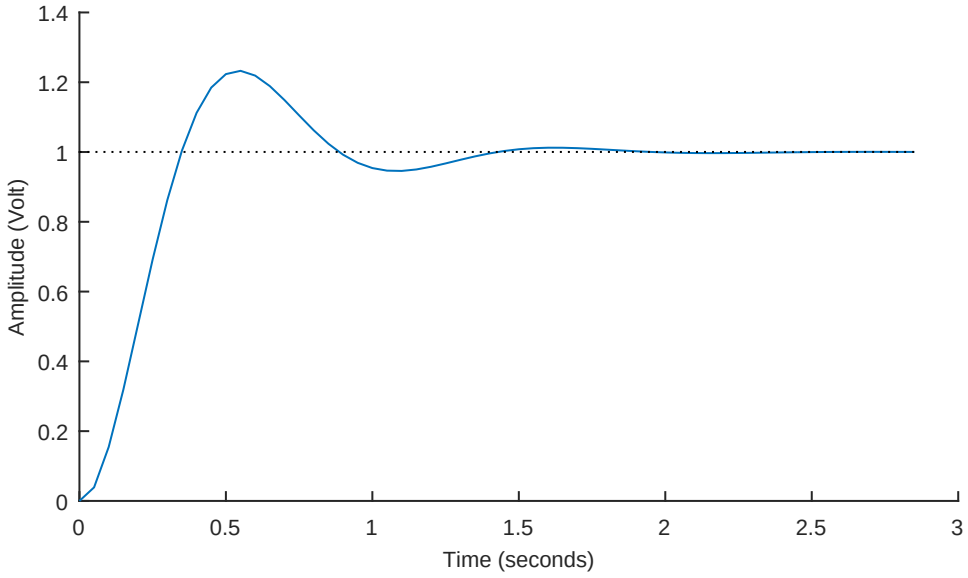


Figure 5.13: Step signal of the second order model

Comparing the step signal (Figure 5.13) with the transient analysis present on Appendix E, it is possible to confirm the correctness of the identification algorithm. Although when it was executed the

RST with RLS for the first order, the data obtained let us reach the same conclusion. However, this way we also ensure that the RLS works for second order plants.

RST CONTROLLER ON MATLAB

The reference signal continues the same from the last time, running multiple times from 2V to -2V and back to 2V, finishing in 0V. Again the load signal is the same, it starts approximately at second 14^{th}, going to -0.5V, then to 0.5V and finishing at 0V. As the estimated coefficients of the model only are correct and stable after one and half seconds, as showed on Figure 5.12, the controlled system is only capable to follow the reference signal after that moment, as shown in Figure 5.14. The closed-loop system was select to have the natural frequency of five rads and damping ratio of one, to have a non oscillatory behavior. On Table 5.5 there is the execution time of each part of the code; as previously the first three iterations were discarded for being much superior to the mean value, and for happen in the beginning when then controller was not operate.

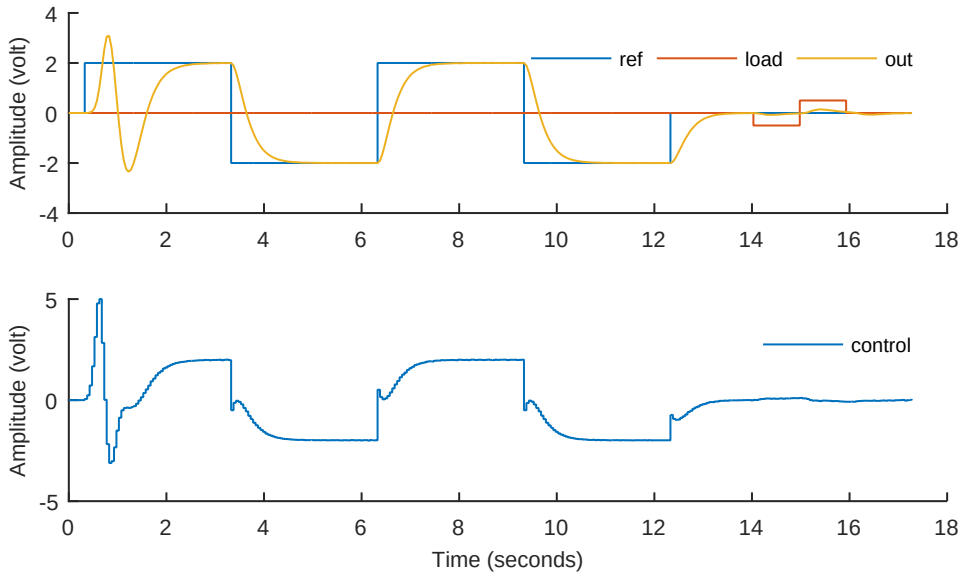


Figure 5.14: Second order RST controller on MATLAB

	Mean time	Standard deviation	Maximum time
Period	50.015ms	0.084ms	51.119ms
ADC	4.422ms	0.676ms	7.459ms
Calculations	0.212ms	0.087ms	1.698ms
DAC	3.483ms	0.554ms	7.369ms

Table 5.5: Timing for second order RST controller on MATLAB

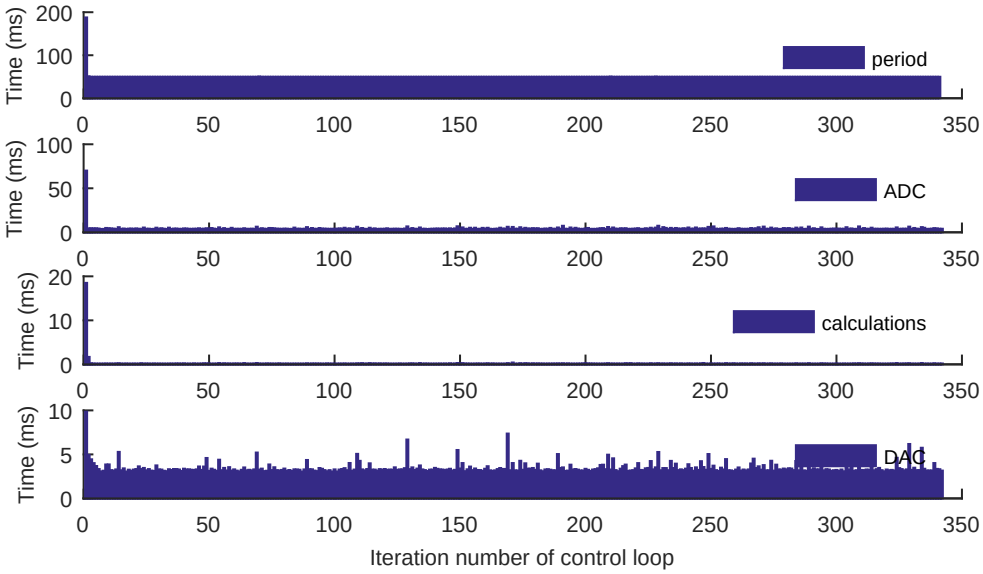


Figure 5.15: Measured time on each iteration of the control loop on MATLAB

RST CONTROLLER ON RASPBERRY PI

The procedure is repeated on Raspberry Pi. It was used the code translator to automate the process of translation and compiling in the Raspberry, making only small changes in the code to print the output, reference, load and control signal of the system in closed-loop (Figure 5.16), and the execution time of each part of the controller, (Table 5.6).

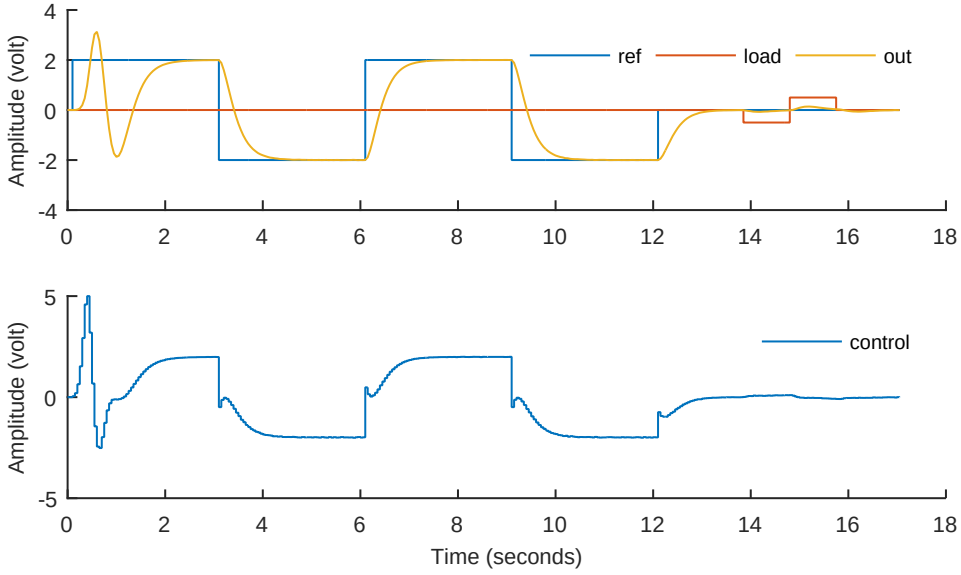


Figure 5.16: Second order RST controller on Raspberry Pi

As it happened for the first order plant, there is also an initial delay that was already discussed in the previous section. The execution time on Raspberry Pi (Table 5.6 and Figure 5.17), continues to

show more stable time than MATLAB, with shorter execution times, less jitter, and maximum values not far from the mean values.

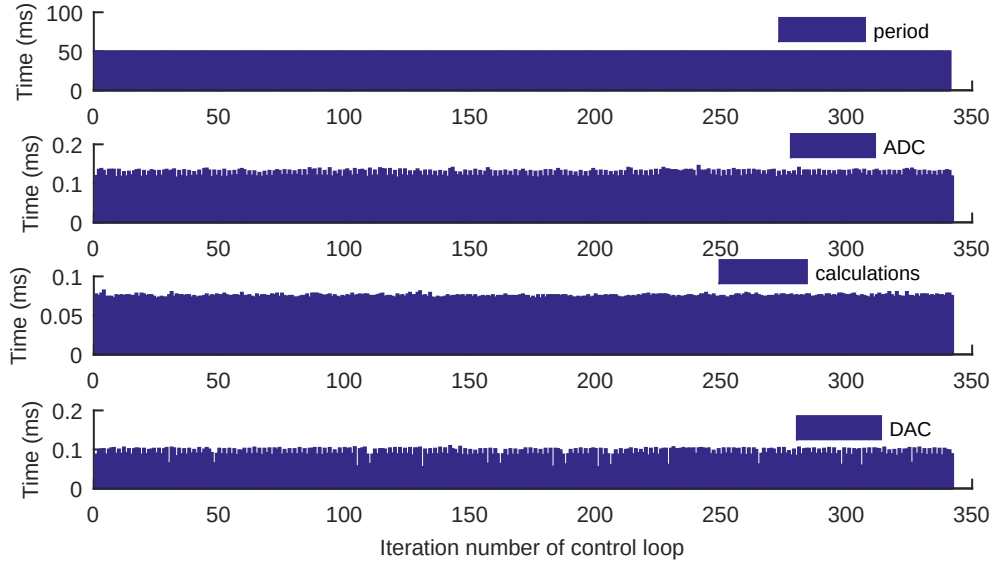


Figure 5.17: Measured time on each iteration of the control loop on Raspberry Pi

	Mean time	Standard deviation	Maximum time
Period	50.000ms	0.006ms	50.028ms
ADC	0.128ms	0.007ms	0.145ms
Calculations	0.075ms	0.002ms	0.082ms
DAC	0.095ms	0.010ms	0.109ms

Table 5.6: Timing for second order RST controller on Raspberry Pi

5.3 PID CONTROLLER WITH AUTO TUNING

One of the control algorithms mentioned in Section 2.4 was the PID auto-tuning with relay feedback. To employ this technique of tuning, the plant under test has to have an excess of poles that allow to put the system in oscillation. For this reason it was chosen a third order system.

5.3.1 PHYSICAL SYSTEM

$$G(s) = \frac{210.4}{s^3 + 18.79s^2 + 112.4s + 210.4} \tag{5.4}$$

Before starting with PID controller, it was made a study about the plant to see if it would be possible to bring to oscillation with the RasP:IO interface. First, using the command margin() on

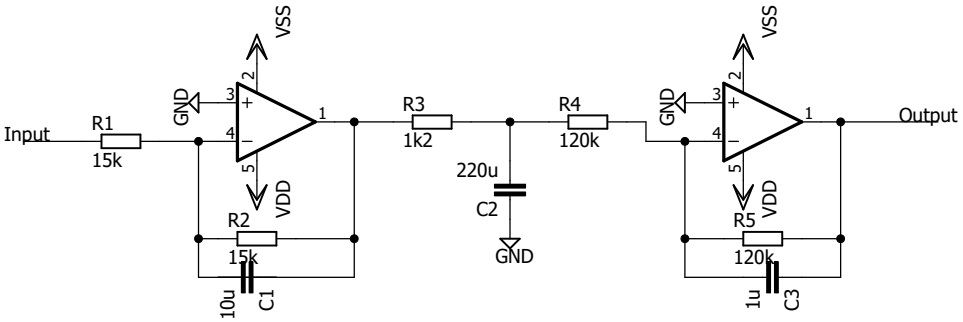


Figure 5.18: Third order plant

MATLAB, was discovered the gain that would make the system marginally stable on closed-loop, as well as its frequency of oscillation. From the command was obtained the gain of 9.03 and a period of 0.59s. The Figure 5.19 shows the model in closed-loop, oscillating at its natural frequency. Now that it is known that the oscillation period is superior to 0.5 seconds, the experience can continue. The bigger the oscillation period, lesser will be the frequency required for MATLAB to perform correctly the control loop.

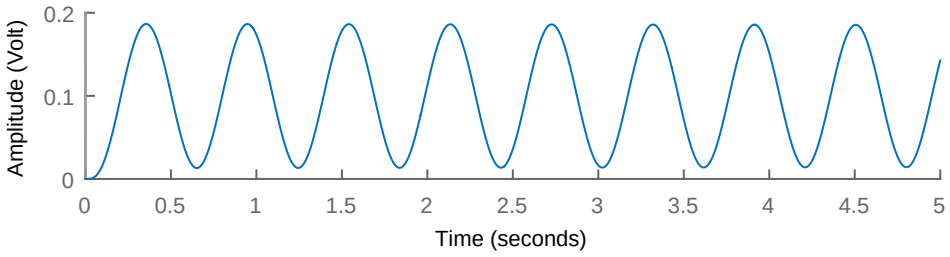


Figure 5.19: Theoretical system response to step signal

5.3.2 PID CONTROLLER

The relay feedback method works in two phases: first, the system is put in oscillation with a ON-OFF controller; to obtain a precise period and amplitude, the sampling rate was configured to 60Hz. The second phase activates when the system output reaches 0V and the ON-OFF controller is switched for a PID controller working at 20Hz.

The algorithm created to detect the period stopped the execution at 2.5 seconds, that was when it detected two consecutive oscillations periods with a difference of less than 0.5% and two consecutive oscillations with a difference in the amplitude of less than 0.5%. The values returned by this method are at the Table 5.7.

From the formulas of Table 2.4 it was calculated the parameters for the PID, that are in the Table 5.8.

After the algorithm switches to the PID controller, it is applied to the input of the controller the reference signal, 0V, and then goes to 2V during three seconds, comes back to 0V and stays there

parameter	value
T_u	0.65ms
a	1.604V

Table 5.7: Period and amplitude obtain from ON-OFF controller

parameter	value
K_p	4.764
T_i	0.325
T_d	0.081

Table 5.8: Tunned parameters from the relay feedback method

for three more seconds before it stops the control. The result of this method can be see in Figure 5.20. The control signal resulting from the tunned parameters, in dash blue, has a maximum and minimum values much larger than the supported by the RasP:IO. This causes the system to take longer to reach the setpoint. It is also noticeable that the overshoot value of the system's output, in some cases, reaches the double of the reference signal. This behavior is expected due to the tuning by Ziegler-Nichols.

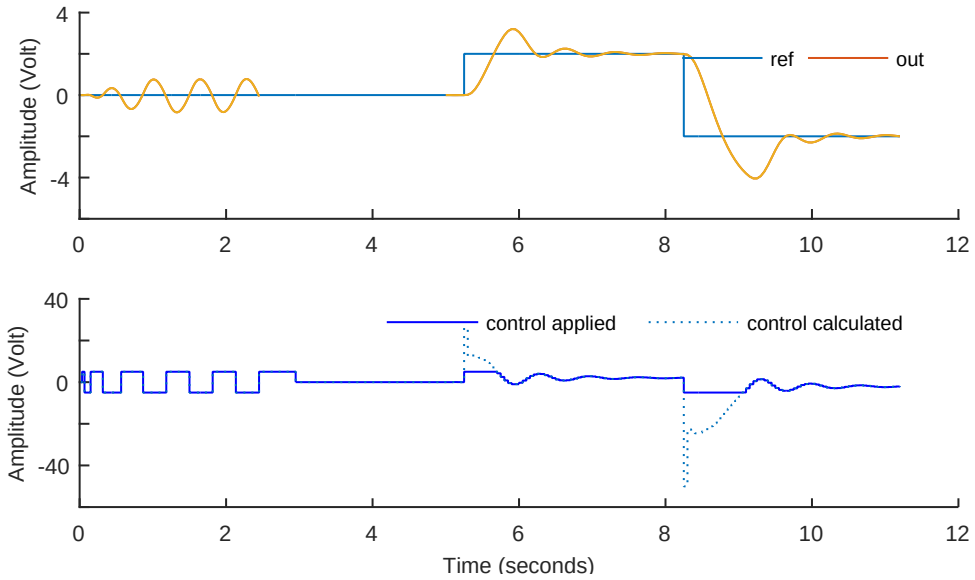


Figure 5.20: PID with relay feedback

5.3.3 I-PD CONTROLLER

To test the I-PD controller it was used the parameters obtained in the last subsection. The result of the test can be found on Figure 5.21. As it was expected because of the controller nature, it does not overshoot when the setpoint changes, because the error value is only feed on integrator, as opposite to the PID, in which the error is feed to the three parts of the controller.

This controller is a good improvement in relation to the last one, as the system becomes faster to converge to the reference input, and without overshoot, which in some applications is not allowed to happen.

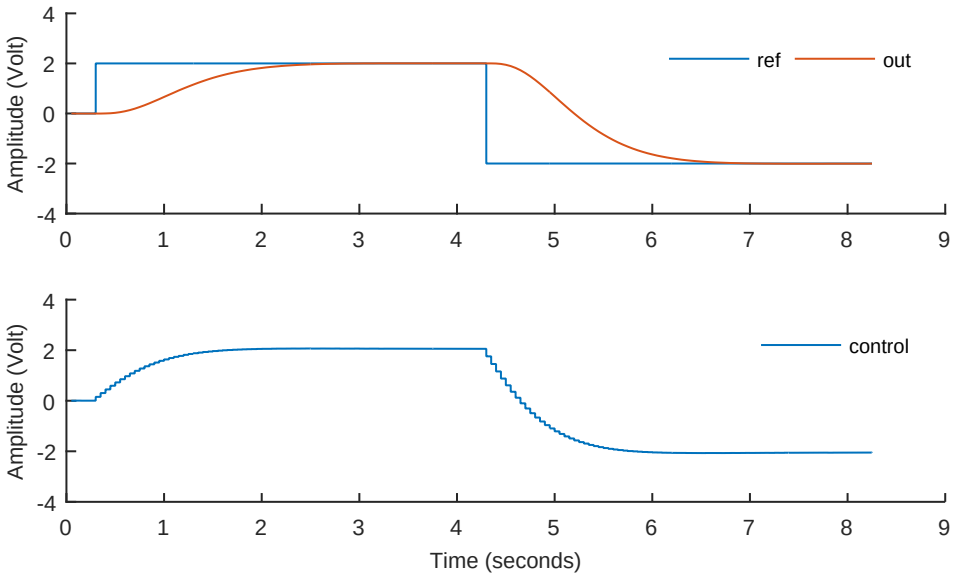


Figure 5.21: Third order I-PD controller on MATLAB

5.4 MATLAB TO C CODE

In the Section 5.2 the code translator was already used to run the control algorithm in Raspberry Pi and it was working correctly. Here it will be presented one simple example of a proportional controller that can be run as it is on MATLAB or translated to C code. This section will not only serve to show the result of the code generated in C, but also to address with an example of the structure that should be followed when using this software.

First it should be written all the code that will not be translated by the software, and it has to be executed in the workspace of MATLAB by the user. In this case is the following code:

```
r = [0 2*ones(1,20)];
fim = length(r);
u = zeros(fim,1);
e = zeros(fim,1);
y = zeros(fim,1);
```

Next is the code needed to set the communications with RasP:IO. This code will also not be translated by the software, as it will only start to translate the code after the line with 'udp_message'.

```
LOCAL_PORT   = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP    = 'raspberrypi.mshome.net';
```

```
REMOTE_PORT = 10000;
TIMEOUT     = 9;    % Depending on the transmit period of the remote device
    (ms)
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT);
```

Now the software will start to decode and translate the code to C.

To avoid errors, it is recommended to write all the variables in the beginning of this part.

```
h = 1/20;
Kp = 2;
r;
fim = length(r);
u = zeros(fim,1);
e = zeros(fim,1);
y = zeros(fim,1);

raspio.start();
set_periodic(h);
for k=2:fim
  wait_period();
  y(k) = raspio.getADC(0);
  %control algorithm
  e(k) = r(k)-y(k);
  u(k) = e(k)*Kp;
  %saturation
  if( u(k) > 4.5 )
    u(k) = 4.5;
  end
  if( u(k) < -4.5)
    u(k) = -4.5;
  end
  raspio.setDAC(0,u(k));
end
raspio.setDAC(0,0);
raspio.end();
```

The result of the code translator his present next. After it finishes to print the code to the MATLAB console, it will also open in a text editor, to let the user do the necessary corrections. Then it will present multiple questions to the user, such as: if the user wants to send and compile the code on Raspberry Pi and if he wants to run the code.

```
void control_task_body(void){
  float h;
  float r[] = { 0, 2.000000, 2.000000, 2.000000, 2.000000, 2.000000, 2.000000,
      2.000000, 2.000000, 2.000000, 2.000000, 2.000000, 2.000000, 2.000000,
      2.000000, 2.000000, 2.000000, 2.000000, 2.000000, 2.000000, 2.000000};
  float Kp;
  int fim = 21;
  float u[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  float e[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

```
  float y[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  int k;

  h = (float) 1 / 20;
  r;
  Kp = (int) 2;
  fim;
  u;
  e;
  y;
  MAX11300init( );
  rt_task_set_periodic( NULL, TM_NOW, h*1000000000 );
  for( k = 2 ; k <= fim ; k++ ) {
    rt_task_wait_period( NULL );
    getADC( (uint8_t) 0, &y[k-1] );
    // %control algorithm
    e[k-1] = r[k-1] - y[k-1];
    u[k-1] = e[k-1] * Kp;
    // %saturation
    if( u[k-1] > 4.5 ) {
      u[k-1] = (float) 4.5;
    }
    if( u[k-1] < - 4.5 ) {
      u[k-1] = (float) - 4.5;
    }
    setDAC( (uint8_t) 0 , u[k-1] );
  }
  setDAC( (uint8_t) 0 , 0 );
  return;
}
```

5.5 SUMMARY

In the beginning of this thesis it was intended to have the algorithms running on MATLAB and on Raspberry Pi. When running in the last, it was aimed to achieve a better performance, and by performance it is meant a faster execution with smaller standard deviation and maximum time. From the comparison of the Tables on Section 5.2, it is concluded that the objective was reached, with improvements by a factor of 14 to 150 for period time, and in ADC and DAC operation. The Raspberry Pi was up to two times faster than MATLAB when performing calculations.

When using MATLAB, the control algorithm should only be initiated after three periods, to avoid the initial delays that are presented on Figure 5.9 and 5.15. If this is done, it can be achieved satisfactory results as seen on Tables 5.3 and 5.5, as in most cases the standard deviation was not bigger than 1.5ms and the maximum value smaller than 22ms. This values are expected for a best effort OS.

Regarding the individual tests, the communication protocol working in a non RTOS presents good results with a low standard deviation, although with the maximum time being 10 times bigger than the mean value. During the execution of the test, there were not captured many errors, a total of 44 errors

for 2000000 executions, what gives a 0.0022% error rate in the communication protocol. Although, they still happen, and when they did, the user was warned with a message during the execution of the code. To note, there is a slight difference from the mean values of Table 5.1 and from the RST test on Table 5.3 and 5.5. This difference is due to two factors: the test realized in the Section 5.1.1 were done in a free running and in the control section (5.2), at a fix period accomplished by doing busy-waiting delay. The second reason is that this tests were performed without the hardware connected, as it was only intended to test the communication protocol, but as can be seen in the Subsection 5.1.2, the hardware does not increase the time significantly.

From the tests done in this chapter with the controllers and protocol communication, it can be concluded that it will not be possible to do data acquisition or to implement the controllers on MATLAB working at a frequency of 1000Hz. However this is not a problem as the primarily intended was to have the DAC and ADC capable of working at a frequency of 1000Hz on Raspberry Pi, which is proved on Table 5.2.

About the hardware implemented, the IC MAX11300 responsible for the digital and analog interface, in the test realized on the analog output, presents an almost constant voltage offset of 2LSBs that could be reduce to almost zero. This offset was not corrected on software as it would require a better characterization of all channels for a different set of temperatures.

About the analog input, it has a non linear difference at -5V to the next measured value; the range in use should be reduced to -4.5V to 4.5V.

For the PWM interface, it was implemented the necessary functions to interact with MATLAB, but the same were not created to execute on Raspberry Pi, so it were not performed any tests to the functionalities of this interface.

Finally, the code translator worked as it was supposed to work, whenever the code structure was followed. When errors of translation happen during the initial builds, the line was well identified and it was easy to fix. The errors generated were caused by the use of functions unknown to the translator or missing declaration of arrays or variables. Thought the software does not support a big range of function, in total only 25, these were enough to perform the translations in this chapter.

CHAPTER 6

Conclusions and Future work

6.1 conclusion

For the development of this thesis it was necessary to acquire multiple knowledge in several fields. First it was done a research to discover if there was some solution that could fulfill entirely our requirements. When it was concluded that it was not available, it was done a different study about the several embedded system that could perform most of the intended task. Next it was needed to do a study about the different communication protocols, and which were more advantageous for our system. As the developed platform had time constrains to be met, there was the need to select an appropriate real-time operative system, that would bring an adequate performance to the system and also available to Raspberry Pi.

The goal of this master degree thesis was to develop an aggregation of analog and digital I/O to interact with a physical system. This platform was called RasP:IO. With the hardware implemented, it was created a communication protocol that would allow the exchange of information between MATLAB and Raspberry Pi, and the strategies applied to maximize the success of the sent messages, within a certain time. It was created a toolchain that would automatize all the process of code translation from the MATLAB to C, and the controlling of its execution.

In the end, there were performed numerous tests to determine the performance of the developed work. The test on the communication protocol revealed that the error rate was less than 0.01%, better than the initial intended for non RTOS. Afterwards, were performed multiple tests to the RasP:IO interface, and was determined a good time performance with constant times of execution, and a sporadic maximum time of two times the mean value. The analog performance of the interface was not perfect but when considering the range that should be of use for control applications, the accuracy of the interface was enough. With the tests realized with the controllers, MATLAB reveals to have a satisfactory performance when took in consideration precaution stated in the previous chapter. When the controllers' algorithm were translated to C programming language to execute on Raspberry Pi, the quality of control was improved, with the jitter of sampling and actuation time of a few microseconds.

6.2 FUTURE WORK

For future work it would be interesting to improve the general performance and add additional features, such as:

- Create a command to the communication protocol that allows it to make an analog read and write as one operation.
- Create a real-time device drivers to SPI and I2C to improve the maximum execution time on Raspberry Pi.
- Develop the HAL for the IC responsible by the PWM feature.
- Do an extensive temporal analysis of the behavior of Raspberry Pi in normal conditions and under load.

APPENDIX A

Rasp:IO schematic

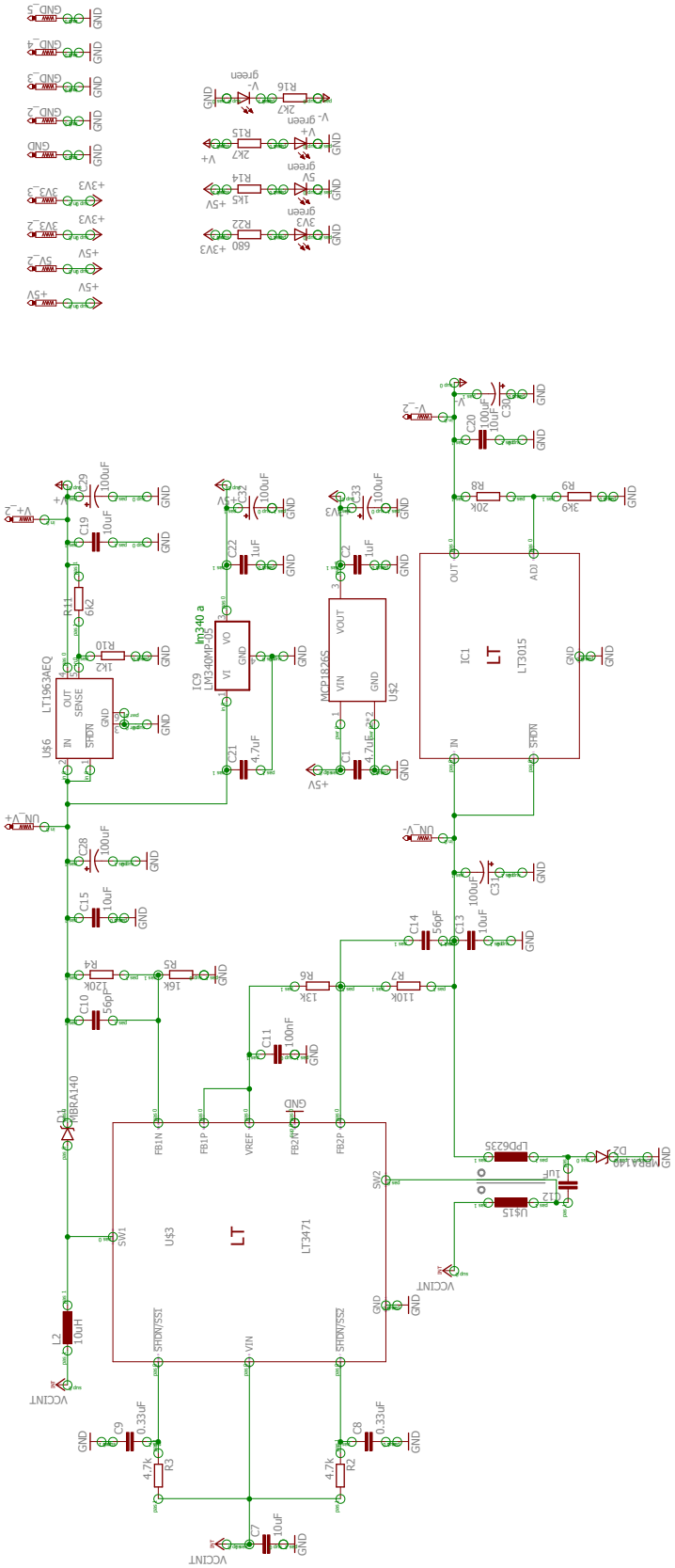


Figure A.1: Power converter

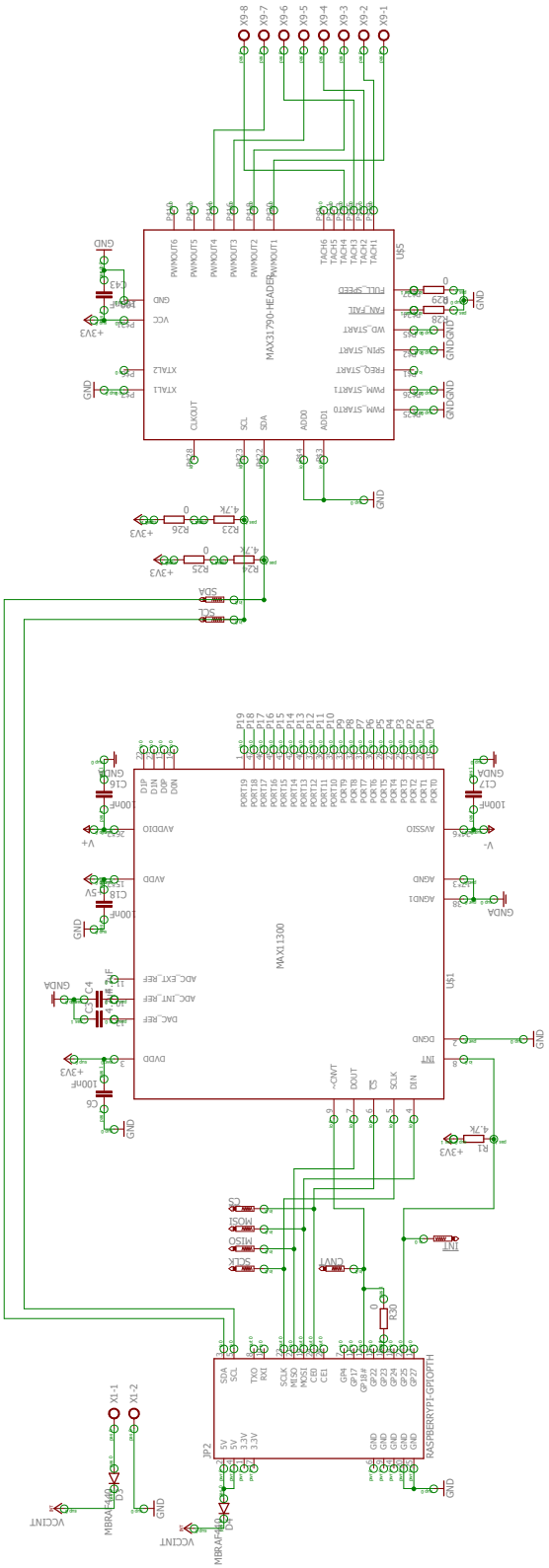


Figure A.2: Main ICs

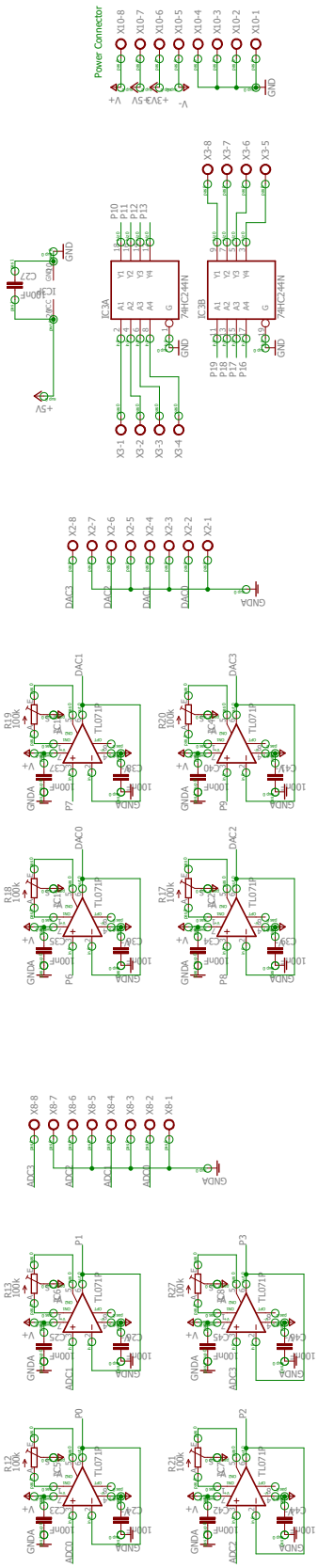


Figure A.3: Protection circuit

APPENDIX B

Installing Xenomai on Raspberry Pi

This guide explains how to install Xenomai in Raspberry Pi 1.

B.1 PREREQUISITES

It is required a computer with Linux installed, preferable with some Ubuntu variant, with Internet connection, Raspberry Pi version 1 and a SD card.

B.2 INSTALL RASPBIAN

1. Download **RASPBIAN WHEEZY** from: `https://www.raspberrypi.org/downloads/raspbian/`
2. Format the SD card, using Gparted and delete all partitions of the SD card and create one in FAT32.
3. Copy the downloaded image to the SD card. Use /dev/sdd not /dev/sdd1. Assuming /dev/sdd is the name of the card.

```
$ dd bs=4M if=2015-05-05-raspbian-wheezy.zip of=/dev/sdd
```

4. Plug the SD card on the Raspberry Pi, and let it do the first boot. To connect to the Raspberry Pi without a screen and keyboard, it is need the IP address, that can be find with:

```
$ arp -a
```

Or connect directly to a router.
The wanted IP, has the physical address like B8:27:EB:**:**:**

5. To access to the Raspberry Pi.

```
$ ssh pi@IP_ADDRESS -X
```

6. Shut down the Raspberry Pi and put the SD card back on the PC. To shut down, type on terminal:

```
$ sudo shutdown -h now
```

B.3 DONWLOAD AND PREPARE XENOMAI

On the PC execute the following steps:

1. Go to the website **https://github.com/awesomebytes/xenorasp** and download:

```
./1_download_xenorasp_stuff.sh
./2_apply_xenomai_ipipe_patches.sh
rpi_xenomai_2.6.3_linux_3.10_config
```

2. Execute both scripts on the terminal.
3. Type the following command:

```
$ cp ../rpi_xenomai_2.6.3_linux_3.10_config
  linux-rpi-3.10.y/build/.config
$ cd linux-rpi-3.10.y
$ mkdir build
$ make mrproper
$ make ARCH=arm O=build menuconfig
```

4. Inside Kernel Configuration load the "rpi_xenomai_2.6.3_linux_3.10_config" and do the following changes:

```
System Type -> Broadcom BCM2708 Development Platform ->
  ->Bind spidev to SPI0 master *
CPU Power Management -> CPU idle PM support (NO)
CPU Power Management -> CPU Frequency scaling (NO)
Device Drivers -> SPI support -> BCM2708 SPI controller driver (SPI0) *
```

```
Device Drivers -> SPI support -> User mode SPI device driver support *
Real-time sub-system -> Drivers -> Testing drivers ->
  ->Timer benchmark driver *
Real-time sub-system -> Drivers -> Testing drivers ->
  ->Kernel-only latency measurement module *
```

5. Compile Xenomai.

```
$ make ARCH=arm O=build CROSS_COMPILE=../../tools-master/arm-bcm2708/
arm-bcm2708hardfp-linux-gnueabi/bin/arm-bcm2708hardfp-linux-gnueabi- -j5
$ make ARCH=arm O=build INSTALL_MOD_PATH=dist modules_install
$ make ARCH=arm O=build INSTALL_HDR_PATH=dist headers_install
```

6. Convert absolute symlinks of the source directory to relative.

```
$ sudo apt-get install symlinks
$ symlinks -cr ./
```

B.4 INSTALL XENOMAI ON RASPBERRY PI

In this section it will be explained how to copy the files generated by the last step and some required source files to Raspberry Pi.

1. Copy the zImage to boot

```
$ cp build/arch/arm/boot/zImage /media/RASP_BOOT_PARTITION
```

2. Copy source. If it is not intended to compile modules for kernel, this step can be skipped, and some hundred megabytes will be saved.

```
$ cp -r ../linux-rpi-3.10.y/. /media/RASP_FILESYSTEM_ROOT/home/pi/linux/
```

3. Copy the dist to / of Linux filesystem (include and lib is there).

```
$ cd build/dist
$ sudo cp -r * /media/RASP_FILESYSTEM_ROOT
```

4. Add to config.txt of your BOOT partition the line:

```
kernel=zImage
```

5. Now plug the SD card on your Raspberry Pi to download some examples, code, test code and documentation.

```
$ cd /home
$ wget http://download.gna.org/xenomai/stable/xenomai-2.6.2.1.tar.bz2
$ tar -jxvf xenomai-2.6.2.1.tar.bz2
$ mv -r xenomai-2.6.2.1 xenomai-head
$ cd xenomai-head
$ make
$ sudo make install
```

B.5 TEST INSTALLATION AND PERFORMANCE EVALUATION

1. Test the performance in user space:

```
$ sudo su
$ echo "0" >>/proc/xenomai/latency
$ exit
$ sudo /usr/xenomai/bin/latency
```

The expected results are shown in Figure B.1.

```
pi@raspberrypi ~ $ sudo /usr/xenomai/bin/latency
== Sampling period: 1000 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 1000 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      6.836|     10.608|     23.540|       0|     0|      6.836|     23.540
RTD|      5.316|     10.268|     44.080|       0|     0|      5.316|     44.080
RTD|      6.616|     10.308|     40.300|       0|     0|      5.316|     44.080
RTD|      6.652|     10.364|     35.396|       0|     0|      5.316|     44.080
RTD|      6.448|     10.212|     29.756|       0|     0|      5.316|     44.080
RTD|      6.748|     10.320|     40.048|       0|     0|      5.316|     44.080
^C---|-----------|-----------|-----------|--------|------|-----------------------
RTS|      5.316|     10.344|     44.080|       0|     0|    00:00:06/00:00:06
```

Figure B.1: User space latency results

2. To test the latency of Xenomai in kernel space:

```
$ cd /lib/modules/`uname -r`/kernel/drivers/xenomai/testing
$ sudo modprobe ./xeno_klat.ko
```

```
$ /usr/xenomai/bin/klatency
```

NOTE: If the last method gives error, try the following command:

```
$ sudo modprobe xeno_timerbench
$ sudo /usr/xenomai/bin/latency -t1
```

The expected results are shown in Figure B.2.

```
pi@raspberrypi ~ $ sudo /usr/xenomai/bin/latency -t1
== Sampling period: 1000 us
== Test mode: in-kernel periodic task
== All results in microseconds
warming up...
RTT|  00:00:01  (in-kernel periodic task, 1000 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      2.596|      5.203|     15.624|       0|     0|      2.596|     15.624
RTD|      2.612|      5.152|     25.084|       0|     0|      2.596|     25.084
RTD|      2.736|      5.112|     23.716|       0|     0|      2.596|     25.084
RTD|      2.640|      5.167|     28.316|       0|     0|      2.596|     28.316
RTD|      2.644|      5.116|     25.624|       0|     0|      2.596|     28.316
RTD|      2.724|      5.120|     22.456|       0|     0|      2.596|     28.316
^C---|-----------|-----------|-----------|--------|------|--------------------------
RTS|      2.596|      5.145|     28.316|       0|     0|    00:00:06/00:00:06
```

Figure B.2: Kernel space latency results

B.6 FINAL ADJUSTMENTS

1. Join build and source directory in one.

```
$ cd ~/linux
$ rm ./build/Makefile
$ rm ./build/source
$ cp -R -v build/. .
```

2. In order to compile some modules in kernel space it is necessary to do the following fix:

```
$ sudo cp -R /home/pi/linux/lib/modules/'uname -r'/ /lib/modules/
$ sudo cd /lib/modules/'uname -r'/
$ sudo rm build source
$ ln -s /home/pi/linux build
$ ln -s /home/pi/linux source
```

3. To see if it can compile modules, you can run the next comands:

```
$ cd ~/Documents
```

```
$ wget:
    https://dl.dropboxusercontent.com/u/31280811/Xenomai/Hello_module.tar
$ tar -xvf Hello_module.tar
$ make
$ sudo insmod hello.ko
$ sudo rmmod hello
$ dmesg | tail
```

Expected:

```
Hello world 1.
Goodbye world 1.
```

APPENDIX C

UDP based Communication Protocol

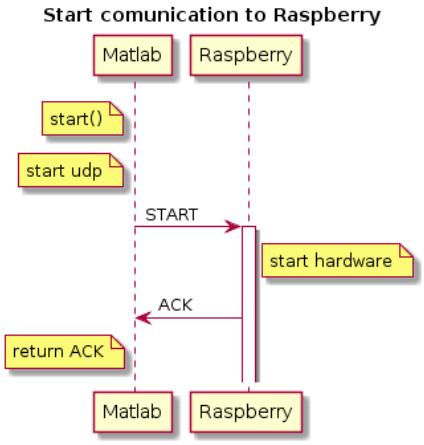
C.1 Interface specifications

This appendix provides a detailed description of the protocol that was implemented to allow the communication between the PC and the Rasp Pi.

START CONNECTION

The client has to initiate the communication with the server, done with the start command. This will initiate the UDP protocol, send a START message and wait until the ACK reception.

Syntax: start( )

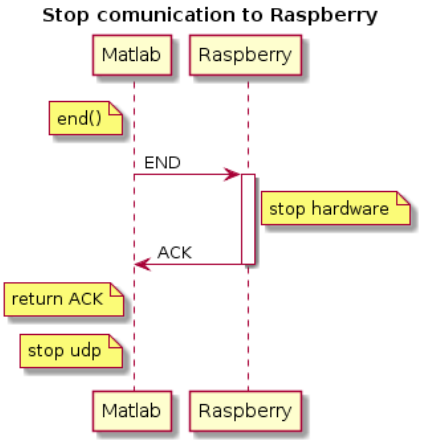


Arguments: None
Return: 0 or NaN (Not a Number) in case of error

END CONNECTION

When client does not need to do any more operations with the server, it must send the end command, that will transmit the STOP message and will wait until the ACK, after it will to close the UDP socket.

Syntax: end( )

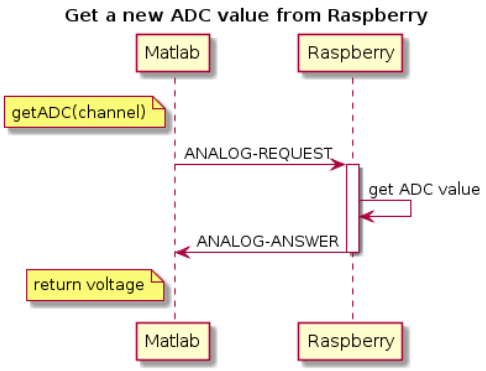


Arguments: None
Return: 0 or NaN in case of error

GET A NEW ADC VALUE

The command getADC is used to get a new read from an ADC channel. It will send a request to the server with ANALOG-REQUEST message, and will receive the new value on ANALOG-ANSWER message.

Syntax: getADC(channel)

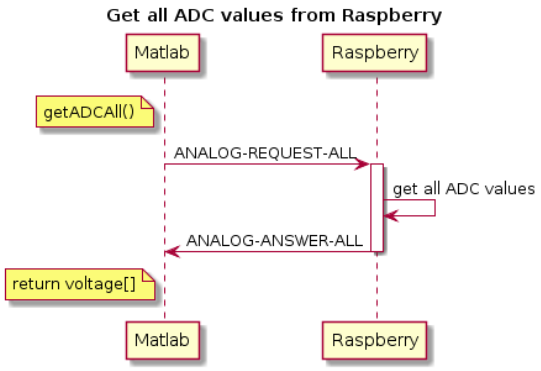


Arguments: Channel: int value from 0 to 3
Return: Voltage, float beteween -5 or 5, or NaN in case of error

GET ALL ADC VALUES

The command getADCAll is used to get all the analog values at the same time from the ADC of each analog channel. It will send a request to the server with ANALOG-REQUEST-ALL message, and will receive the four values on ANALOG-ANSWER-ALL message.

Syntax: getADCAll( )

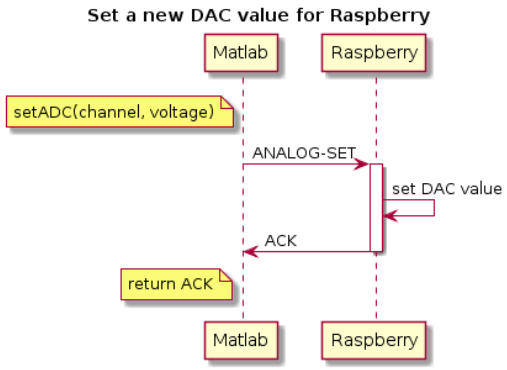


Arguments: None
Return: Voltage, float' array with 4 elements between -5 and 5 volt, or NaN in the position(s) with error(s)

SET A NEW DAC VALUE

The command setDAC is used to set the output voltage of one of the DAC channels. It will send an ANALOG-SET message and will wait for the ACK message.

Syntax: setDAC(channel, voltage)

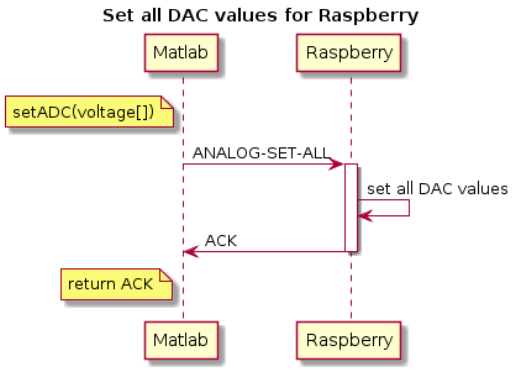


Arguments: Channel: int value from 0 to 3
Voltage: float between -5 and 5 volts
Return: 0 or NaN in case of error

SET ALL DAC VALUES

The command setDACAll is used to set the output voltage of all the DAC channels. It will send a ANALOG-SET-ALL message, and will wait for the ACK message.

Syntax: setDACAll(voltage[])

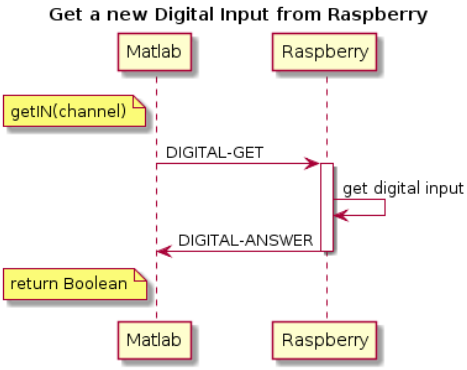


Arguments: Voltage: float' array with 4 elements between -5 and 5 volts, or NaN on the position(s) to not set any
Return: 0 or NaN in case of error

GET A NEW DIGITAL INPUT

The command getIN will get the digital value from one of the digital inputs. The command will send a DIGITAL-GET message and will wait for the DIGITAL-ANSWER.

Syntax: getIN(channel)

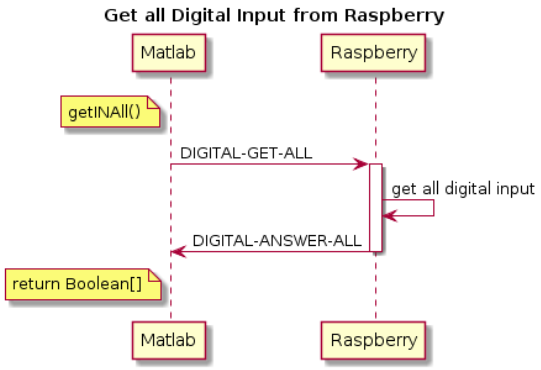


Arguments: Channel: int value from 0 to 3
Return: Boolean 0, 1 or NaN in case of error

GET ALL DIGITAL INPUTS

The command getINAll will get the digital value from all channels. The command will send a DIGITAL-GET-ALL message and will wait for the DIGITAL-ANSWER-ALL.

Syntax: getINAll( )

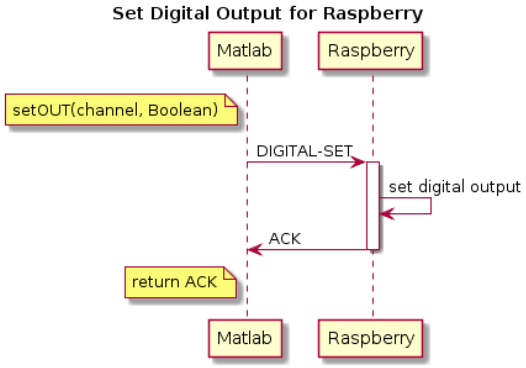


Arguments: None
Return: Boolean' array with 4 elements of 0, 1 or NaN in the position(s) with error(s)

SET A NEW DIGITAL OUTPUT

The command setOUT sets the digital value of one of the digital outputs. When executing the command, is sent a DIGITAL-SET message to the server and it will answer back with a ACK massage.

Syntax: setOUT(channel, boolean)

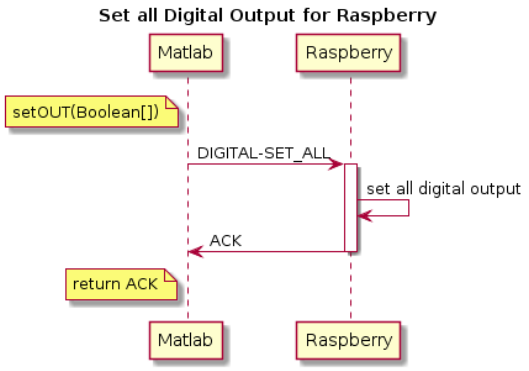


Arguments: Channel: int value from 0 to 3
Boolean: 0 or 1
Return: 0 or NaN in case of error

SET ALL DIGITAL OUTPUTS

The command setOUTAll sets the digital value of all digital outputs. When executing the command, is send a DIGITAL-SET-All message to the server and it will answer back with a ACK massage.

Syntax: setOUTAll(boolean[])

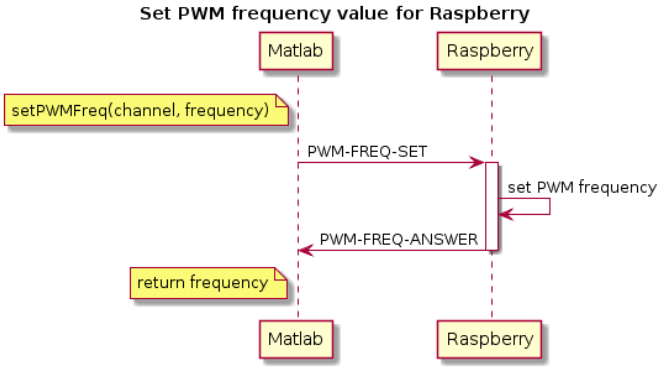


Arguments: Boolean' array with 4 elements of 0, 1 or NaN in the position(s) to not set any value(s)
Return: 0 or NaN in case of error

SET PWM FREQUENCY VALUE

The command setPWMFreq, sets the frequency of the PWM channel. It is sent a PWM-FREQ-SET by the client and it will wait for a PWM-FREQ-ANSWER. In case the requested frequency is not supported by the hardware, it is applied the next lower value, and returned in the answer message. It is important to notice in the Rasp:IO, the first three channels share the same clock, so any change in frequency in one channel will be applied to the other two. For more information about this limitation, consult Section 3.3.1 on Chapter 3.

Syntax: setPWMFreq(channel, frequency)

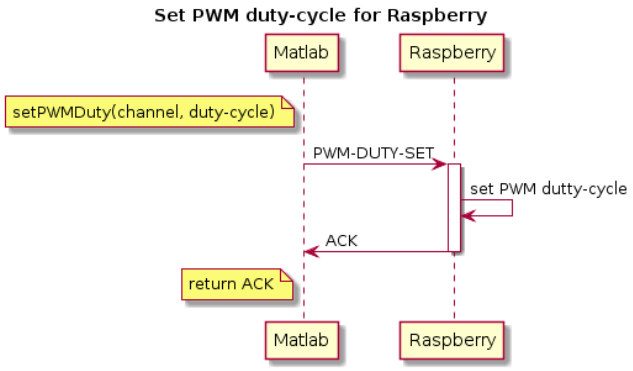


Arguments: Channel: int value from 0 to 3
Frequency: 25, 30, 35, 100, 125, 150, 1250, 1470, 3570, 5000, 12500 or 25000 in Hz
Return: Frequency or NaN in case of error

SET PWM DUTY-CYCLE VALUE

The command setPWMDuty sets the duty-cycle value of one of the the PWM channels. It is sent a PWM-DUTY-SET message and the server answers with an ACK message.

Syntax: setPWMDuty(channel, duty-cycle)



Arguments:	Channel:	int value from 0 to 3
	Duty-cycle:	float between 0 and 100
Return:	0 or NaN in case of error	

C.2 MESSAGE FORMAT

In this section will be shown the registers that compose each message, the order that take as well the number of bytes.

- Start:

Byte	0	1	2
	Seq Num	ID	CheckSum

- End:

Byte	0	1	2
	Seq Num	ID	CheckSum

- Analog-Request:

Byte	0	1	2	3
	Seq Num	ID	Channel	CheckSum

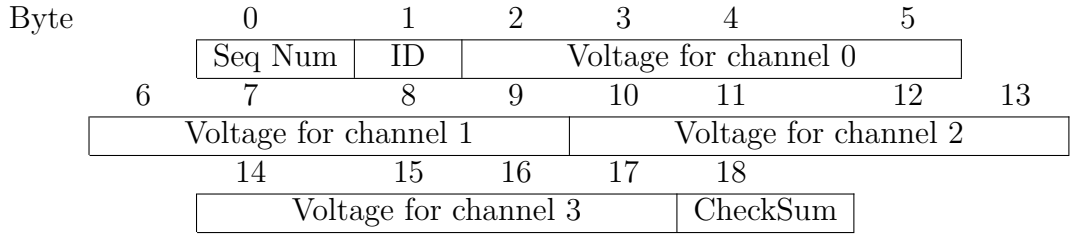
- Analog-Request-All:

Byte	0	1	2
	Seq Num	ID	CheckSum

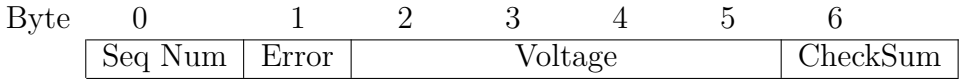
- Analog-Set:

Byte	0	1	2	3	4	5	6	7
	Seq Num	ID	Channel	Voltage				CheckSum

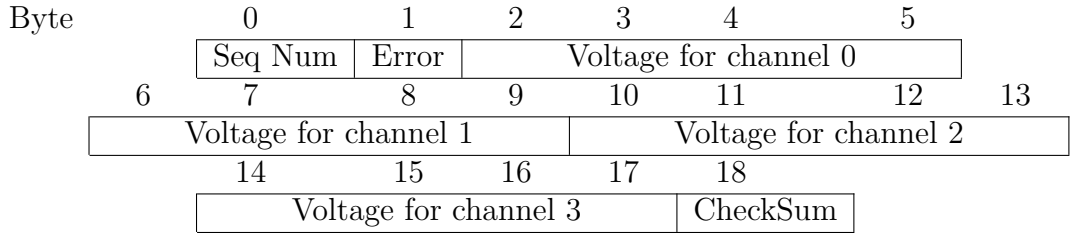
- Analog-Set-All:



- Analog-Answer:

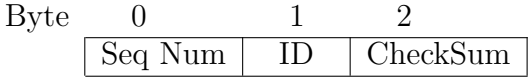


- Analog-Answer-All:

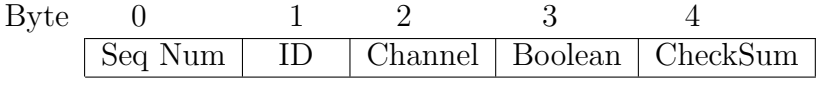


- Digital-Get:

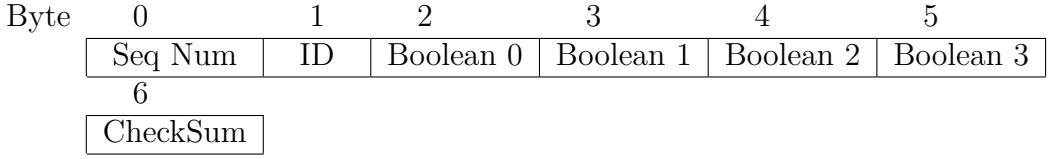
- Digital-Get-All:



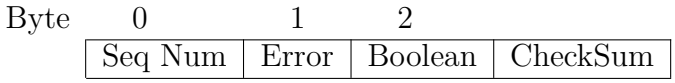
- Digital-Set:



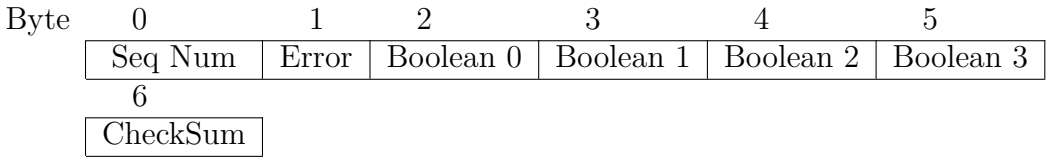
- Digital-Set-All:



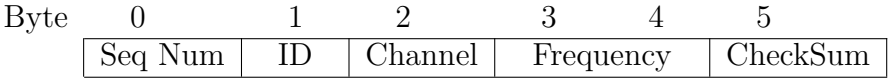
- Digital-Answer:



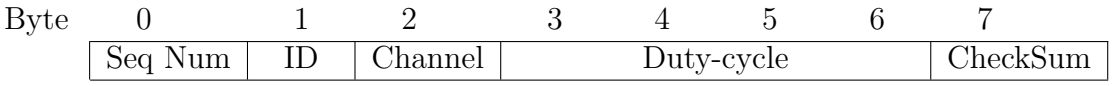
- Digital-Answer-All:



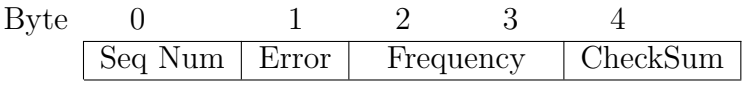
- PWM-Freq-Set:



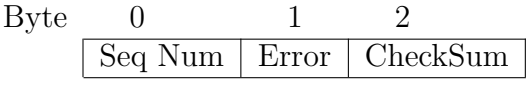
- PWM-Duty-Set:



- PWM-Freq-Answer:



- ACK:



C.3 TABLE ID

This section specifies the ID (identification number) for all the commands.

ID number	Related command
100	Start
101	End
110	Analog-Request
112	Analog-Request-All
115	Analog-Set
116	Analog-Set-All
120	Digital-Get
121	Digital-Get-All
125	Digital-Set
126	Digital-Set-All
130	PWM-Freq-Set
131	PWM-Duty-Set

Table C.1: ID number of each command

C.4 TABLE ERROR

This section specifies the error numbers.

Error number	Related name
0	No-Error
100	Already-Open
101	Not-Open
110	Channel
115	Truncate-Value
116	Invalid-Value

Table C.2: Error number and it name

APPENDIX D

Communication Protocol test

D.1 script to test getADC() command

```
% Add dependencies to path
addpath('../mcu_comm');
path = ['getADC_' datestr(now,'dd-mm-yy_HH-MM')];
mkdir('.\test_protocolo',path);
addpath(['test_protocolo\' path]);
cd (['test_protocolo\' path]);
diary cmd_line.txt
cd ..\..\;

%% Initialize the UDP connection
LOCAL_PORT  = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP   = 'raspberrypi.mshome.net';
REMOTE_PORT = 10000;
TIMEOUT     = 8;                % Depending on the transmit period of the
    remote device (ms)

% Initialize the communication object
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT);

%% Setup the loop
sim_k = 200000;
M = 5;

for m=1:M
    telapsed = zeros(1,sim_k);
    % Send data to MCU
    raspio.start();
    adc = [-500 -500 -500 -500];
    warn(m) = 0;
```

```
29      error_n(m) = 0;
30      for k=1:sim_k
31          channel = randi([0 3],1,1);
32
33          tstart = tic;
34          aux = raspio.getADC(channel);
35          telapsed(k) = toc(tstart);
36
37          if( isnan(aux) )
38              fprintf('Error: %d\n',k);
39              error_n(m) = error_n(m)+1;
40              telapsed(k) = nan;
41          end
42
43          if( abs(adc(channel +1)*0.01 - aux) > 0.0001 )
44              warning('Ch %d\tk %d\tExpeted %d and received
                    %d\n',channel,k,adc(channel +1)*0.01, aux)
45              warn(m) = warn(m) +1;
46              adc(channel +1) = int16(aux * 100);
47          end
48          adc(channel +1) = adc(channel +1) + 1;
49          if( adc(channel +1) > 500 )
50              adc(channel +1) = -500;
51          end
52
53  %           if( mod(k,10000) == 0)
54  %               fprintf('k = %d\n',k);
55  %           end
56      end
57      raspio.end();
58      time(m,:)= telapsed *1000;
59      figure;
60      h(m) = histogram(time(m,:));
61      med(m) = nanmean(time(m,:));
62      desvio(m) = nanstd(time(m,:));
63      maximo(m) = nanmax(time(m,:));
64      str = sprintf('Media: %fms \tDesvio: %fms\tMaximo: %fms\tWarning:
            %d\tError:
            %d\n----------------------------\n\n',med(m),desvio(m),maximo(m),warn(m),error_n(m));
65      disp(str);
66      pause(2);
67  end
68
69  cd (['test_protocolo\' path]);
70  for k=1:M
71    saveas(h(k),sprintf('figure_%d.fig',k))
72    saveas(h(k),sprintf('figure_%d.png',k))
73  end
74  clearvars -except time med desvio maximo warn error_n
75  save('workspace.mat');
76  datestr(now,'dd-mm-yy_HH-MM');
77  diary off
78  cd ..\..\;
```

D.2 SCRIPT TO TEST GETADCALL() COMMAND

```
% Add dependencies to path
% addpath('../mcu_comm');
path = ['getADCAll_' datestr(now,'dd-mm-yy_HH-MM')];
mkdir('.\test_protocolo',path);
addpath(['test_protocolo\' path]);
cd (['test_protocolo\' path]);
diary cmd_line.txt
cd ..\..\;

%% Initialize the UDP connection
LOCAL_PORT  = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP   = 'raspberrypi.mshome.net';
REMOTE_PORT = 10000;
TIMEOUT     = 8;                % Depending on the transmit period of the
    remote device (ms)

% Initialize the communication object
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT);

%% Setup the loop
sim_k = 200000;
M = 5;

for m=1:M
    telapsed = zeros(1,sim_k);
    % Send data to MCU
    raspio.start();
    adc = [-500 -400 -300 -200];
    warn(m) = 0;
    error_n(m) = 0;
    for k=1:sim_k

        tstart = tic;
        aux = raspio.getADCAll();
        telapsed(k) = toc(tstart);

        if( isnan(aux) )
            fprintf('Error: %d\n',k);
            error_n(m) = error_n(m)+1;
            telapsed(k) = nan;
            continue;
        end

        for i=1:4
            if( abs(adc(i)*0.01 - aux(i)) > 0.0001 )
                warning('Ch %d\tk %d\tExpeted %d and received
                    %d\n',i,k,adc(i)*0.01, aux(i))
                warn(m) = warn(m) +1;
                adc(i) = int16(aux(i) * 100);
            end
        end
```

```
50
51          for i=1:4
52              adc(i) = adc(i) + 1;
53              if( adc(i) > 500 )
54                  adc(i) = -500;
55              end
56          end
57
58      end
59      raspio.end();
60      time(m,:)= telapsed *1000;
61      figure;
62      h(m) = histogram(time(m,:));
63      med(m) = nanmean(time(m,:));
64      desvio(m) = nanstd(time(m,:));
65      maximo(m) = nanmax(time(m,:));
66      str = sprintf('Media: %fms \tDesvio: %fms\tMaximo: %fms\tWarning:
            %d\tError:
            %d\n----------------------------\n\n',med(m),desvio(m),maximo(m),warn(m),error_n(m));
67      disp(str);
68      pause(2);
69  end
70
71  cd (['test_protocolo\' path]);
72  for k=1:M
73    saveas(h(k),sprintf('figure_%d.fig',k))
74    saveas(h(k),sprintf('figure_%d.png',k))
75  end
76  clearvars -except time med desvio maximo warn error_n
77  save('workspace.mat');
78  datestr(now,'dd-mm-yy_HH-MM');
79  diary off
80  cd ..\..\;
```

D.3 SCRIPT TO TEST SETDAC() COMMAND

```
% Add dependencies to path
addpath('../mcu_comm');
path = ['setDAC_' datestr(now,'dd-mm-yy_HH-MM')];
mkdir('.\test_protocolo',path);
addpath(['test_protocolo\' path]);
cd (['test_protocolo\' path]);
diary cmd_line.txt
cd ..\..\;

%% Initialize the UDP connection
LOCAL_PORT  = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP   = 'raspberrypi.mshome.net';
REMOTE_PORT = 10000;
TIMEOUT     = 8;                % Depending on the transmit period of the
    remote device (ms)

% Initialize the communication object
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT);

%% Setup the loop
sim_k = 200000;
M = 5;

for m=1:M
    telapsed = zeros(1,sim_k);
    % Send data to MCU
    raspio.start();
    dac = [-500 -500 -500 -500];
    warn(m) = 0;
    error_n(m) = 0;
    for k=1:sim_k
        channel = randi([0 3],1,1);

        tstart = tic;
        aux = raspio.setDAC(channel, dac(channel+1)*0.01);
        telapsed(k) = toc(tstart);

        if( isnan(aux) )
            fprintf('Error: %d\n',k);
            error_n(m) = error_n(m)+1;
            telapsed(k) = nan;
        end

        dac(channel+1) = dac(channel+1) + 1;
        if( dac(channel+1) > 500 )
            dac(channel+1) = -500;
        end

%         if( mod(k,10000) == 0)
%             fprintf('k = %d\n',k);
%         end
```

```
51      end
52      raspio.end();
53      time(m,:)= telapsed *1000;
54      figure;
55      h(m) = histogram(time(m,:));
56      med(m) = nanmean(time(m,:));
57      desvio(m) = nanstd(time(m,:));
58      maximo(m) = nanmax(time(m,:));
59      str = sprintf('Media: %fms \tDesvio: %fms\tMaximo: %fms\tWarning:
            %d\tError:
            %d\n----------\n\n',med(m),desvio(m),maximo(m),warn(m),error_n(m));
60      disp(str);
61      pause(2);
62  end
63
64  cd (['test_protocolo\' path]);
65  for k=1:M
66    saveas(h(k),sprintf('figure_%d.fig',k))
67    saveas(h(k),sprintf('figure_%d.png',k))
68  end
69  clearvars -except time med desvio maximo warn error_n
70  save('workspace.mat');
71  datestr(now,'dd-mm-yy_HH-MM');
72  diary off
73  cd ..\..\;
```

D.4 SCRIPT TO TEST SETDACALL() COMMAND

```
% Add dependencies to path
% addpath('../mcu_comm');
path = ['setDACAll_' datestr(now,'dd-mm-yy_HH-MM')];
mkdir('.\test_protocolo',path);
addpath(['test_protocolo\' path]);
cd (['test_protocolo\' path]);
diary cmd_line.txt
cd ..\..\;

%% Initialize the UDP connection
LOCAL_PORT  = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP   = 'raspberrypi.mshome.net';
REMOTE_PORT = 10000;
TIMEOUT     = 8;                % Depending on the transmit period of the
    remote device (ms)

% Initialize the communication object
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT);

%% Setup the loop
sim_k = 200000;
M = 5;

for m=1:M
    telapsed = zeros(1,sim_k);
    % Send data to MCU
    raspio.start();
    dac = [-500 -500 -500 -500];
    warn(m) = 0;
    error_n(m) = 0;
    for k=1:sim_k
        tstart = tic;
        aux = raspio.setDACAll(dac*0.01);
        telapsed(k) = toc(tstart);

        if( isnan(aux) )
            fprintf('Error: %d\n',k);
            error_n(m) = error_n(m)+1;
            telapsed(k) = nan;
        end

        for i=1:4
            dac(i) = dac(i) + 1;
            if( dac(i) > 500 )
                dac(i) = -500;
            end
        end

%         if( mod(k,10000) == 0)
%             fprintf('k = %d\n',k);
%         end
```

```
51      end
52      raspio.end();
53      time(m,:)= telapsed *1000;
54      figure;
55      h(m) = histogram(time(m,:));
56      med(m) = nanmean(time(m,:));
57      desvio(m) = nanstd(time(m,:));
58      maximo(m) = nanmax(time(m,:));
59      str = sprintf('Media: %fms \tDesvio: %fms\tMaximo: %fms\tWarning:
            %d\tError:
            %d\n----------\n\n',med(m),desvio(m),maximo(m),warn(m),error_n(m));
60      disp(str);
61      pause(2);
62  end
63
64  cd (['test_protocolo\' path]);
65  for k=1:M
66    saveas(h(k),sprintf('figure_%d.fig',k))
67    saveas(h(k),sprintf('figure_%d.png',k))
68  end
69  clearvars -except time med desvio maximo warn error_n
70  save('workspace.mat');
71  datestr(now,'dd-mm-yy_HH-MM');
72  diary off
73  cd ..\..\;
```

D.5 SCRIPT TO TEST GETIN() COMMAND

```
% Add dependencies to path
addpath('../mcu_comm');
path = ['getIN_' datestr(now,'dd-mm-yy_HH-MM')];
mkdir('.\test_protocolo',path);
addpath(['test_protocolo\' path]);
cd (['test_protocolo\' path]);
diary cmd_line.txt
cd ..\..\;

%% Initialize the UDP connection
LOCAL_PORT  = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP   = 'raspberrypi.mshome.net';
REMOTE_PORT = 10000;
TIMEOUT     = 8;                % Depending on the transmit period of the
    remote device (ms)

% Initialize the communication object
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT);

%% Setup the loop
sim_k = 200000;
M = 5;

for m=1:M
    telapsed = zeros(1,sim_k);
    % Send data to MCU
    raspio.start();
    dac = [-500 -500 -500 -500];
    warn(m) = 0;
    error_n(m) = 0;
    for k=1:sim_k
        channel = randi([0 3],1,1);

        tstart = tic;
        aux = raspio.getIN();
        telapsed(k) = toc(tstart);

        if( isnan(aux) )
            fprintf('Error: %d\n',k);
            error_n(m) = error_n(m)+1;
            telapsed(k) = nan;
        end

        if( bitand(raspio.channel + raspio.seq_num,1) ~= aux )
            warning('Ch %d\tk %d\tExpeted %d and received
                %d\n',channel,k,bitand(raspio.channel + raspio.seq_num,1),
                aux);
            warn(m) = warn(m) +1;
        end

%         if( mod(k,10000) == 0)
```

```
49 %                fprintf('k = %d\n',k);
50 %            end
51       end
52       raspio.end();
53       time(m,:)= telapsed *1000;
54       figure;
55       h(m) = histogram(time(m,:));
56       med(m) = nanmean(time(m,:));
57       desvio(m) = nanstd(time(m,:));
58       maximo(m) = nanmax(time(m,:));
59       str = sprintf('Media: %fms \tDesvio: %fms\tMaximo: %fms\tWarning:
             %d\tError:
             %d\n----------------------------\n\n',med(m),desvio(m),maximo(m),warn(m),error_n(m));
60       disp(str);
61       pause(2);
62   end
63
64   cd (['test_protocolo\' path]);
65   for k=1:M
66     saveas(h(k),sprintf('figure_%d.fig',k))
67     saveas(h(k),sprintf('figure_%d.png',k))
68   end
69   clearvars -except time med desvio maximo warn error_n
70   save('workspace.mat');
71   datestr(now,'dd-mm-yy_HH-MM');
72   diary off
73   cd ..\..\;
```

D.6 SCRIPT TO TEST GETINALL() COMMAND

```
% Add dependencies to path
% addpath('../mcu_comm');
path = ['getINAll_' datestr(now,'dd-mm-yy_HH-MM')];
mkdir('.\test_protocolo',path);
addpath(['test_protocolo\' path]);
cd (['test_protocolo\' path]);
diary cmd_line.txt
cd ..\..\;

%% Initialize the UDP connection
LOCAL_PORT  = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP   = 'raspberrypi.mshome.net';
REMOTE_PORT = 10000;
TIMEOUT     = 8;                % Depending on the transmit period of the
    remote device (ms)

% Initialize the communication object
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT);

%% Setup the loop
sim_k = 200000;
M = 5;

for m=1:M
    telapsed = zeros(1,sim_k);
    % Send data to MCU
    raspio.start();
    warn(m) = 0;
    error_n(m) = 0;
    for k=1:sim_k

        tstart = tic;
        aux = raspio.getINAll();
        telapsed(k) = toc(tstart);

        if( isnan(aux) )
            fprintf('Error: %d\n',k);
            error_n(m) = error_n(m)+1;
            telapsed(k) = nan;
            continue;
        end

        for i=1:4
            if( length(aux) == 4 )
                if( bitand((i-1) + raspio.seq_num,1) ~= aux(i) )
                    warning('Ch %d\tk %d\tExpeted %d and received
                        %d\n',(i-1),k,bitand((i-1) + raspio.seq_num,1),
                        aux(i));
                    warn(m) = warn(m) +1;
                end
            else
```

```
49                  warning('%d\tonly received %d values\n',k,length(aux));
50                  warn(m) = warn(m) +1;
51                  break;
52              end
53          end
54
55      end
56      raspio.end();
57      time(m,:)= telapsed *1000;
58      figure;
59      h(m) = histogram(time(m,:));
60      med(m) = nanmean(time(m,:));
61      desvio(m) = nanstd(time(m,:));
62      maximo(m) = nanmax(time(m,:));
63      str = sprintf('Media: %fms \tDesvio: %fms\tMaximo: %fms\tWarning:
            %d\tError:
            %d\n----------------------------\n\n',med(m),desvio(m),maximo(m),warn(m),error_n(m));
64      disp(str);
65      pause(2);
66  end
67
68  cd (['test_protocolo\' path]);
69  for k=1:M
70    saveas(h(k),sprintf('figure_%d.fig',k))
71    saveas(h(k),sprintf('figure_%d.png',k))
72  end
73  clearvars -except time med desvio maximo warn error_n
74  save('workspace.mat');
75  datestr(now,'dd-mm-yy_HH-MM');
76  diary off
77  cd ..\..\;
```

D.7 SCRIPT TO TEST SETOUT() COMMAND

```
% Add dependencies to path
addpath('../mcu_comm');
path = ['setOUT_' datestr(now,'dd-mm-yy_HH-MM')];
mkdir('.\test_protocolo',path);
addpath(['test_protocolo\' path]);
cd (['test_protocolo\' path]);
diary cmd_line.txt
cd ..\..\;

%% Initialize the UDP connection
LOCAL_PORT  = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP   = 'raspberrypi.mshome.net';
REMOTE_PORT = 10000;
TIMEOUT     = 8;                % Depending on the transmit period of the
    remote device (ms)

% Initialize the communication object
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT);

%% Setup the loop
sim_k = 200000;
M = 5;

for m=1:M
    telapsed = zeros(1,sim_k);
    % Send data to MCU
    raspio.start();
    dac = [-500 -500 -500 -500];
    warn(m) = 0;
    error_n(m) = 0;
    for k=1:sim_k
        channel = randi([0 3],1,1);

        seq = bitand(raspio.seq_num +1,255);

        tstart = tic;
        aux = raspio.setOUT(channel,bitand(channel + seq,1));
        telapsed(k) = toc(tstart);

        if( isnan(aux) )
            fprintf('Error: %d\n',k);
            error_n(m) = error_n(m)+1;
            telapsed(k) = nan;
        end


%         if( mod(k,10000) == 0)
%             fprintf('k = %d\n',k);
%         end
    end
    raspio.end();
```

```
51      time(m,:)= telapsed *1000;
52      figure;
53      h(m) = histogram(time(m,:));
54      med(m) = nanmean(time(m,:));
55      desvio(m) = nanstd(time(m,:));
56      maximo(m) = nanmax(time(m,:));
57      str = sprintf('Media: %fms \tDesvio: %fms\tMaximo: %fms\tWarning:
            %d\tError:
            %d\n----------\n\n',med(m),desvio(m),maximo(m),warn(m),error_n(m));
58      disp(str);
59      pause(2);
60  end
61
62  cd (['test_protocolo\' path]);
63  for k=1:M
64    saveas(h(k),sprintf('figure_%d.fig',k))
65    saveas(h(k),sprintf('figure_%d.png',k))
66  end
67  clearvars -except time med desvio maximo warn error_n
68  save('workspace.mat');
69  datestr(now,'dd-mm-yy_HH-MM');
70  diary off
71  cd ..\..\;
```

D.8 SCRIPT TO TEST SETOUTALL() COMMAND

```
% Add dependencies to path
% addpath('../mcu_comm');
path = ['setOUTAll_' datestr(now,'dd-mm-yy_HH-MM')];
mkdir('.\test_protocolo',path);
addpath(['test_protocolo\' path]);
cd (['test_protocolo\' path]);
diary cmd_line.txt
cd ..\..\;

%% Initialize the UDP connection
LOCAL_PORT  = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP   = 'raspberrypi.mshome.net';
REMOTE_PORT = 10000;
TIMEOUT     = 8;                % Depending on the transmit period of the
    remote device (ms)

% Initialize the communication object
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT);

%% Setup the loop
sim_k = 200000;
M = 5;

for m=1:M
    telapsed = zeros(1,sim_k);
    % Send data to MCU
    raspio.start();
    warn(m) = 0;
    error_n(m) = 0;
    for k=1:sim_k
        seq = bitand(raspio.seq_num +1,255);
        out = [bitand(0 + seq,1), bitand(1 + seq,1), bitand(2 + seq,1),
            bitand(3 + seq,1)];
        tstart = tic;
        aux = raspio.setOUTAll(out);
        telapsed(k) = toc(tstart);

        if( isnan(aux) )
            fprintf('Error: %d\n',k);
            error_n(m) = error_n(m)+1;
            telapsed(k) = nan;
        end

    end
    raspio.end();
    time(m,:)= telapsed *1000;
    figure;
    h(m) = histogram(time(m,:));
    med(m) = nanmean(time(m,:));
    desvio(m) = nanstd(time(m,:));
    maximo(m) = nanmax(time(m,:));
```

```
50      str = sprintf('Media: %fms \tDesvio: %fms\tMaximo: %fms\tWarning:
            %d\tError:
            %d\n----------------------------\n\n',med(m),desvio(m),maximo(m),warn(m),error_n(m));
51      disp(str);
52      pause(2);
53  end
54
55  cd (['test_protocolo\' path]);
56  for k=1:M
57    saveas(h(k),sprintf('figure_%d.fig',k))
58    saveas(h(k),sprintf('figure_%d.png',k))
59  end
60  clearvars -except time med desvio maximo warn error_n
61  save('workspace.mat');
62  datestr(now,'dd-mm-yy_HH-MM');
63  diary off
64  cd ..\..\;
```

D.9 SCRIPT TO TEST SETPWMDUTY() COMMAND

```
% Add dependencies to path
addpath('../mcu_comm');
path = ['setPWMDuty_' datestr(now,'dd-mm-yy_HH-MM')];
mkdir('.\test_protocolo',path);
addpath(['test_protocolo\' path]);
cd (['test_protocolo\' path]);
diary cmd_line.txt
cd ..\..\;

%% Initialize the UDP connection
LOCAL_PORT  = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP   = '192.168.137.183';
REMOTE_PORT = 10000;
TIMEOUT     = 8;                % Depending on the transmit period of the
    remote device (ms)
MSG_LENGTH  = 10;               % Maximum datagram size (bytes)

% Initialize the communication object
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT,MSG_LENGTH);

%% Setup the loop
sim_k = 200000;
M = 5;

for m=1:M
    telapsed = zeros(1,sim_k);
    % Send data to MCU
    raspio.start();
    duty = [ 10 10 10 10];
    warn(m) = 0;
    error_n(m) = 0;
    for k=1:sim_k
        channel = randi([0 3],1,1);

        tstart = tic;
        aux = raspio.setDutyPWM(channel, duty(channel +1)*0.1);
        telapsed(k) = toc(tstart);

        if( isnan(aux) )
            fprintf('Error: %d\n',k);
            error_n(m) = error_n(m)+1;
            telapsed(k) = nan;
        end

        duty(channel +1) = duty(channel +1) + 1;
        if( duty(channel +1) > 1000 )
            duty(channel +1) = 0;
        end

%         if( mod(k,10000) == 0)
%             fprintf('k = %d\n',k);
```

```
51 %           end
52       end
53       raspio.end();
54       time(m,:)= telapsed *1000;
55       figure;
56       h(m) = histogram(time(m,:));
57       med(m) = nanmean(time(m,:));
58       desvio(m) = nanstd(time(m,:));
59       maximo(m) = nanmax(time(m,:));
60       str = sprintf('Media: %fms \tDesvio: %fms\tMaximo: %fms\tWarning:
             %d\tError:
             %d\n----------\n\n',med(m),desvio(m),maximo(m),warn(m),error_n(m));
61       disp(str);
62       pause(2);
63   end
64
65   cd (['test_protocolo\' path]);
66   for k=1:M
67     saveas(h(k),sprintf('figure_%d.fig',k))
68     saveas(h(k),sprintf('figure_%d.png',k))
69   end
70   clearvars -except time med desvio maximo warn error_n
71   save('workspace.mat');
72   datestr(now,'dd-mm-yy_HH-MM');
73   diary off
74   cd ..\..\;
```

D.10 SCRIPT TO TEST SETPWMFREQ() COMMAND

```
%Add dependencies to path
%addpath('../mcu_comm');
path = ['setPWMFreq_' datestr(now,'dd-mm-yy_HH-MM')];
mkdir('.\test_protocolo',path);
addpath(['test_protocolo\' path]);
cd (['test_protocolo\' path]);
diary cmd_line.txt
cd ..\..\;

%% Initialize the UDP connection
LOCAL_PORT  = cast(single(8000+50000*rand()),'uint16');
REMOTE_IP   = '192.168.137.183';
REMOTE_PORT = 10000;
TIMEOUT     = 8;                % Depending on the transmit period of the
    remote device (ms)
MSG_LENGTH  = 10;               % Maximum datagram size (bytes)

% Initialize the communication object
raspio = udp_message(LOCAL_PORT,REMOTE_IP,REMOTE_PORT,TIMEOUT,MSG_LENGTH);

%% Setup the loop
sim_k = 200000;
M = 5;

for m=1:M
    telapsed = zeros(1,sim_k);
    % Send data to MCU
    raspio.start();
    freq = [ 10 10 10 10];
    warn(m) = 0;
    error_n(m) = 0;
    for k=1:sim_k
        channel = randi([0 3],1,1);

        %warning('off');
        tstart = tic;
        aux = raspio.setFreqPWM(channel,freq(channel+1));
        telapsed(k) = toc(tstart);
        %warning('on');

        if( isnan(aux) )
            fprintf('Error: %d\t Ch= %d\t Freq= %d expected= %d\n',k,channel,
                freq(channel+1), raspio.frequency);
            error_n(m) = error_n(m)+1;
            telapsed(k) = nan;
        else
            if( aux ~= 100*floor(freq(channel+1)/100) )
                warning('Ch %d\tk %d\tExpeted %d and received
                    %d\n',channel,k,freq(channel +1), aux);
                warn(m) = warn(m)+1;
            end
```

```
49          end
50
51          freq(channel+1) = freq(channel+1) +10;
52          if( freq(channel+1) > 2^16 )
53              freq(channel+1) = 0;
54          end
55
56
57  %           if( mod(k,10000) == 0)
58  %               fprintf('k = %d\n',k);
59  %           end
60      end
61      raspio.end();
62      time(m,:)= telapsed *1000;
63      figure;
64      h(m) = histogram(time(m,:));
65      med(m) = nanmean(time(m,:));
66      desvio(m) = nanstd(time(m,:));
67      maximo(m) = nanmax(time(m,:));
68      str = sprintf('Media: %fms \tDesvio: %fms\tMaximo: %fms\tWarning:
            %d\tError:
            %d\n----------------------------\n\n',med(m),desvio(m),maximo(m),warn(m),error_n(m));
69      disp(str);
70      pause(2);
71  end
72
73  cd (['test_protocolo\' path]);
74  for k=1:M
75    saveas(h(k),sprintf('figure_%d.fig',k))
76    saveas(h(k),sprintf('figure_%d.png',k))
77  end
78  clearvars -except time med desvio maximo warn error_n
79  save('workspace.mat');
80  datestr(now,'dd-mm-yy_HH-MM');
81  diary off
82  cd ..\..\;
```

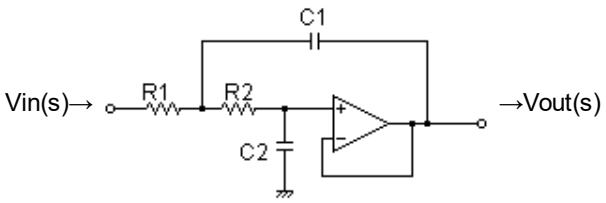
APPENDIX E

Second Order plant for RST controller

OKAWA Electric Design

Sallen-Key Low-pass Filter Design Tool - Result -

Calculated the Transfer Function for the Sallen-Key Low-pass filter, displayed on graphs, showing Bode diagram, Nyquist diagram, Impulse response and Step response.



Transfer Function

$$G(s)=\frac{44.0917107584}{s^2+5.48941798942s+44.0917107584}$$

R1 = 560kΩ
R2 = 270kΩ
C1 = 1uF
C2 = 0.15uF

Sallen-Key Low-pass Filter

Cut-off frequency
fc = 1.05681411834[Hz]

Quality factor
Q = 1.20962895403

Damping ratio
ζ = 0.413349894061

Pole(s)
p = -0.436834003857 +0.962305634291i[Hz]
$|p|$= 1.05681411834[Hz]
p = -0.436834003857-0.962305634291i[Hz]
$|p|$= 1.05681411834[Hz]

Phase margin
pm= 71.6[deg] (f =1.2[Hz])

Oscillation frequency
f = 0.962305634291[Hz]

Overshoot (in absolute value)
The 1st peak g_{pk} = 1.24 (t =0.52[sec])
The 2nd peak g_{pk} = 0.94 (t =1.05[sec])
The 3rd peak g_{pk} = 1.01 (t =1.55[sec])

Final value of the step response (on the condition that the system converged when t goes to infinity)
$g(\infty)$ = 1

f_c= 1 Hz

Q factor | Damping ratio ζ
- (•) Quality factor Q = 1.2
- ( ) Damping ratio ζ = 0.5

C1 = 1u F C2 = 150n F

C1, C2 is optional. But when setting these capacitances, C1 and C2 of both are needed to give following the equation
$(C2/C1) \le \zeta^2$
$(C1/C2) \ge 4Q^2$

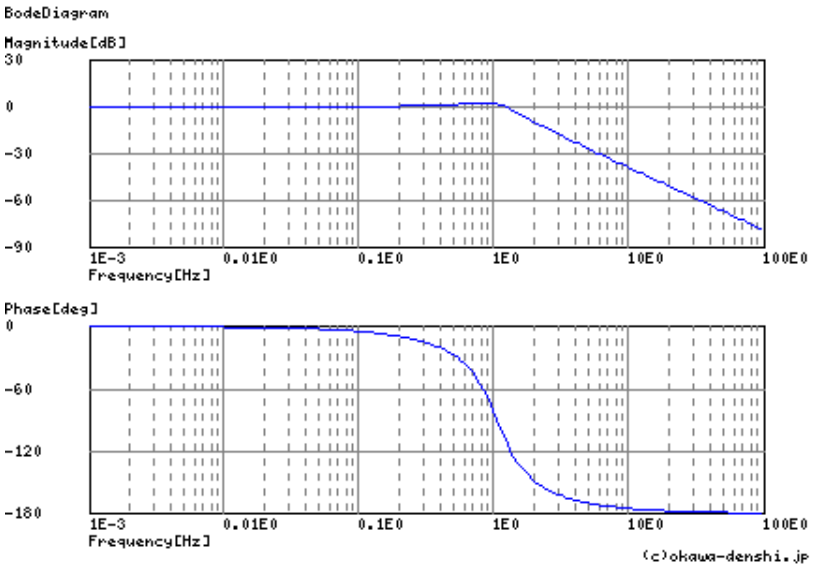
Select Capacitor Sequence: E6
Select Resistor Sequence: E12

Frequency analysis
- [x] Bode diagram
- [ ] Nyquist diagram
- [x] Pole, zero
- [x] Phase margin
- [x] Oscillation analysis

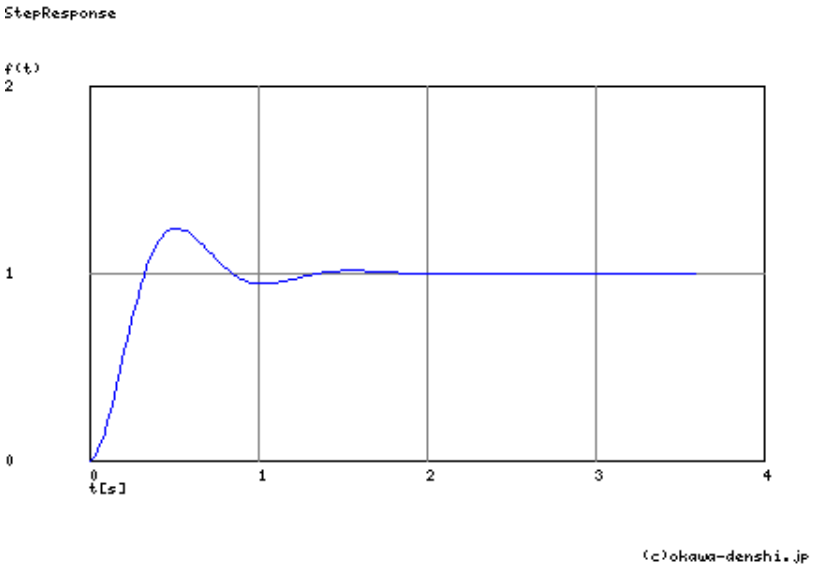
Transient analysis
- [x] Step response [ ] Impulse response
- [x] Overshoot
- [x] Final value of the step response

Calculate

Frequency analysis



Transient analysis



Top

♦**Suggestion box**

We'll use your suggestion to improve site quality in future.

Submit

APPENDIX F

List of functions supported by code translator

The functions supported by code translator can be found on file 'fixFunction.m'. Here is a preview of the file:

```
%Add here:
%                          MATLAB FUNCTION      |                        C FUNCTION
%                  return    function_name       |return      function_name          arg1    arg2   arg3   arg4   ...
FUNCTION_LIST = {{'',        'obj.start',        '',        'MAX11300init',        ''};
                 %{'' ,       'obj.end' ,         '' ,       'rt_task_delete' ,     'NULL'};
                 {'' ,       'obj.end' ,         '' ,       'return' ,      ''};
                 {'ret' ,    'obj.getADC' ,      '' ,       'getADC' ,             '(uint8_t)arg1' ,    '&ret' };
                 {'ret' ,    'obj.getADCAll' ,   '' ,       'getADCAll' ,          'ret'};
                 {'' ,       'obj.setDAC' ,      '' ,       'setDAC' ,             '(uint8_t)arg1' ,    'arg2'};
                 {'' ,       'obj.setDACAll' ,   '' ,       'setDACAll' ,          'arg1'};
                 {'ret' ,    'obj.getIN' ,       '' ,       'getIN' ,              '(uint8_t)arg1' ,    '&ret'};
                 {'ret' ,    'obj.getINAll' ,    '' ,       'getINAll' ,           'ret'};
                 {'' ,       'obj.setOUT' ,      '' ,       'setOUT' ,             '(uint8_t)arg1' ,    'arg2'};
                 {'' ,       'obj.setOUTAll' ,   '' ,       'setOUTAll' ,          'arg1'};
                 {'' ,       'rls_am' ,          '' ,       'rls' ,                '(uint8_t)arg5' ,    'arg1' ,    'arg2' ,↙
'arg3',  'arg4' , 'arg6'};
                 {'ret' ,    'zeros' ,           '' ,       'memset' ,             'ret' ,     '0' ,       'sizeof(ret)'};
                 {'ret' ,    'sizeof' ,          'int ret' , 'sizeof' ,             'arg1'};
                 {'ret' ,    'length' ,          'int ret' , 'length' ,             'arg1'};
                 {'' ,       'pause' ,           '' ,       'myDelay' ,            'arg1'};
                 {'' ,       'set_periodic' ,    '' ,       'rt_task_set_periodic', 'NULL' ,    'TM_NOW' ,↙
'arg1*1000000000'};
                 {'' ,       'wait_period' ,     '' ,       'rt_task_wait_period' , 'NULL'};
                 {'' ,       'tic' ,             '' ,       '' ,                    '' ,  };
                 {'' ,       'toc' ,             '' ,       '' ,                    '' ,  };
                 %function from math.h
                 {'ret' ,    'cos' ,           'ret' ,      'cos' ,                'arg1'};
                 {'ret' ,    'sin' ,           'ret' ,      'sin' ,                'arg1'};
                 {'ret' ,    'tan' ,           'ret' ,      'tan' ,                'arg1'};
                 {'ret' ,    'abs' ,           'ret' ,      'fabs' ,               'arg1'};
                 {'ret' ,    'sqrt' ,          'ret' ,      'sqrt' ,               'arg1'};
                 {'ret' ,    'exp' ,           'ret' ,      'exp' ,                'arg1'}};
```

Bibliography

[1] (Jun. 2016). Regulation Of Body Temperature, [Online]. Available: http://163.178.103.176/tema1g/grupos1/germant1/gatp2/b/fever.htm.

[2] R. Schreiber. (2007). MATLAB.

[3] (May 2016). About, LabJack, [Online]. Available: https://labjack.com/about.

[4] (May 2016). 25.0 Scripting, LabJack, [Online]. Available: https://labjack.com/support/datasheets/t7/scripting.

[5] (May 2016). Key Advantages Of T7 LUA Scripting, LabJack, [Online]. Available: https://labjack.com/news/key-advantages-t7-lua-scripting.

[6] *Analog Discovery™ Technical Reference Manual*, version C, Linear Technology Corporation, Mar. 2015. [Online]. Available: https://reference.digilentinc.com/_media/analog_discovery:analog_discovery_rm.pdf.

[7] Mathworks. (May 2016). MATLAB Support Package For Raspberry Pi Hardware, [Online]. Available: http://www.mathworks.com/help/supportpkg/raspberrypiio/index.html.

[8] (May 2016). MATLAB Support Package For BeagleBone Black Hardware, Mathworks, [Online]. Available: http://www.mathworks.com/help/supportpkg/beagleboneio/index.html.

[9] (May 2016). MATLAB Support Package For Arduino Hardware, Mathworks, [Online]. Available: http://www.mathworks.com/help/supportpkg/arduinoio/index.html.

[10] (May 2016). Analog Input Using SPI, Mathworks, [Online]. Available: http://www.mathworks.com/help/supportpkg/raspberrypiio/examples/analog-input-using-spi.html.

[11] (Mar. 2008). MCP3004/3008, [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21295d.pdf.

[12] (May 2016). Use The BeagleBone Black ADC To Capture Analog Data, Mathworks, [Online]. Available: http://www.mathworks.com/help/supportpkg/beagleboneio/ug/use-the-beaglebone-black-adc-to-capture-analog-data.html.

[13] MakerZone. (Jun. 2016). Using MATLAB And Arduino To Acquire Analog Signals, [Online]. Available: http://www.mathworks.com/videos/using-matlab-and-arduino-to-acquire-analog-signals-100739.html.

[14] (May 2016). MATLAB Coder - Generate C And C++ Code From MATLAB Code, Mathworks, [Online]. Available: http://www.mathworks.com/products/matlab-coder/.

[15] (Jun. 2016). Maximum Data Rates for the LabJack U12, LabJack, [Online]. Available: https://labjack.com/u12/datarates.

[16] (Jun. 2016). 3.1 - Command/Response, LabJack, [Online]. Available: https://labjack.com/support/datasheets/u6/operation/command-response.

[17] (Jun. 2016). A-1 Data Rates, LabJack, [Online]. Available: https://labjack.com/support/datasheets/t7/appendix-a-1.

[18] P. Pedreiras. (May 2016). Scheduling Basics, [Online]. Available: http://ppedreiras.av.it.pt/resources/str1516/docs/STR-4.pdf.

[19] (May 2016). Real-Time Linux Wiki, [Online]. Available: https://rt.wiki.kernel.org/.

[20] J. Davies. (Jan. 2012). TCP/IP Fundamentals For Microsoft Windows, [Online]. Available: https://download.microsoft.com/download/9/4/6/946958ef-7b86-4ddc-bfdb-c7ed2af4ce51/TCPIP_Fund.pdf.

[21] (Dec. 2012). IEEE Standard for Ethernet.

[22] *SPI Protocol And Bus Configuration Of Multiple DCPs*, Intersil, Aug. 2007. [Online]. Available: http://www.intersil.com/content/dam/Intersil/documents/an13/an1340.pdf.

[23] *UM10204 - I2C-Bus Specification And User Manual*, version 6, NXP, Apr. 2014. [Online]. Available: http://www.nxp.com/documents/user_manual/UM10204.pdf.

[24] «6. PID Controller Design», in *Linear Feedback Control*, ch. 6, pp. 181–233. DOI: 10.1137/1.9780898718621.ch6. eprint: http://epubs.siam.org/doi/pdf/10.1137/1.9780898718621.ch6. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9780898718621.ch6.

[25] (Jul. 2016). PID Theory Explained, National Instruments, [Online]. Available: http://www.ni.com/white-paper/3782/en/.

[26] A. M. Mota, «Controladores PID», in *Sistemas de Controlo 2 - Texto de Apoio*, Beta 0.2. Apr. 2014.

[27] V. Rajinikanth and K. Latha. (Jun. 2016). I-PD Controller Tuning for Unstable System Using Bacterial Foraging Algorithm: A Study Based on Various Error Criterion, [Online]. Available: http://www.hindawi.com/journals/acisc/2012/329389/.

[28] A. M. Mota, «Identificação De Modelos Discretos Através Do Método Dos Minimos Quadrados», in *Sistemas de Controlo 2 - Texto de Apoio*, Beta 0.2. Apr. 2014.

[29] T. Cunha and A. Mota. (Jan. 2016). System Identification Based On Recursive Least Squares Estimation.

[30] A. M. Mota, «Controladores RST», in *Sistemas de Controlo 2 - Texto de Apoio*, Beta 0.2. Apr. 2014.

[31] (May 2016). Model B Hardware General Specifications, [Online]. Available: www.raspberry-projects.com/pi/pi-hardware/raspberry-pi-model-b/hardware-general-specifications.

[32] (May 2016). About Debian, [Online]. Available: https://www.debian.org/intro/about.

[33] *MAX11300*, version 3, Maxim Integrated Products, Mar. 2014. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX11300.pdf.

[34] (May 2016). Programmable Analog: Maxim Creates Swiss Army Knife, EETimes, [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1322790.

[35] *MAX31790*, version 2, Maxim Integrated Products, Dec. 2012. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX31790.pdf.

[36] *74HC244*, version .5, NXP Semiconductors N.V., Dec. 1990. [Online]. Available: http://www.nxp.com/documents/data_sheet/74HC_HCT244.pdf.

[37] *TL071*, Texas Instruments, Sep. 1978. [Online]. Available: http://www.ti.com/lit/ds/symlink/tl071.pdf.

[38] *OP27*, version H, NXP Semiconductors N.V. [Online]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/OP27.pdf.

[39] *LT3471*, version B, Linear Technology Corporation. [Online]. Available: http://cds.linear.com/docs/en/datasheet/3471fb.pdf.

[40] (May 2016). SPI Devices, [Online]. Available: https://www.kernel.org/doc/Documentation/spi/spidev.

[41] (May 2016). SPI Driver Latency, [Online]. Available: http://elinux.org/RPi_SPI#SPI_driver_latency.

[42] M. McCauley. (Jan. 2016). C Library for Broadcom BCM2835 as Used in Raspberry Pi, [Online]. Available: http://www.airspayce.com/mikem/bcm2835/index.html.

[43] (Feb. 2016). EV KIT SOFTWARE. version 1.2, [Online]. Available: https://www.maximintegrated.com/en/design/tools/applications/evkit-software/index.mvp/id/1196.

[44] *Sample-And-Hold Amplifiers - MT-090 TUTORIAL*, Analog Devices. [Online]. Available: http://www.analog.com/media/cn/training-seminars/tutorials/MT-090.pdf.

[45] W. S. Levine, *THE CONTROL HANDBOOK*, ser. The electrical engineering handbook series. Boca Raton (Fl.): CRC Press New York, 1996, ISBN: 0-8493-8570-9. [Online]. Available: http://opac.inria.fr/record=b1079196.

[46] Amplicon. (Jun. 2016). 10 Key Benefits of Industrial Ethernet, [Online]. Available: https://www.amplicon.com/docs/Articles/10-key-benefits-of-Industrial-Ethernet-Tech-Article.pdf.

[47] K. Bartlett. (Mar. 2010). A Simple UDP Communications Application, [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/24525.

[48] R. Antão, «Typ-2 Fuzzy Logic: Uncertain System's Modeling And Control», PhD thesis, Universidade de Aveiro, Department of Electronics, Telecommunications and Informatics, 2016.

[49] K. Bartlett. (Aug. 2014). SSH/SFTP/SCP For Matlab. version 1.10, [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/35409-ssh-sftp-scp-for-matlab--v2-.

[50] (Jun. 2007). 287/289 Users Manual, [Online]. Available: http://media.fluke.com/documents/287_289_umeng0200.pdf.

[51] (Apr. 2016). Sallen-Key Low-pass Filter Design Tool, OKAWA Electric Design, [Online]. Available: http://sim.okawa-denshi.jp/en/OPseikiLowkeisan.htm.