Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials (GETI)

Controllers implementation in low-cost platforms

MEMORANDUM

January 2023

Author: Iván Barrachina Sabariego

Director: Carlos Ocampo-Martínez

Call: 02/2023

ETSEIB

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

UPC

Summary

Català

L'objectiu d'aquest treball és implementar un controlador en una placa basada en Arduino per controlar una maqueta del sistema *ball and beam* i desenvolupar una documentació que pugui ajudar estudiants tant del Grau com del Màster d'Enginyeria Industrial de l'ETSEIB (o qualsevol persona amb coneixements bàsics de control automàtic) a entendre el funcionament del controlador i com implementar-lo en un entorn real.

Aquesta memòria documenta els diferents processos que s'han seguit per implementar el controlador. Aquests processos inclouen el muntatge del model físic, incloent els esquemàtics del circuit elèctric o les característiques més rellevants dels components, la inicialització i configuració de la placa i la sintonització dels seus perifèrics. La memòria també inclou l'abstracció matemàtica del model físic, la qual és la clau del càlcul del controlador que finalment serà implementat a la placa Arduino i evaluat.

Castellano

El objetivo de este trabajo es implementar un controlador en una placa basada en Arduino para controlar una maqueta del sistema *ball and beam* y desarrollar una documentación que pueda ayudar a estudiantes tanto del Grado como del Máster de Ingeniería Industrial de la ETSEIB (o a cualquier persona con conocimientos básicos de control automático) a entender el funcionamiento del controlador y como implementarlo en un entorno real.

Esta memoria documenta los diferentes procesos que se han seguido para implementar el controlador. Estos procesos incluyen el montaje del modelo físico, incluyendo los esquemáticos del circuito eléctrico o las características más relevantes de los componentes, la inicialización y configuración de la placa y la sintonización de sus periféricos. La memoria también incluye la abstracción matemática del modelo físico, la cual es la clave del cálculo del controlador que finalmente será implementado en la placa Arduino y evaluado.

English

The goal of this project is to implement a controller in an Arduino based board in order to control a low-cost *ball and beam* model, and to develop a series of documentation which can help students from either ETSEIB Industrial Engineering Bachelor's or Master's Degree -or anyone with basic automatic control knowledge- understand the controller's functioning and how to implement it into a real environment.

This memory documents the different processes which have been followed in order to implement the controller. These processes include the model's physical layout assembly, including the electric circuit's schematics or the components' most relevant data, the initialization and configuration of the board and the tuning of its peripherals. It also includes the mathematical abstraction of the physical model, which is the key to the calculation of the controller which will be then implemented in the Arduino board and tested.

ETSEIB

Contents

List of Figures

List of Tables

1 Preface

This project was started by ETSEIB Automatic Control Professor Carlos Ocampo-Martínez alongside two of his former students, Santi Prats Moreu and Eduard Morera Torres as a free-time project whose objective was to build a physical ball and beam model from common and cheap materials such as wooden planks, screws or glue and to control it using widely used hardware and control techniques such as the PID controller structure. They managed to build a couple of models and to successfully implement three functioning controllers which could stabilize the system as desired using the PID, State Feedback and MPC control structures before Santi and Eduard ended their studies.

The project was later resumed by the author when he addressed Professor Ocampo-Martínez looking for ideas to develop his TFG (Bachelor Studies Final Project). This project phase's idea, as will be explained further later, is to follow the controller calculation and implementation steps in order to create didactic and educational materials for other Automatic Control students in ETSEIB.

2 Introduction

2.1 Project goals

The goal of this project is to use low-cost platforms to implement a controller to control a *ball and beam* system, which consists of a ball which can move along a straight beam in a one degree of freedom movement. The beam, at the same time, has another degree of freedom set by the beam angle, which can be set through a servomotor. Thus, the global system has two degrees of freedom that will be governed by controlling the angle of the beam and by limiting the position of the ball with two wooden pieces placed at the end of the beam.

The controller's function is to drive the ball to the beam's central point or to maintain it there if it is already in that position. The controller will receive the ball's position signal using two infrared position sensors located at each end of the beam and will send the required signal to the servomotor so it can rapidly correct the beam's angle as necessary to make the ball slide from its previous position to the desired one. The controller structure that will be used to achieve this goal is the PID controller, which will be presented with more detail later.

This project will also include a comparison between several PID controllers with different characteristics in their specifications or their external conditions -by changing the ball for another ball with different characteristics-. These different controllers will be implemented to compare their performance, which will be evaluated by observing how precise, fast and stable is the final position of the ball compared with the desired state of the system.

The main goal, though, is to synthesize and keep record of all this process so the generated documentation can be used to introduce this system to ETSEIB's Automatic control students in a way it helps them to understand how these controllers work and how they are implemented physically in a practical example. For this reason, the final documentation will be presented in a practice script format which will be included in the memory annex, as it is how Automatic Control professors introduce practical examples to students.

2.2 Project scope

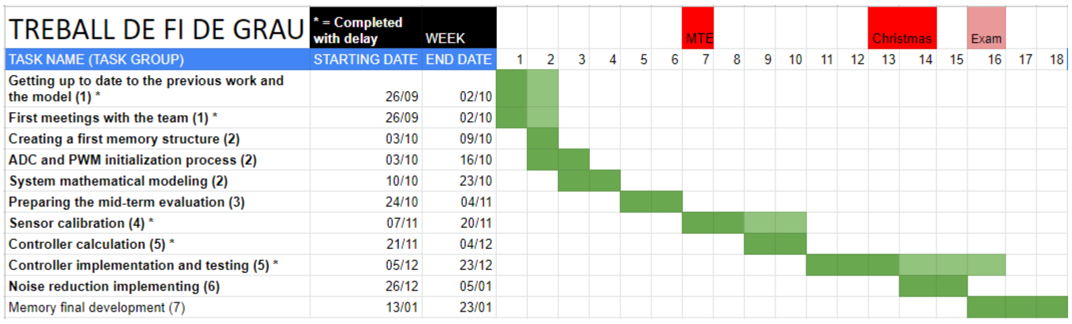
As explained previously, this project's scope is to design and implement these two controllers and to test their performance so they behave as expected. This process will be considered as successful if the controllers are able to keep the ball at the center of the beam with a maximum error of ± 2 centimeters and if they show resistance to perturbations on the model, such as the presence of an obstacle which obstructs the beam or a disturbance in the ball's position due to a person's action on the model.

The scope also includes the final approval of the developed documentation by an ETSEIB automatic control professor, so it is labeled as quality content which could be used anytime for educational or informative purposes. This documentation will be prepared in order to make students synthesize everything that has been seen during the project, making a special emphasis on the implementation and testing of the controllers, which is essentially what students are not particularly familiar with.

2.3 Project planning

The project will be completed during the span of four months, starting on September 2022 and ending in late January 2023. An estimate project planning has been prepared at the beginning in order to have a reference of the project's times, although, since there are many programs, processes and concepts to learn and to get adapted to, the periods are subject to alterations due to unexpected problems or difficulties.

The planning has been illustrating with a Gantt diagram, which includes the main tasks -classified in several groups of tasks that are related to each other or have overlapped- carried out during the eighteen weeks that the project's span occupies. The planning has also made taking into account external events and circumstances, such as holidays or other university commitments such as exams or other projects. In this Gantt diagram each task has been assigned an starting and an end date, although the tasks that have required more time have been highlighted with an asterisk and a lighter shade of color for the extra time.



TREBALL DE FI DE GRAU	* = Completed with delay	WEEK							MTE						Christmas			Exam		
TASK NAME (TASK GROUP)	STARTING DATE	END DATE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Getting up to date to the previous work and the model (1) *	26/09	02/10	🟩	▨																
First meetings with the team (1) *	26/09	02/10	🟩	▨																
Creating a first memory structure (2)	03/10	09/10		🟩																
ADC and PWM initialization process (2)	03/10	16/10		🟩	🟩															
System mathematical modeling (2)	10/10	23/10			🟩	🟩														
Preparing the mid-term evaluation (3)	24/10	04/11					🟩	🟩												
Sensor calibration (4) *	07/11	20/11							🟩	🟩	▨	▨								
Controller calculation (5) *	21/11	04/12									🟩	🟩								
Controller implementation and testing (5) *	05/12	23/12											🟩	🟩	🟩	▨	▨	▨		
Noise reduction implementing (6)	26/12	05/01														🟩	🟩			
Memory final development (7)	13/01	23/01																🟩	🟩	🟩

Figure 1: The project's Gantt diagram.

The weekly hours spent in the project have not been constant though its span, but an estimated average of 10 hours per week is considered as accurate for the whole project duration.

2.4 State of the art

The ball and beam model is one of the most widely known systems in automatic control since it is a simple example of an open-loop unstable system which requires a controller to acquire stability. Because of this main characteristic, it has become an excellent exercise for beginners in this field of knowledge, as it is a really interesting but accessible tool to practice the design of controllers and the calculation of mathematical models from physical systems which can be built and assembled following a low-cost DIY (do it yourself) philosophy. This is the reason why a huge number of teachers and educators use it as a practical example of the use of automatic control for students without needing a big budget, which is often the case these professionals face during their classes. This philosophy, apart from being the philosophy which has been followed during this project is the way lots of online science and tech content creators have been approaching during recent years to bring a wide variety of fields to the massive public of the Internet and encourage people to discover more themselves.

3 Model components, assembly and software

This project's model was created before its start by ETSEIB professor Carlos Ocampo-Martinez and two of his former students, Santi Prats Moreu and Eduard Morera Torres, since the creation of the model and its controllers was originally conceived as a past-time activity, not an academic one. Despite the fact that all the components, then, were provided and assembled together by them, it is necessary to list what materials make up the model, their function in the model and how they are united together to form the final system.

3.1 Model components

The model is composed of five distinctive basic elements which are interconnected by a series of connections and connectors:

1. **The physical model,** which consists on a V-shaped wooden beam (1) fixed to a wooden base (2) and articulated by two wooden articulations (3) and a set of metallic screws and joints. The beam incorporates two wooden pieces (4) which limit the ball's movement to 150 millimeters in each direction from the center of the beam. The beam also incorporates two wooden supports (5) at the end of the slide for the infra-red sensors and a small wooden box (6) to accommodate the servo-motor.

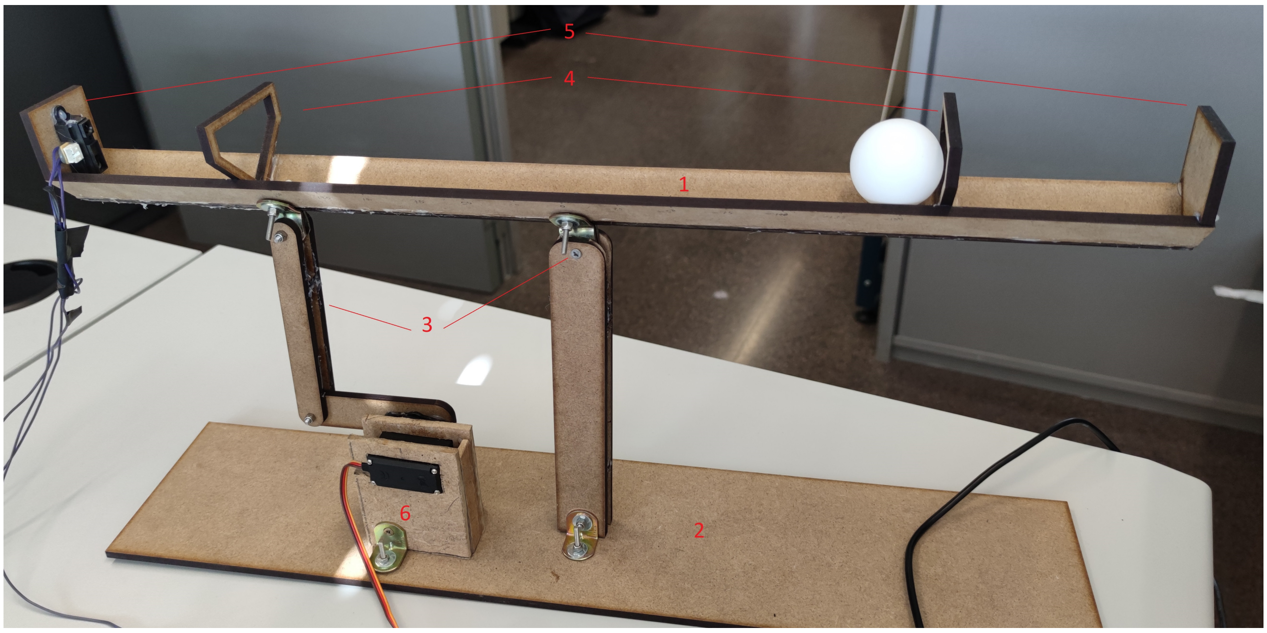


Figure 2: An image of the model studied in the project with numbered parts.

2. **Two ping-pong balls,** whose function is to slide down the beam, with whom they are connected via friction. The reason why there are two of them is that, since they have different diameters, weights, friction coefficients and moments of inertia, it is possible to study how the system's behaviour changes when its conditions are changed, and therefore, to study its robustness.

3. **Two SHARP GP2Y0A21 infrared distance sensors,** which calculate the distance between them and the ball to determine its position measuring the time it takes to an infrared light beam to get to the ball and return to the sensor. Then, they send an analogical 1-3.3 V

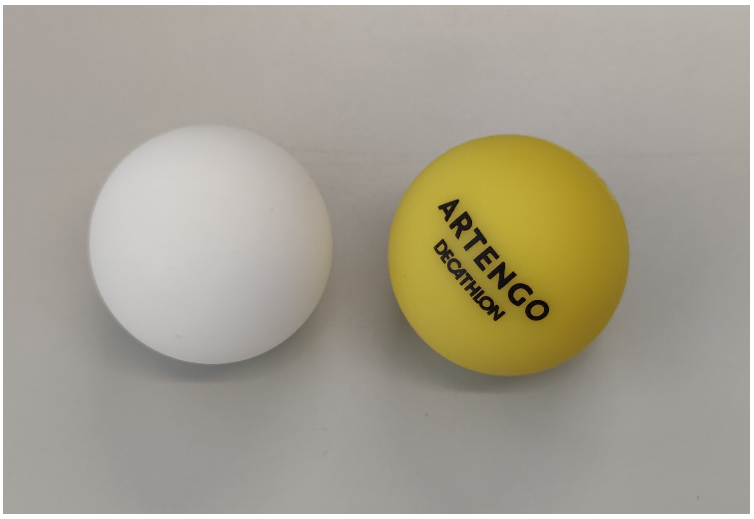


Figure 3: The two ping-pong balls that have been used. The yellow one is larger, heavier and has a higher friction coefficient.

electric signal which is proportional to the distance to the micro-controller.

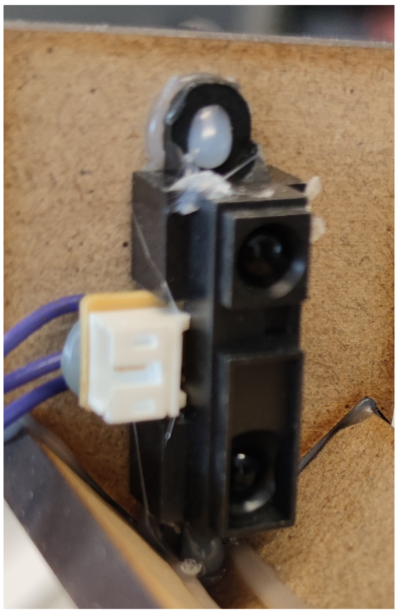


Figure 4: The infrared distance sensor located at the left part of the model.

This analogical electric signal, though, does not present a linear correspondence to the measured distance. The theoretical correspondence graph is presented next:

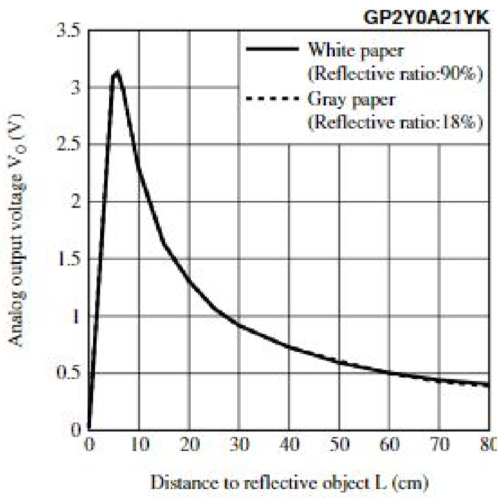


Figure 5: Sensors' theoretical relative distance to analog value graph.

However, since each sensor has its own variability, an experimental graph has been done for each sensor during their calibration. The ball's relative distance to the setpoint to analog value -proportional to the analog electric signal- graph for Sensor 1 is presented next:

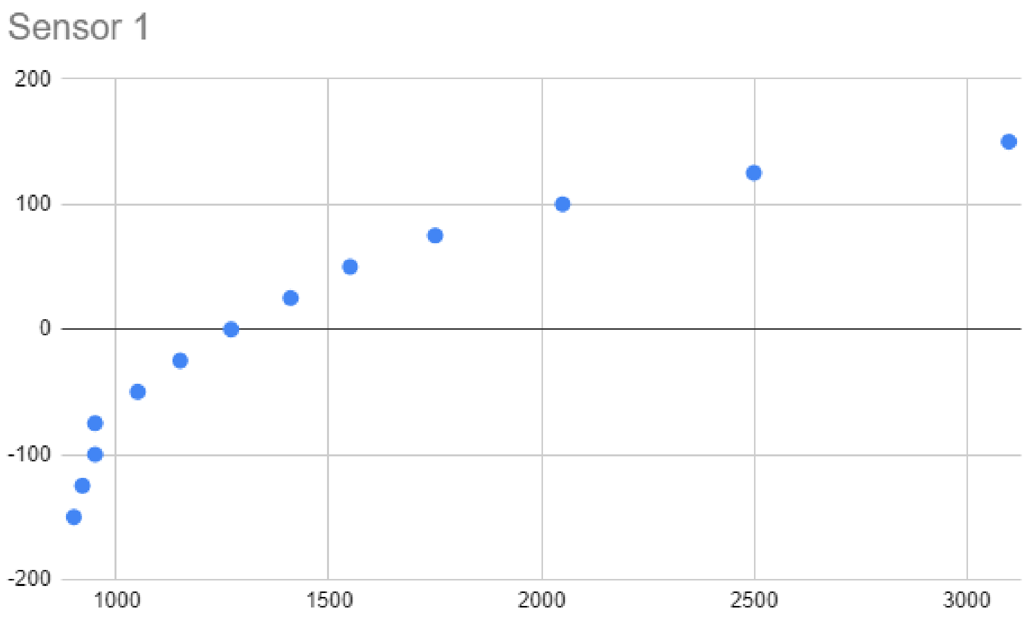


Figure 6: Sensor 1's relative distance to analog value graph.

This relative distance, though, is not the same as the sensor's measured distance because the sensor is located at the end of the beam. In this case, the higher the relative distance, the closer to the sensor the ball is. With Sensor 2 the opposite case happens, as it is located at the other end of the beam. Its graph is presented next:

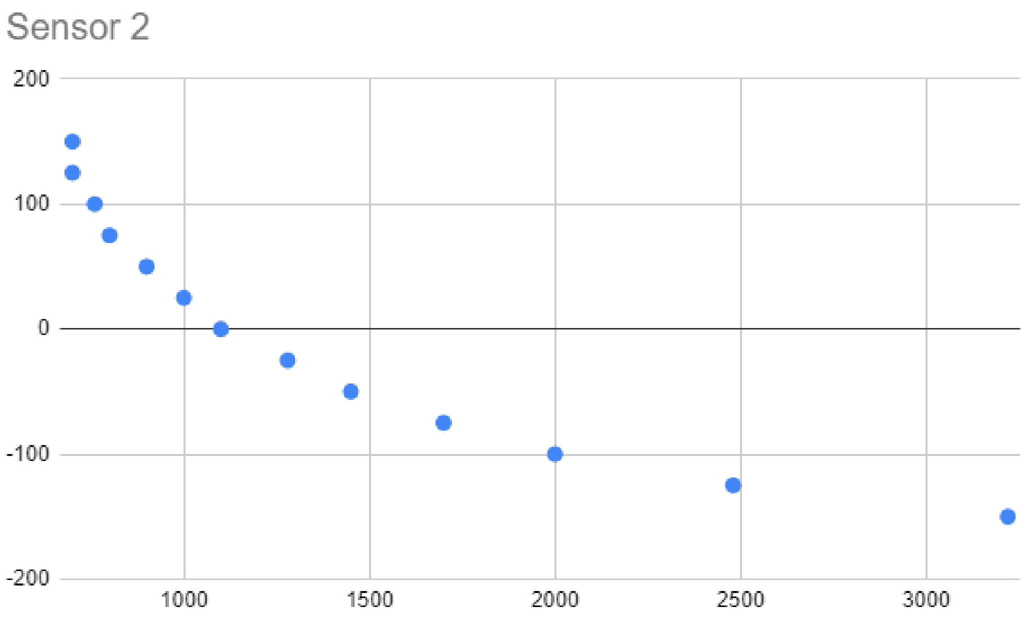


Figure 7: Sensor 2's relative distance to analog value graph.

It has been proved that the measured distance to analog signal relation is not linear. This will make the distance measurement process a bit more complex, as it is shown in Section 5.6

4. **A Futaba S3003 servo-motor,** the model's actuator, which receives a control signal from the micro-controller's PWM pins. This control signal determines how the motor will rotate so the beam angle changes. This is possible because the motor shaft is mechanically attached to the rotating articulation of the beam.

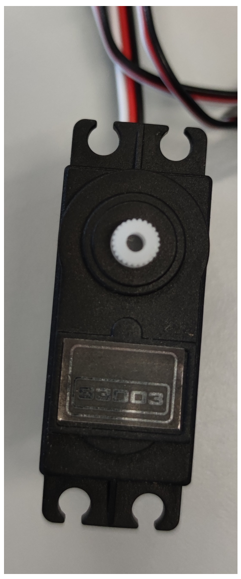


Figure 8: One of the replacement Futaba S3003 servo-motors.

The motor's motion is controlled via pulse-width modulation (PWM), which consists on discretizing an electric signal in two unequal parts to reduce its average power. In one of the parts the signal delivers its nominal power, while in the other it shuts down. The proportion between them is called the duty cycle and it is what determines the movement of the servo-motor. The S3003 servo functions with a 50 Hz pulse frequency and a 5-10%, as showed in the following figure:

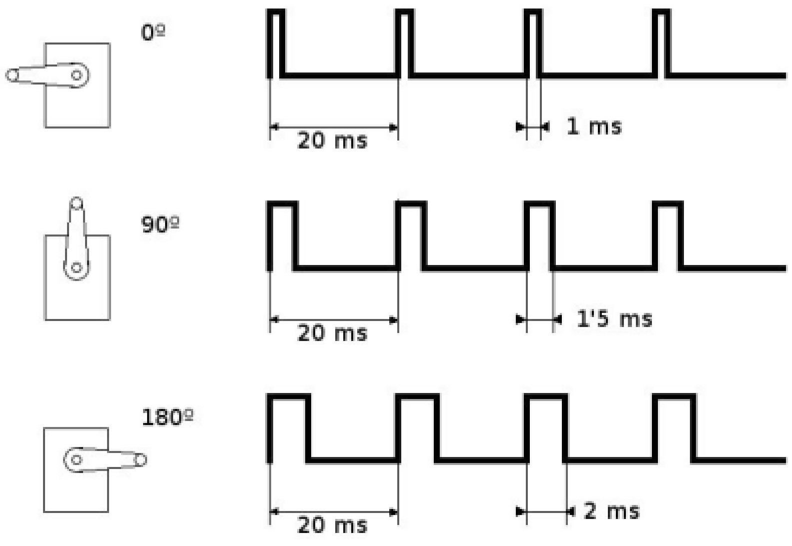


Figure 9: Diagram which shows the relationship between the PWM duty cycle and the motor's position.

As it can be seen, a duty cycle of 5% will maintain the motor's angle at the minimum angle, while a duty cycle of 10% will drive it to the maximum angle, which is 180º. This will be important when initializing the board's PWM pins at Chapter 4.

It is remarkable, though, that the real duty cycles obtained by experimentation and by adjusting the motor's edge position correspond to the 2.5-8.5% duty cycle range. If the motor received a signal with a duty cycle regime out of this range it could get saturated and even be damaged.

5. **A 64 pin STM32F411RET6 micro-controller** compatible with Arduino, which is the center of the system and the key element that stabilizes the ball at the desired position. It receives and processes the signal coming from the sensors through its analogical input pins, it also controls the motor's position modifying the duty cycle of the PWM signal it sends through the PWM output pins, as explained before.

6. **Hardware filters**, which were deemed as necessary the first time the sensor calibration process was started. The sensor readings contained an important noise component in their values that could not be attenuated by software moving average filters. This caused the sensor calibration to be less accurate and thus the ball's positioning more unstable to the point it seriously conditioned the controller's validity.

   To improve the readings' quality, a capacitive power stabilizer and low-pass filter have been installed in the electronic circuit [1].

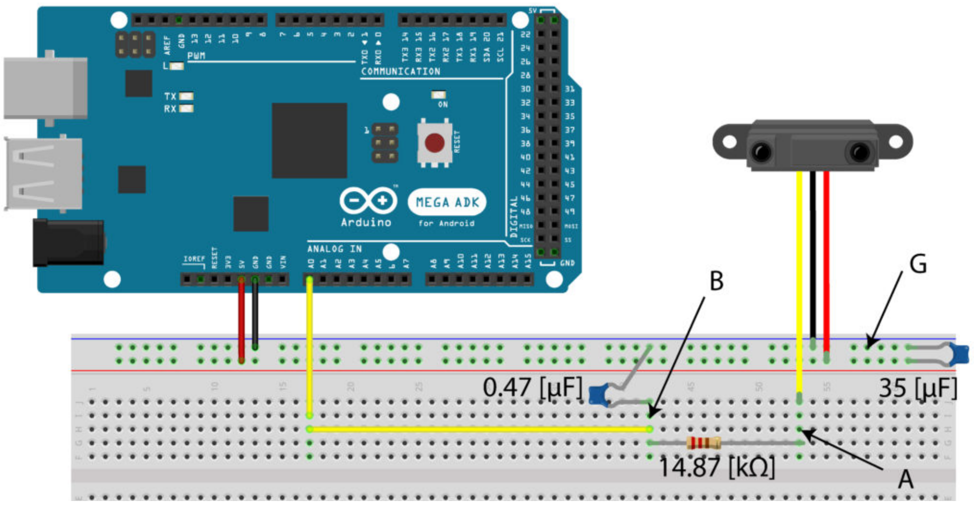


Figure 10: The sensor noise filtering circuit.

The power stabilizer consists on a 35μF capacitor between the positive and ground power lines, while the low-pass filter consists on a 14.87kΩ resistor and a 0.47μF capacitor between the signal line and the ground. However, due to availability reasons, the 35μF capacitor has been substituted by a 45μF one, while the resistor has been changed to a 22kΩ one. These substitutes proved to be the best available options compared to different magnitude components when tested.

These filters, combined with the previously mentioned software moving average filters have brought the noise factor to acceptable levels, which have made the distance calibration process viable.

3.2 Model assembly

The assembly of the model, though done prior to the project's start and by different people, is a key factor for the project's objectives, since the project's aim is to create sustainable, low-cost automatic control materials. This factor, though, also conditions the project's results, as the available models will not have the quality and reliability that could be found in commercial or industrial systems which could be studied otherwise.

The model assembly has largely consisted on the construction fixation of the wooden-made pieces, which has been done by drilling the necessary holes and creating the fixations and articulations with brass screws, washers and joints and gluing the pieces together -in the beam's case-.

The electronic circuit has been installed in a prototyping connection board and has consisted on connecting the different circuits using common electronic prototype wires to the different components through the board's pins.

3.3 Software used during the project

The project has relied heavily on the use of different software programs. In this section a brief description on their relevance for the project will be included for each of them. It is important to consider that these programs have been used in their latest versions available during the project's development, and the different processes described during this memory may be have to be done in a different way if posterior or previous versions are used.

1. **MATLAB.** This programming language and numeric computing environment has been key for making controller calculations and making the system's mathematical modelling. The model calculation scripts were already prepared by the project's previous students, but they have been adapted and corrected during the project's development. The controller calculation scripts have been fully developed during the project. All of them can be consulted in Annex B.

2. **Simulink**, a MATLAB-based modeling and simulating environment. It has been used to simulate the calculated controllers and their impact in the whole system before actually implementing them. This has been done using material prepared by former students, even though understanding the models has been considered part of the project's tasks. The Simulink model that has been used for these simulations will be shown in Section 7.1.1. Both MATLAB and Simulink have been used in their *MATLAB R2022b* version.

3. **STM32CubeIDE,** an Integrated Development Environment (IDE) which has been used for the board's initialization and configuration, but also for the C code development through its C code view. This program has allowed to generate all the settings code automatically without needing how to code all these instructions manually. This makes this software specially useful for beginner users, since even if the initialization instructions in Chapter 4 are not followed, it is possible to work with the boards using one of the many tutorials available on the Internet for many different types of applications. This software has been used in its *STM32CubeIDE 1.10.1* version.

4. For this memory's redaction, the LaTeXOverleaf interface has been used in order to make the document's arrangement easier and more uniform.

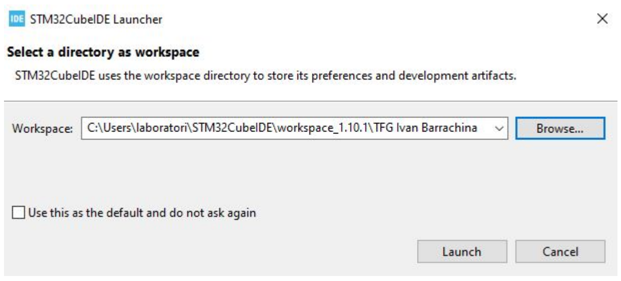
4 Board initialization and peripheral tuning

As mentioned previously in Section 3.1, the STM32F411RET6 micro-controller is the key component of the system because its software and hardware will be the nexus between the model's sensors and its actuator. Even though the controller's code implementation will be the most important step in the project's software development, there are some previous requirements which must be accomplished so the board works as desired. These steps will include a base code creation, which will create the required environment to initialize the different peripherals, the PWM timer initialization, which will configure the board's signal output, the ADC reading initialization and posterior calibration, which will set the analogical measuring, the SWD plot configuration and finally the base code generation which will include the desired settings in the final C code.

4.1 Base code creation

This initial stage consists on the creation of a configuration environment using STM32CubeIDE, an integrated development environment (IDE) which allows the user to set their board basic features without necessarily needing to have knowledge of coding in C/C++ language. To do so, it is necessary to follow these steps:

1. Creating a folder which will contain the project's workspace. Once STM32CubeIDE is opened for the first time, the program will ask for a folder to use as workspace, where the user must select the desired folder as shown in the next figure:


STM32CubeIDE Launcher

Select a directory as workspace

STM32CubeIDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\laboratori\STM32CubeIDE\workspace_1.10.1\TFG Ivan Barrachina

Browse...

☐ Use this as the default and do not ask again

Launch Cancel

Figure 11: This figure shows what will appear for this step of the initialization.

2. The program's home window appears now, where the *Start a new STM32 project* option must be selected. In this step it may be necessary to install packages.

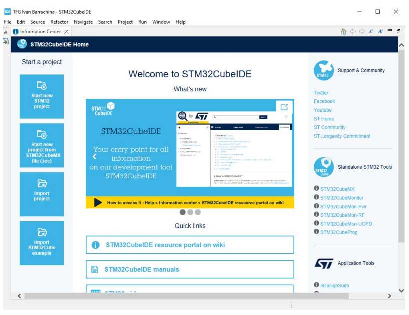
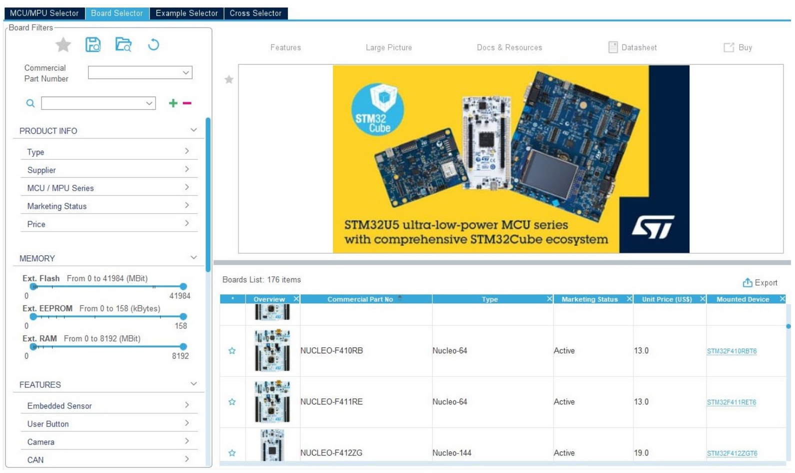


Figure 12: This figure shows what the user will see when completing Step 2.

3. After creating a new project, it is necessary to specify which board is being used for the project. In this case, the selected board must be the *Nucleo-F411RE*, which should be found in the list at the *Board Selector* tab. After that, the user only needs to click *Next* and give a name to the project. The user will now have to select the option that initializes the peripherals in their *Default Mode* and open the *Device Configuration Tool* perspective.



*	Overview	Commercial Part No	Type	Marketing Status	Unit Price (US$)	Mounted Device
☆		NUCLEO-F410RB	Nucleo-64	Active	13.0	STM32F410RBT6
☆		NUCLEO-F411RE	Nucleo-64	Active	13.0	STM32F411RET6
☆		NUCLEO-F412ZG	Nucleo-144	Active	19.0	STM32F412ZGT6

Figure 13: This figure shows where to select the correct board.

4. Now the board is ready to be configured, even though there is some default pinout configuration generated by the program, which must be deleted. To do so, select the *Clear Pinout* option at the *Pinout* tab.

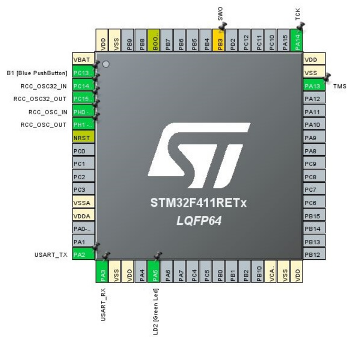


Figure 14: This figure shows what the user can see before resetting the board's pinout.

4.2 PWM timer initialization

The actuator's control regime has been described in Section 3.1. In this stage of the initialization the board will be configured to use one of its output pins as the PWM signal source for the servo-motor with the previously stated duty cycle and pulse frequency. The following steps must be followed to complete this stage:

1. Opening the *Clock configuration* tab to set the APB1 timer frequency at 16 MHz. The default status will show a wide variety of frequencies for the different timers. To solve this, it is necessary to switch the *System Clock Mux* input to HSI and set the *APB1 prescaler* to /1. The result should be the one shown at the next figure:

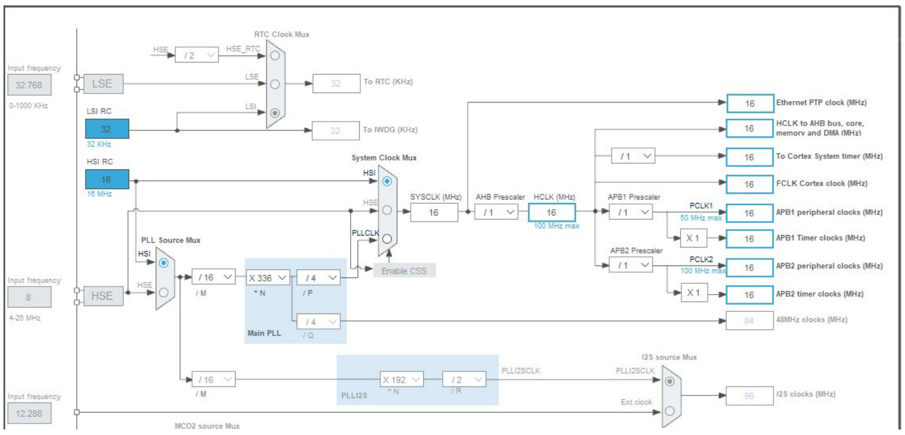
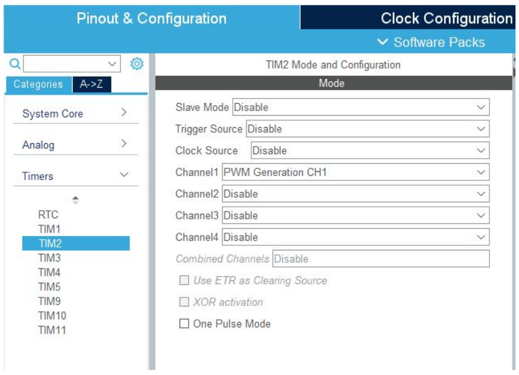


Figure 15: This should be the clock configuration status.

2. Going back to the *Pinout and Configuration* tab, then select the *Timers* dropdown to enable the *TIM2* timer, which will be achieved after setting its Channel 1 as *PWM Generation CH1*.



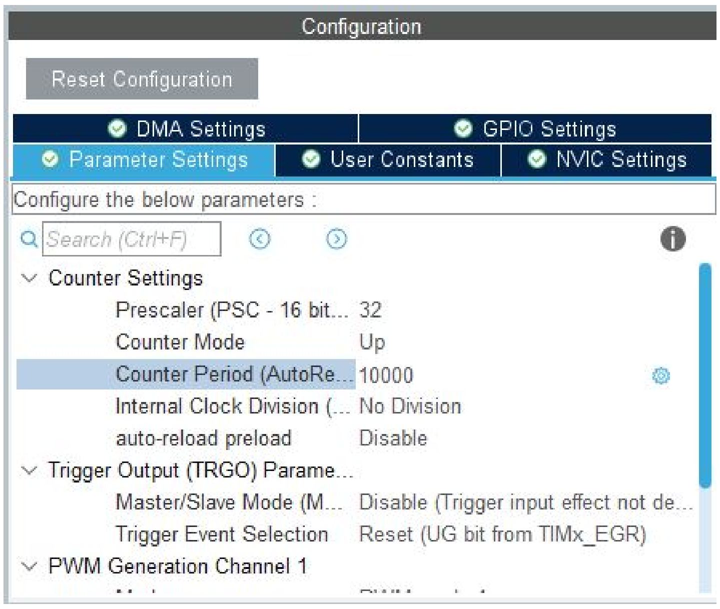
TIM2 Mode and Configuration

Mode

Slave Mode Disable
Trigger Source Disable
Clock Source Disable
Channel1 PWM Generation CH1
Channel2 Disable
Channel3 Disable
Channel4 Disable
Combined Channels Disable
☐ Use ETR as Clearing Source
☐ XOR activation
☐ One Pulse Mode

Figure 16: This figure shows how to enable the TIM2 timer.

3. The last step consists on changing the timer's prescaler and counter period. This can be done at the *Parameter Settings* tab of the timer menu. The prescaler must be set at 32 and the counter period at 10000. The prescaler and the counter period will reduce the timer's original 16 MHz frequency to the desired 50 Hz that will create 20 millisecond long pulses -the prescaler will set its value to 0.5 MHz and the counter period to the final 50 Hz-. After completing this step, the PA5 pin should be appear named as *TIM2_CH1*, which means the PWM output is tuned.


Configuration

Reset Configuration

DMA Settings | GPIO Settings

Parameter Settings | User Constants | NVIC Settings

Configure the below parameters :

Search (Ctrl+F)

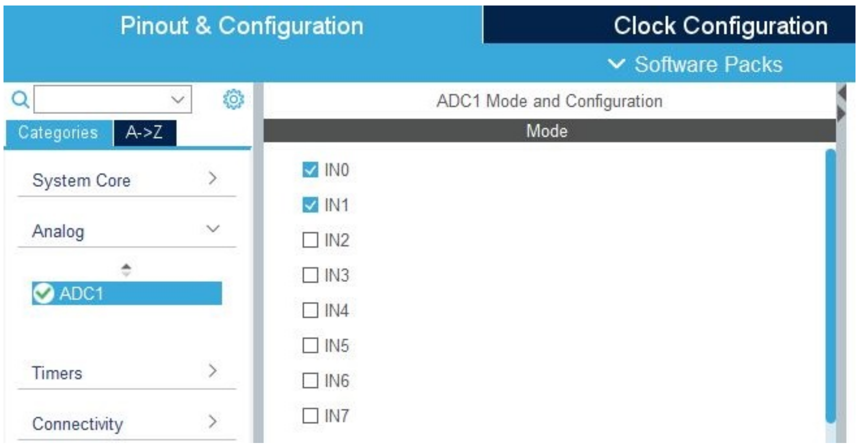
Counter Settings	
Prescaler (PSC - 16 bit...	32
Counter Mode	Up
Counter Period (AutoRe...	10000
Internal Clock Division (...	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parame...	
Master/Slave Mode (M...	Disable (Trigger input effect not de...
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
PWM Generation Channel 1	

Figure 17: This figure shows how to set TIM2's parameters.

4.3 ADC reading initialization

The next stage of the board initialization consists on the ADC reading settings. The ADC reading process is based on an analogical electric signal input into one of the board's analogical-digital converter pins, which converts these analogical values to a digital value which can be stored in the board's memory. These settings are done following the following steps:

1. Selecting the *Analog* option at the *Pinout and Configuration* tab, which will make appear the *ADC1* dropdown. After selecting that item it will be possible to select a set of ADC input modes. Since the model only has two sensors, that are the only analogical inputs the project will include, there is only need to select two inputs. Therefore, the *IN0* and *IN1* inputs will be selected.


Pinout & Configuration | Clock Configuration

Software Packs

Categories | A->Z

System Core

Analog

ADC1

Timers

Connectivity

ADC1 Mode and Configuration

Mode

- ☑ IN0
- ☑ IN1
- ☐ IN2
- ☐ IN3
- ☐ IN4
- ☐ IN5
- ☐ IN6
- ☐ IN7

Figure 18: This figure shows how to set the ADC inputs.

2. Checking the ADC configuration to set the *Clock Prescaler* to *PCLK2 divided by 8* in the

Parameter Settings tab. The ADC *Resolution* will be set to 12 bits, it is desired to have the highest possible value since the higher the measure's resolution the more accurate the controller's input will be as a result of having more digital values available for storing information.

3. Since the STM32F411 board has only one ADC data output register it can only store readings into its memory from one channel at a time. To do this with more channels the most efficient method is to activate the *Scan Conversion Mode*, an embedded scanning protocol which consists on a continuous sequenced scanning through the selected channels and storing its readings directly into the board's RAM memory through a *DMA* (*Direct Memory Access* without needing to use the board's CPU. This reading mode was created by the board's designers so it is possible to read from multiple channels without needing to reset the ADC, which would be a lot slower and resource-consuming.

   To activate this conversion mode, the *Scan Conversion Mode*, *Continuous Conversion Mode* and *DMA Continuous Requests* options will be enabled at the previous tab. At the *ADC Conversion Mode* dropdown it will be possible to set the channel scanning sequence by setting the *Number of Conversion* to 2 (the number of channels that will be scanned) and then selecting Channels 0 and 1 for the 1 and 2 ranks respectively. The *Sampling Time* will be set at 480 cycles, the maximum one, as it is not necessary to scan samples at an extremely high rate.

   To set the DMA, *ADC1* will be added at the *DMA Settings* tab in *Circular* mode, which will make the DMA store the values of the channel array recursively. The DMA will also be set with a data width of half word since the ADC resolution is 12 bits.

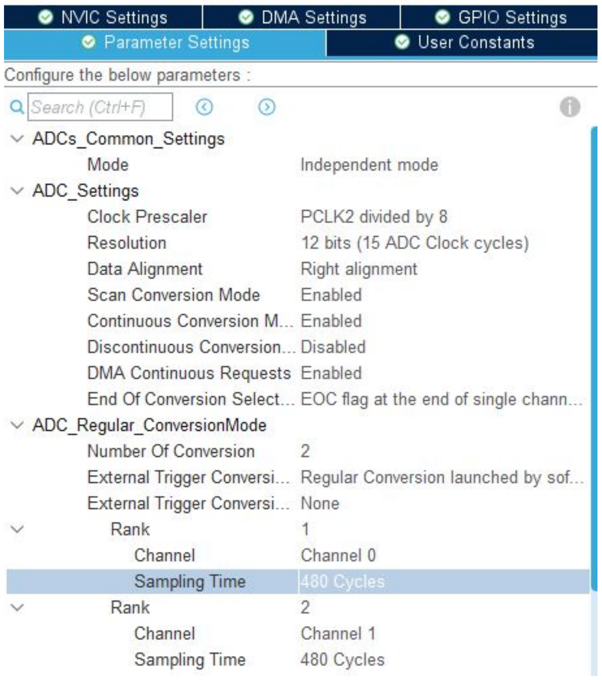
   The *ADC1 global interrupt* at the *NVIC settings* tab will also be activated. Activating this option will make the C code use a timer (which will be set at the next step next) to trigger the ADC interrupt.

4. The same process followed at the PWM timer initialization step 3 will now be repeated similarly to set the new timer, this time using the *TIM3* timer. This time the *Prescaler* will be set to 16, while the *Counter Period* is set to 50000. This results in a 20 Hz sample acquisition frequency, or a 0.05 second sampling period, which will be used later to define the system's discrete time mathematical representation.

   This sampling period has quite a reasonable value, as the ball's position measure will be update relatively quickly, but it will not make the reading unstable, which happens if an excessively low sampling period is introduced. This is due to the fact that, as the measures occur very frequently, so do the inaccurate measures, which gives a higher amount of misleading information to the controller and makes the system likely to get unstable.

   Instead of designating a role to some channel, it is now necessary to set the timer's mode to clock source. It will be set as *Internal Clock*, which means it will be act as the internal clock of the board. At the *NVIC Settings* tab of the timer's configuration the *TIM3 global interrupt* option will be selected to link this timer to the global interrupt which will be used to trigger the ADC interrupt.

After completing this step, the *PA0* and *PA1* pins should appear highlighted in green the same way the *PA5* pin was highlighted after setting the PWM timer.



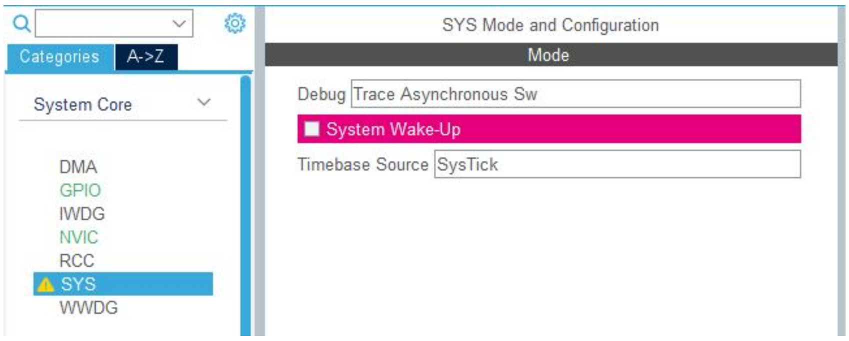
ADCs_Common_Settings	
Mode	Independent mode
ADC_Settings	
Clock Prescaler	PCLK2 divided by 8
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Enabled
Continuous Conversion M...	Enabled
Discontinuous Conversion...	Disabled
DMA Continuous Requests	Enabled
End Of Conversion Select...	EOC flag at the end of single chann...
ADC_Regular_ConversionMode	
Number Of Conversion	2
External Trigger Conversi...	Regular Conversion launched by sof...
External Trigger Conversi...	None
Rank	1
Channel	Channel 0
Sampling Time	480 Cycles
Rank	2
Channel	Channel 1
Sampling Time	480 Cycles

Figure 19: This figure shows the Parameter Settings tab configuration for the ADC inputs.

4.4 SWD plot information

The Serial Wire Debug (SWD) plot is a debug tool which is incorporated in most STM32 and similar boards. It allows the user to watch and control the behaviour of the board embedded code (which has been generated previously) while running from the STM32Cube IDE and therefore, to be able to measure variables and to check if the code is working as expected.

In order to configure the SWD plot the user must use the *SYS* (SYS Mode and Configutation) tab inside the *System Core* dropdown and select the *Trace Asynchronous Sw* debug option. This should make the program highligh in green the *PB3*, *PA14* and *PA13* pins.



Debug: Trace Asynchronous Sw
Timebase Source: SysTick

Figure 20: This figure shows how to set the SWD plot.

4.5 Base code generation

Once all the previous steps have been completed, select the yellow gear figure at the upper toolbar and accept the pop-up's request to generate the project's base code file which will be eventually implemented in the board. This code will be visible and ready to explore and edit, under the name of "main.c".

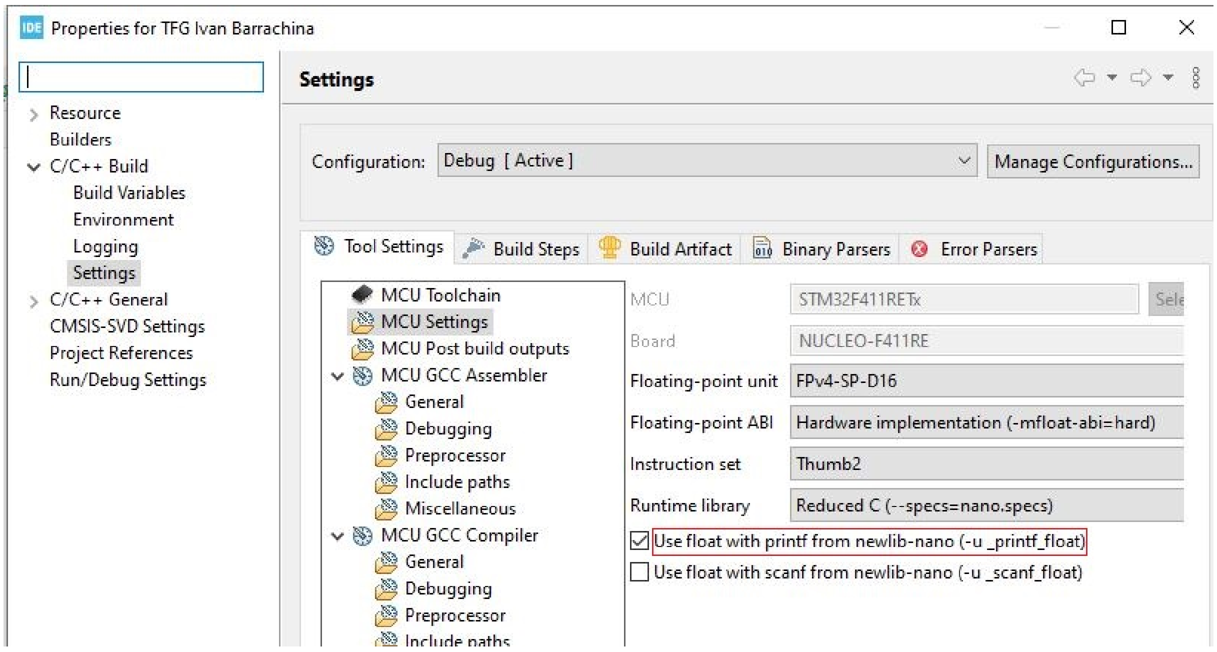
5 Non-controller coding

Once the base code has been generated, the next step is to include some elements which will make possible to finally implement a controller through it. It must be stated that the pieces of code depicted in this chapter will not include the whole final code, only the basic elements needed for the controller's function.

When coding in C, it is important to take into account that not all the space in the code is suitable for generating code, since this can only be done in certain coding dedicated spaces generated by the program. If the developer generates some piece of code outside these spaces, everything that has been written will disappear when the code is compiled. These spaces can be easily recognized since there are many of them all over any C code and they always follow a similar structure. There are two "barriers" which limit every space, which begin and end with "/* USER CODE", followed with "BEGIN" or "END" depending on which of the barriers is it and finally and argument that declares what is that space used for followed by "*/".

For example, the space started by "/* USER CODE BEGIN Includes */" is used to declare libraries which must be included to import necessary functions for the code. In this case, apart from the library inclusion there only will be new code written into the spaces 0 and 3 of the program, the first being used for variable and function definitions and the second to create code inside a loop. On the other hand, space 2 will be dedicated to peripheral initialization, so it must not contain any other piece of code generated for other reasons.

Before explaining the C code any further, it will be necessary to change some IDE settings so it is possible for the user to print float numbers from the code. To do so, it is necessary to select the *Project* tab at the upper toolbar and then select the *Properties* option. A new menu will open with a content bar on the left. Select the *C/C++ Build* option, followed by *Settings* and *MCU Settings* and then mark the following option:


Properties for TFG Ivan Barrachina

Settings

Configuration: Debug [Active]

MCU: STM32F411RETx
Board: NUCLEO-F411RE
Floating-point unit: FPv4-SP-D16
Floating-point ABI: Hardware implementation (-mfloat-abi=hard)
Instruction set: Thumb2
Runtime library: Reduced C (--specs=nano.specs)
[✓] Use float with printf from newlib-nano (-u _printf_float)
[ ] Use float with scanf from newlib-nano (-u _scanf_float)

Figure 21: This figure shows how to activate float printing for the C code.

Once the IDE is ready to code it is time to show the code that, as explained before, must be created before implementing the controller. This non-controller code is necessary because before implementing any controller it is crucial to check out that the micro-controller will be able to receive measures from the sensor and translate them from voltage to distance properly and also to move the servo the way it is has been planned. To make this process easier to understand, this explanation will be made step by step showing the different spaces that will be filled with code.

5.1 Library inclusion

The code for the libraries inclusion is showed next:

```
/* USER CODE BEGIN Includes */
#include "stdio.h" /* standard in out header, inclou el print f */
#include "math.h"
#include "time.h"
/* USER CODE END Includes */
```

In this part the user must include desired libraries to use throughout the code. This is also the space to incorporate MATLAB-generated functions' headers.

5.2 User code begin 0

As mentioned before, this space is dedicated to define variables and functions which are meant to be used in other sections of the code (especially in Section 3). The following code piece is an example of a function definition which will be implemented:

```
/* USER CODE BEGIN 0 */
int _write(int file, char *ptr, int len) /*funcio Per usar print f*/
        {
        int DataIdx;
        for (DataIdx = 0; DataIdx<len;DataIdx++)
        {
                ITM_SendChar(*ptr++);
        }
        return len;
        }

/* USER CODE END 0 */
```

This function showed before is used to print information from code variables into the console so it is easier to control the way the system and its components are working.

The necessary variables to make the code perform the desired tasks are not included here since it is not relevant information, and because it can be seen at the full code that has been attached in Annex B

5.3 User code begin 2

This part of the code is used to initialize the board's peripherals. The following piece of code starts the PWM and ADC trigger timers which have been defined automatically during the code generation using the parameters defined during Chapter 4:

```
/* USER CODE BEGIN 2 */

//Start the timers
HAL_ADC_Start_DMA(&hadc1, (uint32_t *)adcValArray, 4);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);

/* USER CODE END 2 */
```

In this code, the first function starts the continuous analog scanning from the *hadc1* object, and stores the measures into the $adcValArray$ 16-bit two-element array which has been defined in the 0 coding space. The last input of the function equals the total number of bytes stored into the array. Since each sensor has a measure resolution of half-word (as explained in Section 4.3), which means that each element of the array occupies 16 bits. The total number of bytes, then, equals 4.

The second function, meanwhile, initializes the PWM timer selecting the desired channel even though it does not modify the signal's duty cycle. This will be shown at the third code space.

5.4 User code begin 3

Section 3 is the final code section with which the user will interact in the predicted scope of this project. As stated at the beginning of this chapter, this section is used to create code that is meant to be executed recursively. This mean this section of the code is the core of the controller because it will include key information such as the sensor calibration, the measure translation from the analogical signal to a distance value in meters, the controller structure and parameters and the output signal creation which is sent to the servo-motor. These are only the necessary features that must be included here to make the system function properly, but the user can add any type of code that they might find useful or interesting, such as printings to control some variable.

Even though the final version of this code section is not showed here it can be found in Annex B. The code lines which allow the user to read an analog signal and to change the motor's duty cycle are attached next. These operations are the base of the user code and must be ready to use before calibrating the reading or implementing a controller. From these lines the sensor will be calibrated using a set of analog measures, which will allow to have a functioning system where the controller can be implemented.

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */
```

```
    /* USER CODE BEGIN 3 */

          //SENSOR 1: Moving average filter

          adcMA1=0;
          i1=0;
          for (i1=0; i1<len; i1++){
          adcMA1=adcMA1+adcValArray[0];
          }
          adcMA1=adcMA1/len;

          //SENSOR 2: Moving average filter

          adcMA2=0;
          i2=0;
          for (i2=0; i2<len; i2++){
          adcMA2=adcMA2+adcValArray[1];
          }
          adcMA2=adcMA2/len;

          printf("\n adcMA1 %f \n", adcMA1);
          printf("\n adcMA2 %f \n", adcMA2);

          htim2.Instance->CCR1=DutyCycle;

  }
  /* USER CODE END 3 */
}
```

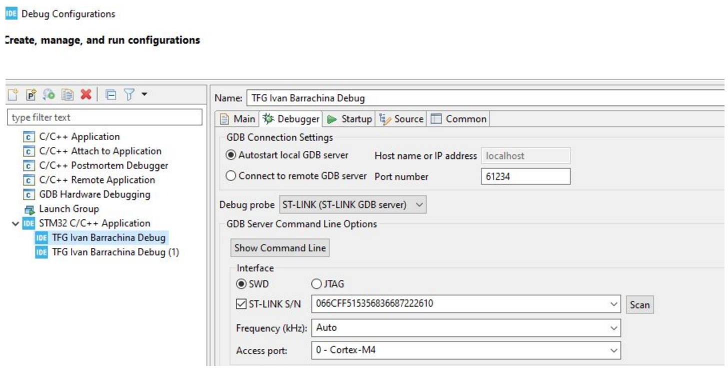
As stated before, this piece of code only includes the necessary instructions to execute a set of readings recursively and to send the motor a new duty cycle value. This duty cycle value is constant by the moment as now the only goal is to check that the motor moves when changing this parameter, but when the controller is operative the motor will receive the controller's output as its new duty cycle.

It can be seen that the reading code section is almost equal for both sensors: it begins with two auxiliary variables, one will store the analog value for each reading and the other will be used to create a moving average filter, which will reduce the signal's noise by calculating the final measure as the arithmetic average of a set (which length is defined by the variable len) of analog readings. The greater the set's length, the more accurate will be the reading, even though an oversized length may bring no significant benefit for the measure but will make the code run slowlier.

5.5 Code compilation and debugging

In order to check if the previously attached pieces of code work, this means, to debug it, it is necessary to compile the code. To do so, the spider-shaped *Debug* button from the top toolbar must be pressed after connecting the board to the computer through an USB cable. This will make the program scan the code to check it does not contain any error and then it will load it down to the microcontroller.

The first time the user presses this button the program will ask them to create a debugging configuration. To do so, the *STM32 Cortex-M C/C++ Application* option must be selected at the first emerging window. Now it is time to configure the debugger. The first step to do so is to make sure that *ST-LINK* (*ST-LINK GDB Server*) is selected as the Debug probe. At the *Interface* section it will be possible to use an specific ST-LINK by checking the option showed next and scanning to find the debugger of the connected board.



Debug Configurations

Create, manage, and run configurations

type filter text
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- Launch Group
- STM32 C/C++ Application
  - TFG Ivan Barrachina Debug
  - TFG Ivan Barrachina Debug (1)

Name: TFG Ivan Barrachina Debug

Main | Debugger | Startup | Source | Common

GDB Connection Settings
- [✓ Autostart local GDB server] Host name or IP address: localhost
- Connect to remote GDB server Port number: 61234

Debug probe: ST-LINK (ST-LINK GDB server)

GDB Server Command Line Options

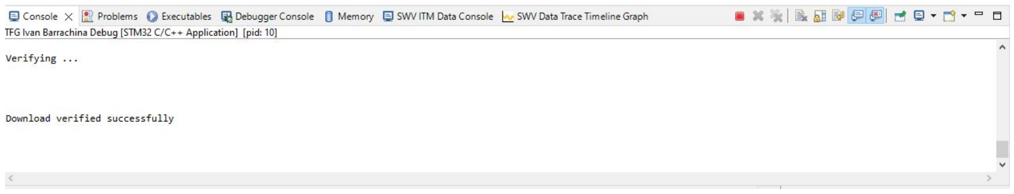
Show Command Line

Interface
- [✓ SWD] JTAG
- [✓ ST-LINK S/N]: 066CFF515356836687222610 Scan
- Frequency (kHz): Auto
- Access port: 0 - Cortex-M4

Figure 22: This figure shows the successful board debugger scanning.

The next step is to enable the *Serial Wire Viewer* that was configured in Section 4.4. To do so, it is necessary to check the *Enable* option below the *Interface* section and make sure the stated *Core Clock* frequency is equal to the board's clock frequency.

The debugger configuration is finished now and if the compilation and the loading are successful, the following menu should appear at the bottom of the screen:



TFG Ivan Barrachina Debug [STM32 C/C++ Application] [pid: 10]

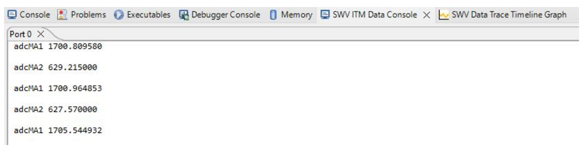
```
Verifying ...



Download verified successfully
```

Figure 23: The debugging window will now appear at the bottom of the page. The successful download message can also be seen at the menu's console.

The debugging menu has several tabs which contain useful information and options for the user. However, the only useful tab for the project's program debugging is the *SWM ITM Data Console* tab, whose job will be to print out all the variables that were included within the $printf$ commands at the end of the third user code section. If this tab does not appear at the toolbar by default, it must be selected at the *Window* tab at the upper toolbar, then check *Show View, SWV* and *SWM ITM Data Console*.

After clicking on the hammer-shaped *Configure trace* button the SWV setting menu will open, within which the 0 stimulus port must be selected. To make the SWV data console work the red circle shaped *Start Trace* button must be checked to start the console's trace.

Now the debugger is ready to use. If the *Resume* button at the top left part of the screen is pressed, the analog readings of both sensors (corrected by the moving average filter) will now appear recursively on the *SWM ITM Data Console*, as shown in the figure below:



```
adcMA1 1700.809580

adcMA2 629.215000

adcMA1 1700.964853

adcMA2 627.570000

adcMA1 1705.544932
```

Figure 24: An example of corrected analog value printings for the $75mm$ position

The motor will also tilt the beam to the new position after its input signal's duty cycle has been modified to the one value written in the code (unless it was already in that position). This means that the code works and that the sensor calibration can be finally started.

5.6 Sensor calibration

Once the code is developed enough to test the analogical input measuring, it is possible to finally calibrate the sensors. This process is crucial for the system's correct operation as the board's analogical reading assumes that the correspondence between the physical magnitude (the ball's relative position to the sensor) and the analogical value is lineal, which is false since, as shown in Section 3.1, the sensor's output voltage is not linearly related to the distance to the reflective object. It is easy to see that the voltage vs distance function provided in the datasheet is excessively difficult to implement into the controller's code. Thus, the calibration will consist of establishing linear approximations between these two magnitudes so it is easier to establish this correspondence.

It is also worth noting that these analogical values will be limited by the measuring resolution which, as stated previously, is a 12-bit resolution. The maximum possible analogical value which can be obtained, then, is $2^{12} - 1 = 4095$.

The calibration's first step consists on establishing a set of measure values for each sensor's readings at different ball positions in the beam. The set of values has been decided to consist of measures from the -150 mm to the 150 mm positions with a 25 millimeter step between each position. The measuring results are shown in the following table:

Table 1: This table shows the calibration measures for each position

Position [mm]	Sensor 1 Measure	Sensor 2 Measure
150	3100	700
125	2500	700
100	2050	760
75	1750	800
50	1550	900
25	1410	1000
0	1270	1100
-25	1150	1280
-50	1050	1450
-75	950	1700
-100	950	2000
-125	920	2480
-150	900	3220

It can be seen with the naked eye that the closer the ball is to the sensor the faster its measure grows. This could also be predicted by looking at the voltage vs distance graph in Chapter 3.1. The noticeably non-linearity of the measure makes it necessary to separate the measurement positions in groups in which the function's gradient is somewhat constant or approximately linear in order to create a set of linear correspondence equations between the two magnitudes. The gradient of the measuring also remains constant when the ball is on the furthest half of the beam, which will make it impossible to extract any useful information from the measure of one of the sensors most of the time. Therefore, each sensor will be in charge of its half of the beam while the other sensor's readings will be discarded.

There is another case, in which the ball is positioned on the central region of the beam, in which both sensor readings are valid. This case adds some difficulty to the calibration and forces the code to create a more complicated sensor choice in this particular range of analog values.

In order to create this adaptation on the equations to make them suit this problem it is necessary to firstly define what will be considered as the central area of the measuring. As it can be seen in Table 1, the analog values for the central position are 1270 for Sensor 1 and 1100 for Sensor 2. This means that when the analog reading approaches these values the actual measured distance will be uncertain, as both sensors will apply their linear equations to approximate the distance value.

This problem has been solved by implementing a measuring selection code which rejects the sensors' measures if they are below a certain threshold -1150 for Sensor 1 and 950 for Sensor 2-. The case in which both sensors' readings can be considered valid has been solved by coding that, if both readings are within a certain range, a mean from both readings is considered as the analog reading. If one of the readings is over this range and the other does not present a significantly higher value, the code will consider that the ball is nearer its sensor and thus will reject the other sensor's reading.

The rejection thresholds and the mean reading range have been established through experimentation in order to optimize the measuring. These values have been tested repeatedly by measuring the ball's position when it is idle in a certain critical spots -when both valid ranges overlapped-. These measures have been compared to the real position watching if there were sudden measure changes after slightly moving the ball. If this happened, it would mean that one of the readings -which have a certain variability- presented borderline values for that position, making it necessary to review the thresholds and ranges.

Once the calibration has been described, the equations will be finally stated. For both sensors, the measurement range will divided in three areas in which a linear or polynomial trend line is calculated. For Sensor 1, the first area is the one obtained from the 150, 125 and 100mm measures -named *EQ1*-, with the others spanning between 100 and 25mm -*EQ2*- and from 25 to -25 mm -*EQ3*-. The three equations are presented next:

$$r = 0.0473x + 4.39\,, \tag{1}$$

$$r = -441 + 0.479x - 0.000105x^2\,, \tag{2}$$

$$r = -332 + 0.326x - 0.0000518x^2\,, \tag{3}$$

where r is the approximated distance in millimeters and x the measured analog value. These linear regressions have R-square parameters of 0.993, 1 and 1, respectively, which shows that the approximation is almost perfect for the whole studied distance range.

The equations for Sensor 2, noted as $EQ4$, $EQ5$ and $EQ6$, are obtained from the 25 to -25mm positions, from -50 to -100mm and from the -100 to -150mm, respectively. They are presented next:

$$r = 354 - 0.447x + 0.000116x^2\,, \tag{4}$$

$$r = -0.0907x + 80.6\,, \tag{5}$$

$$r = -0.0404x - 21.4. \tag{6}$$

These linear regressions have R-square parameters of 0.994, 0.997 and 0.985, respectively.

It is worth pointing out that these approximations have been obtained from filtered measures. The raw measures presented a sensibly high variability, which did not get stabilized after applying the moving average filters. This is due to multiple factors like harmonic frequency electrical interference coming from the electric network, poor cable insulation among others. These problems were solved after establishing the signal electronic filter that was described in Section 3.1. Implementing these filters reduced the signal noise sensibly, making it possible to calibrate the sensor using reliable data and reducing the system's final functioning instability brought by the sensors. However, it is almost impossible to fully eliminate this instability in the readings, that is the reason why the sensors' acquisition period has been modified. Making the controller receive feedback with less frequency makes the system less sensible to flawed data, as the errors are introduced less often into the loop.

6 Ball and Beam mathematical modeling

In order to calculate any controller, it is necessary to generate a mathematical model from the system's dynamics and kinematics, as to do so it is key to know the system's temporal response parameters, specially its poles. This chapter encompasses the calculations performed to calculate the model's transfer function between the servo-motor's angle (represented in the model as θ) and the position of the ball in the beam (represented as r), both in continuous and discrete time, which is the tool that will be used to extract any necessary information to calculate the controllers which will control the motor's desired input signal to keep the ball at the center of the beam.

As explained at the project's introduction, the ball and beam system consists of a ball which can move along a straight beam in a one degree of freedom movement. The beam's inclination inputs another degree of freedom and will be forced into certain values through the servo-motor.

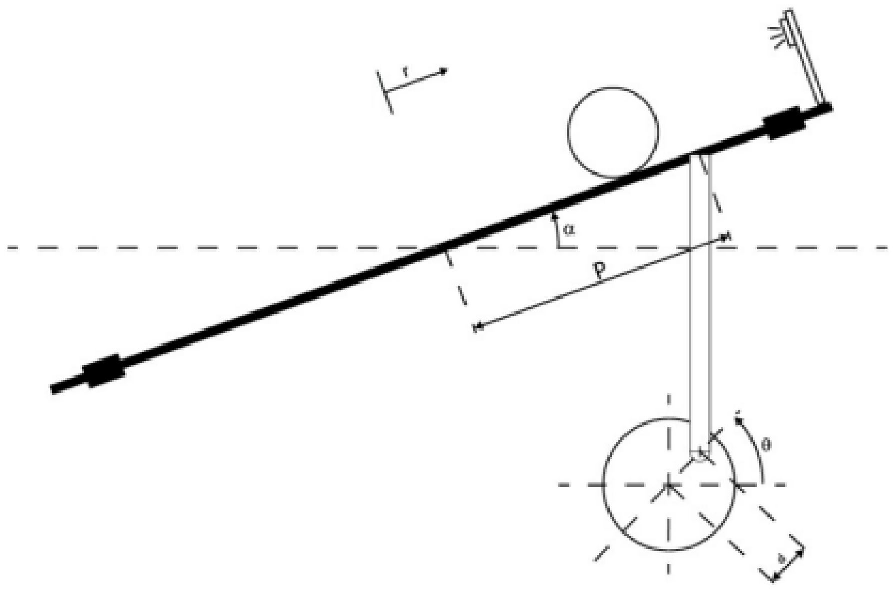


Figure 25: Physical diagram which shows the system's physical variables.

The first stage of the transfer function's calculation is to establish the system's Lagrangian equation of motion, which represents the ball's kinematics over the system's two degrees of freedom. The ball and beam is a classic automatic control problem and its Lagrangian equation is widely known and relatively simple to calculate. Therefore, the previous calculations will be considered trivial and the equation will be directly presented as [2]:

$$\left(\frac{Ib0}{R^2} + m\right)\ddot{r} + k\dot{r} - mr\dot{\alpha}^2 + mg\sin\alpha = 0\,, \tag{7}$$

where $Ib0$ represents the ball's moment of inertia, R its radius, m its mass, k the friction constant between the ball and the beam and g represents gravitational acceleration. The equation's variables are r and α, which represent the ball's relative position on the beam (with $r = 0$ representing the ball being located at the center of the beam) and the beam's angle to the horizontal reference axis. The dots on some of the variables represent the variables' derivatives, with one dot on the variable representing its first derivative and two dots the second derivative.

Since the beam angle α will not be comprised between relatively high values, the linear approximation $\alpha \approx 0$ will be considered valid. This will allow the angle's first derivative nullification and the sinus approximation to α following its Taylor's polynomial. The approximated Lagrangian equation is given by:

$$\left(\frac{Ib0}{R^2}+m\right)\ddot{r}+k\dot{r}+mg\alpha=0\,, \tag{8}$$

in which the moment inertia $Ib0$ is calculated ([5]) by:

$$Ib0=\frac{2}{5}m\frac{R^5-r^5}{R^3-r^3}\,, \tag{9}$$

in which r is referred to the internal radius of the ball, as it is hollow. It is necessary to remark that it is not the same r that is used for the ball's relative position. To avoid confusion, from now on this calculation will not be repeated in the memory and it will be assumed.

It is important to notice that the Lagrangian equation is not expressing any link between the ball's position, which is the one variable desired to control and the servo-motor's angle θ, the one that can be controlled. Therefore, the transfer function which can be obtained from the Lagrangian equation would not contribute with any significant information for the project's goal. To extract the desired relation, the beam's angle must be substituted by the motor's one using a mathematical expression.

According to [2], the equation which describes the relationship between the two angles can be approximated to:

$$\alpha=\frac{d}{P}\theta\,, \tag{10}$$

where d represents the model's articulation lever arm offset to the motor's edge and P the distance between the lever arm's articulation to the beam and the center of the beam.

Thus, substituting this expression into (2) the following expression is obtained:

$$(\frac{Ib0}{R^2}+m)\cdot\ddot{r}+k\dot{r}+\frac{mgd}{P}\theta=0 \tag{11}$$

Now there exists the desired relation between the two degrees of freedom and therefore the continuous time transfer function can be calculated. The transfer function will be obtained from (5)'s Laplace transform, which is showed next:

$$\left(\frac{Ib0}{R^2}+m\right)s^2R(s)+ksR(s)=-\frac{mgd}{P}\Theta(s) \tag{12}$$

Table 2: This table shows the constant's values

Constant	Value	Unit
$Ib0$	5.788e-06	$\text{kg} \cdot \text{m}^2$
R	0.01988	m
m	0.028	kg
d	0.06	m
P	0.15	m
k	0.05	-

The transfer function will have $R(s)$ as its output and $\Theta(s)$ as its input. Therefore, $R(s)$ will be isolated and then divided by $\Theta(s)$. By doing this operation, the continuous time transfer function will be finally obtained. Before stating the final result, the system's constants will be substituted in order to obtain the transfer function at its simplest form. The constants when studying the system with the common, white ping-pong ball are listed in the following table:

Note: The constants' units follow the international system and the gravitational acceleration has been approximated to the second decimal.

Proceeding to operate as stated before the transfer function is obtained. It is mathematically desirable to express it in its canonical form, as showed next:

$$G(s) = \frac{R(s)}{\Theta(s)} = \frac{2.5764}{s^2 - 0.1172s}. \tag{13}$$

Before calculating this transfer function is necessary to take into account that the gravitational constant g takes a negative value according to the physical reference axes. Using a positive value would result in a totally different system representation which would make the calculations impossible to make.

Once this function has been obtained, it is already possible to know how the system will react to any analogical signal input and to study its behaviour in continuous time. However, since the controller will be implemented into a micro-controller, which processes digital signals in discrete time, it will be necessary to obtain the discrete time equivalent of this transfer function so it is possible to study the whole system in this modeling framework.

In order to obtain a discrete time transfer function from a continuous time one it is necessary to perform a variable conversion known as the z-transform. This operation takes a discrete-time equation or function as an input, which in this case will be the sampled continuous-time transfer function, and converts it into a complex discrete time form whose variable is represented as z.

Before going any further, it is convenient to take a look to the global system's block diagram to see what the system's whole picture will look like:

Even though the controller can be calculated in the continuous time domain, it will be more convenient to calculate it directly in its z-domain form as the controller works in the discrete-time domain. The system will be represented as an unitary closed-loop system because the sensors will give the controller a distance feedback which will be compared to the desired distance.

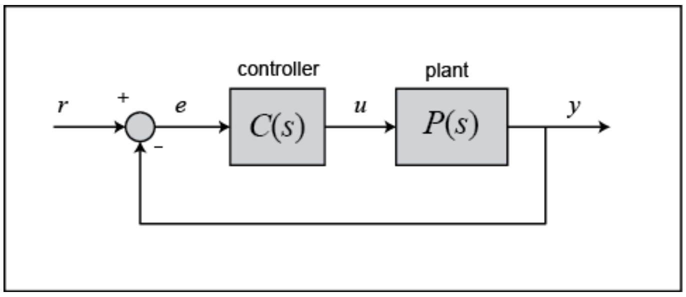


Figure 26: This figure shows a simplified representation of the system's block diagram

The difference between the measured and the desired distance values is named error -noted as e, $E(s)$ or $E(z)$- and will be the controller's real input.

It can be seen that the controller and the plant work in different domains. In theory, this would be solved by placing a sampler -also known as analog to digital converter- and a holder -a digital to analog converter, in this case a zero order holder will be considered- at the controller's output, a sampler at its input and another sampler at the plant's output, which would cause the whole system's equations to be expressed in sampled discrete time. This would also make the calculations simpler, as will be seen soon. But in practice, there is no need to place anything as the model components already perform these tasks. The sensors act as the system's feedback but also take samples -measures- from the system's output, which is the ball's position. This can be considered as the model's equivalent for the the micro-controller's input and the plant's output samplers, as the micro-controller will not have to sample the signal twice. It will also calculate new discrete duty cycle values periodically -the output's sampler-, an information which will be transferred to the motor which functions in continuous time -the holder-.

As it is have been explained, even though the system is considered to be represented in sampled discrete time, the micro-controller's calculations will be made in the z-domain, so it is necessary to calculate the controller in this domain no matter how the rest of the system looks like.

To convert a Laplace transfer function into a z-domain one, it is necessary to state a sampling period which determines the time elapsed between two samples. However, as it is established in a simplified way by Shannon's sampling theorem [3], a signal's sampling period must be equal or lower than the signal's faster fundamental oscillation component in order to capture all the signal's information. If the sampling period does not fulfill this requisite the sampling is not valid as some information is lost.

Since the sampling period T_s must be faster -which means it must have lower values- than the system's oscillation, a period of 0.05 seconds -or a sampling frequency of 20 Hz- will be chosen as it will be easy for the micro-controller to work at this speed and it will be pretty visual for the user to see how the system works. This period is considered valid as the system is not intended -and will not- oscillate at any similar period.

Thus, to calculate the discrete time TF the following expression will be used [4]:

$$G(z) = (\frac{z-1}{z}) \cdot Z(\frac{G(s)}{s}) \, , \tag{14}$$

which calculates the z-domain equivalent of a zero order holder in series with a sampled continuous time plant. It is calculated this way since the micro-controller output acts as a zero order holder. It is worth noting that the Z in this equation represents the direct z-transform of $\frac{G(s)}{s}$.

Applying this formula on (13) results in the following expression:

$$G(z) = \frac{0.0032z + 0.0032}{z^2 - 2.0059z + 1.0059}\,, \tag{15}$$

which will be used as the plant's equation to calculate the controllers at the next chapter.

7 Controller calculation and implementation

Once the physical device has been mathematically modeled, it is possible to study the system and create a controller which can enforce the desired state on the model. As it has already been stated, the controller structure that will be used to control the model and to prepare a laboratory practical session is the PID controller. This controller is one of the most elemental, if not the most, controllers in automatic control, due to its relative simplicity and effectiveness.

Even though the PID controller will the only control structure that will be implemented and tested, a series of specification and model conditions changes will be performed in order to evaluate how the same control structure adapts to be capable of stabilizing the system. This process will be carried on though different iterations. In the first iterations the desired system characteristics will be altered -mainly its settling time-, while in others the model conditions will be altered by changing the regular ping-pong for a heavier one, which will alter three of the system parameters presented at Table 2.

7.1 First iteration

7.1.1 PID Calculation

The PID controller is a closed loop control structure which corrects the plant's input signal by applying and summing three actions -known as Proportional, Integral and Derivative actions- to the system's feedback or error. This error is calculated as the difference between a certain measured plant variable value and the desired value for that variable, also known as the set-point. These three actions are applied to obtain the controller's output for a certain time value -noted as $u(t)$- from the error signal following the expression presented next [3]:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)\,dt + K_d \frac{de(t)}{dt}\,, \tag{16}$$

in which each K constant represents the coefficient for one of the three terms. The error and the controller's output are calculated periodically with a period equal to an integer number n times the sampling period T_s.

This equation is equivalent to Figure 27's block diagram:

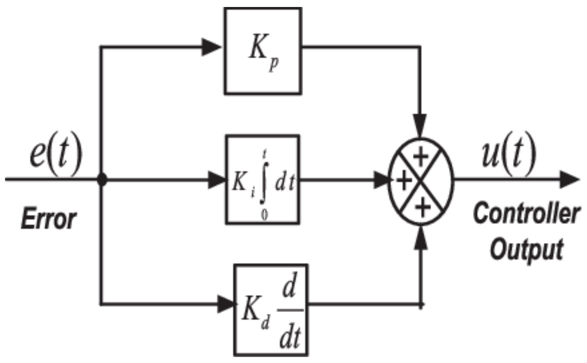


Figure 27: This figure shows the PID controller block diagram continuous time representation.

To allow the controller to compare the ball's relative distance value to the setpoint it is necessary to implement it in a closed loop form, whose structure can be seen represented in its block diagram form in the next figure:

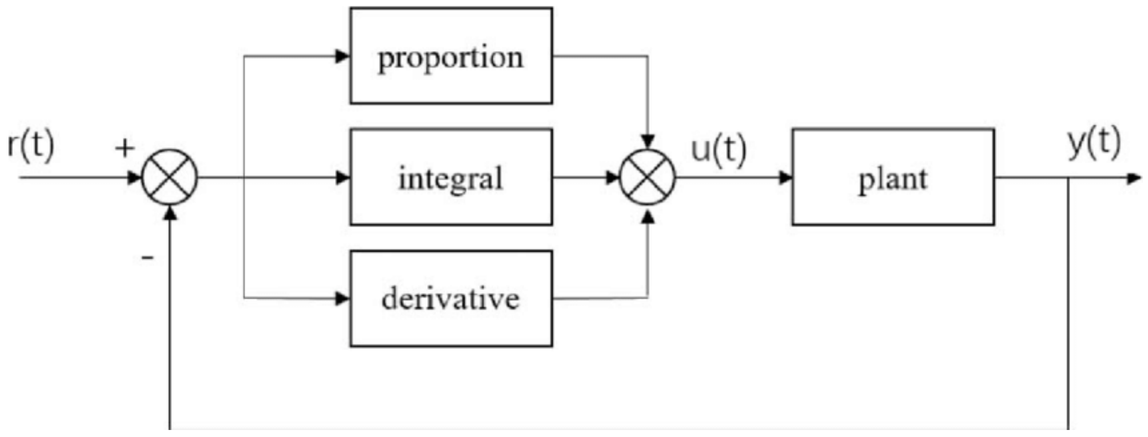


Figure 28: This figure shows the system's closed loop block diagram representation.

Each of the three controller actions have a role inside the system:

1. **The Proportional action (P),** whose role is to be the main reaction to the error signal, since it multiplies it to a constant, so the higher the error signal's value is, the more intense the controller's action will be.

2. **The Derivative action (D),** which, as it applies its coefficient to the error signal's derivative, brings the controller a reaction capability to changes in the error. In this case, the derivative action is the one in charge of making the ball stop if the proportional action to the error has caused the ball to move too fast. Even though it will help stop the ball when the proportional action moves the ball away from big error areas -which means, from points far from the setpoint-, it lacks precision, and it is difficult to make the final point of the ball's trajectory match the setpoint.

3. In order to avoid having the ball stop at points close the setpoint where the other two actions are not very effective, **the Integrative action (I)** integrates the error signal in order to make the system change if small error values are detected over time. This will make the ball's positioning more accurate at the cost of making the system "nervous", which means that the final position will oscillate if minimum error values are measured. If the integrative coefficient is too high, the system's sensibility to these error accumulations will be higher, and it could even destabilize when the positioning has already been completed.

This subsection's main objective is to calculate what values must be assigned to the PID controller's parameters in order to stabilize the ball's position value to the setpoint in a fast but stable way. There are many ways to do so, but in this project the method that will be followed is the desired pole assignation method, which consists on calculating the system's closed loop z-transform monic characteristic equation without assigning any value to the controller's parameters, and then equalizing this polynomial to the desired z-domain equation which is desired to obtain in order to calculate the three coefficients.

There are many alternative methods and algorithms to perform this parameter selection, including MATLAB and Simulink scripts or models which can easily tune PID controllers from the model of the plant to control, but the process is going to be done this way in order to put into practice what was learned at ETSEIB's Automatic Control subject. Even if both polynomials have been calculated by hand -a process which is going to be described in a short time-, the resulting equation system calculations, have been solved using MATLAB with a script that is going to be included in Annex B.

As it was shown at the end of Chapter 6, the plant's z-domain transfer function was the following:

$$G(z) = \frac{0.0032z + 0.0032}{z^2 - 2.0059z + 1.0059}\,, \tag{17}$$

which is represented alternatively as showed next in order to make the system's closed loop polynomial and the MATLAB script development easier to handle:

$$G(z) = \frac{\alpha z + \alpha}{z^2 - \beta z + \gamma}\,, \tag{18}$$

The PID controller's z-domain transfer function is stated next [4]:

$$C_{PID}(z) = K_p + \frac{K_i T_s}{2}\frac{z+1}{z-1} + \frac{K_d}{T_s}\frac{z-1}{z}\,, \tag{19}$$

in which the integrative and derivative actions have been discretized in their trapezoidal and backward Euler approximations, respectively. The trapezoidal integration is helpful in order to avoid sharp changes in the integral action when the error value changes suddenly, which is good to make the controller more resistant to perturbations such as someone moving the ball with their fingers. The backward Euler derivative approximation is a causal, straight-forward form commonly used in PID controllers as it is the simplest one in terms of implementation, which does not make it, on the other hand, be a worse choice than other more complicated strategies.

This transfer function has been restructured in order to make the polynomials easier to calculate and express. It is presented in the following form:

$$C_{PID}(z) = \frac{b_2 z^2 + b_1 z + b_0}{z(z-1)}\,, \tag{20}$$

in which the b_2, b_1 and b_0 variables' correspondence to the controller's parameters is described by the next set of equations:

$$b_2 = K_p + \frac{K_i T_s}{2} + \frac{K_d}{T_s}\,, \tag{21}$$

$$b_1 = -K_p + \frac{K_i T_s}{2} - \frac{2K_d}{T_s}\,, \tag{22}$$

$$b_0 = \frac{K_d}{T_s}. \tag{23}$$

The closed loop polynomial is obtained from the denominator of the closed loop transfer function, which is calculated following the next expression:

$$T(z) = \frac{C_{PID}(z)G(z)}{1 + C_{PID}(z)G(z)}, \tag{24}$$

whose denominator is expressed in its monic form in function of the controller parameters as showed in the next Equation, which corresponds with the system's closed loop characteristic equation. This denominator is stated next:

$$D(z) = z^4 + (b_2\alpha - \beta - 1)z^3 + (\gamma + b_2\alpha + b_1\alpha + \beta)z^2 + (b_1\alpha + b_0\alpha - \gamma)z + b_0\alpha = 0. \tag{25}$$

This characteristic equation is meant to be equaled to the desired denominator of the closed loop transfer function, which is obtained directly by multiplying four -as many as equations the system has- poles in their $(z - p_i)$ form, where $i = 1...4$.

The four selected poles for this first iteration and the reasons of why they have been selected are presented next:

1. A pole located at $z = 1$ in order to make the system be a type 1 system, which will make its steady state error to a step input converge to 0. This is key since the motor input signal is step-shaped. Therefore, having some error to this type of input signal would make the ball positioning be imprecise.

2. A pole located at $z = 0.8752$ in order to set the system's settling time at 1.5 seconds. The settling time is defined as the time taken for a system's response to reach and stay within a 2% range of its final value. This means, to make the ball positioning be certainly fast -even though in practice this time will be higher-.

   Once the desired settling time has been established it is possible to calculate the pole. Since the settling time t_s -which is important to distinguish from the sampling period T_s is known, the Laplace transform pole's real part can be extracted by following this expression [4]:

$$t_s = -\frac{4}{\sigma}. \tag{26}$$

   The pole's imaginary part is set to 0 as it is not desired to work with two conjugated complex poles. Thus, it is obtained that $\sigma = -2.6667$. The resulting z-domain real pole is the already stated $z = 0.8752$ and it is calculated by the z-transform of the Laplace pole by following:

$$z = e^{\sigma T_s}. \tag{27}$$

3. A fast pole located at $z = 0.1$. Its adjective refers to the fact that it will not change the system's properties and it will not determine the system's response speed as there are slower poles in the characteristic equation. The slower poles are the ones with the biggest module, and vice versa.

4. An undetermined pole noted as $z = a$. This undefined pole will be the equation system's fourth variable and must have a module strictly lower than 1 to avoid making the system unstable. It can't have a module equal to 1 either since there is already one (pole 1).

From these poles' product it is possible now to represent the desired characteristic equation of the system in function of the undefined pole a as shown next:

$$D^d(z) = z^4 + (-1.9752 - a)z^3 + (1.9752a + 1.0627)z^2 + (-1.0627a - 0.0875)z + 0.0875a = 0 \tag{28}$$

Now the PID controller's parameters K_p, K_d and K_i and the undefined pole a can be determined by solving the equation system obtained from equaling each term of both polynomials. The necessary calculations to achieve this have been done with computational help through a simple MATLAB code that is attached in Annex B. This system is expressed in its matrix form as shown next:

$$\begin{bmatrix} \alpha & 0 & 0 & 1 \\ \alpha & \alpha & 0 & -1.9752 \\ 0 & \alpha & \alpha & 1.0627 \\ 0 & 0 & \alpha & -0.0857 \end{bmatrix} \times \begin{bmatrix} b_2 \\ b_1 \\ b_0 \\ a \end{bmatrix} = \begin{bmatrix} -0.9752 + \beta \\ 1.0627 - \beta - \gamma \\ -0.0875 + \gamma \\ 0 \end{bmatrix}, \tag{29}$$

which has the following result:

$$\begin{bmatrix} b_2 \\ b_1 \\ b_0 \\ a \end{bmatrix} = \begin{bmatrix} 26.8042 \\ -52.6420 \\ 25.8378 \\ 0.9449 \end{bmatrix}. \tag{30}$$

It is clear now that the $z = a$ pole, which equals 0.9449, satisfies $|z| < 1$. This means that the pole will not destabilize the system and it is valid to include in the controller. By solving the system composed by (21), (22) and (23) the controller parameters will be found. This system is shown next:

$$\begin{bmatrix} 1 & \frac{T_s}{2} & \frac{1}{T_s} \\ -1 & 0 & \frac{-2}{T_s} \\ 0 & 0 & \frac{1}{T_s} \end{bmatrix} \times \begin{bmatrix} K_p \\ K_i \\ K_d \end{bmatrix} = \begin{bmatrix} b_2 \\ b_1 \\ b_0 \end{bmatrix}. \tag{31}$$

Now, the PID controller's parameters are known and it is possible to implement the obtained PID control structure into the micro-controller. The obtained parameters are:

$$\begin{bmatrix} K_p \\ K_i \\ K_d \end{bmatrix} = \begin{bmatrix} 0.9644 \\ 0 \\ 1.2919 \end{bmatrix}. \tag{32}$$

It is noticeable that the integral constant is stated as negligible -since it was of order -13-. However, at the controller implementation it will be shown that the integral action must not be null in order to correctly drive the ball to the setpoint position.

In order to evaluate the controller's theoretical behaviour without entering the implementation process, a Simulink model -which was introduced in Section 3.3- of the closed-loop system which includes this PID controller has been used.

The Simulink model's block structure is an emulation of the system's closed loop. It can be seen at the Figure below:

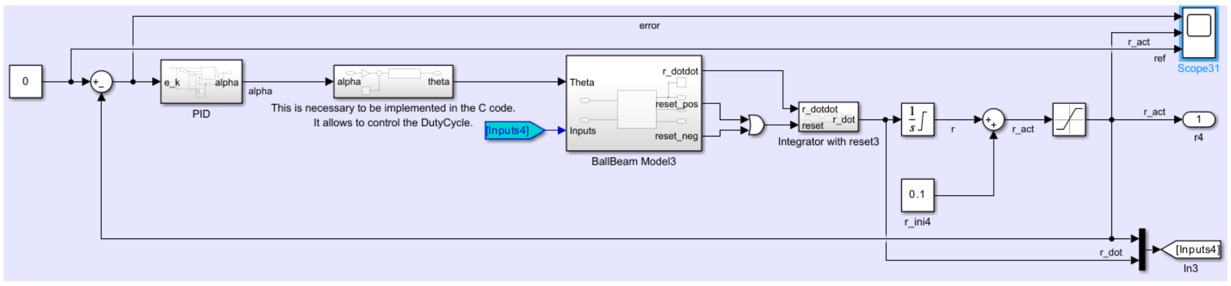


Figure 29: This Figure shows the Simulink model's block diagram.

The model provides a simulation of the ball's trajectory over a specific simulation time from a starting point that can be set at will by modifying its value inside the r_ini4 block. It is also necessary to introduce the system's -both the plant's and the controller's- parameters in the "BallBeam_Model.m" and run it so the simulation can use this information. The simulation results can be seen in a distance to time graph -and an error to time one- at the $Scope31$ block.

A 30 millisecond simulation taking $r = 0.1m$ as the initial position has been carried out. A $K_i = 0.1$ integrative coefficient has been introduced for computational reasons, even though when the controller is implemented it will be seen that it is actually necessary for the correct functioning of the system. The following simulation results have been obtained:

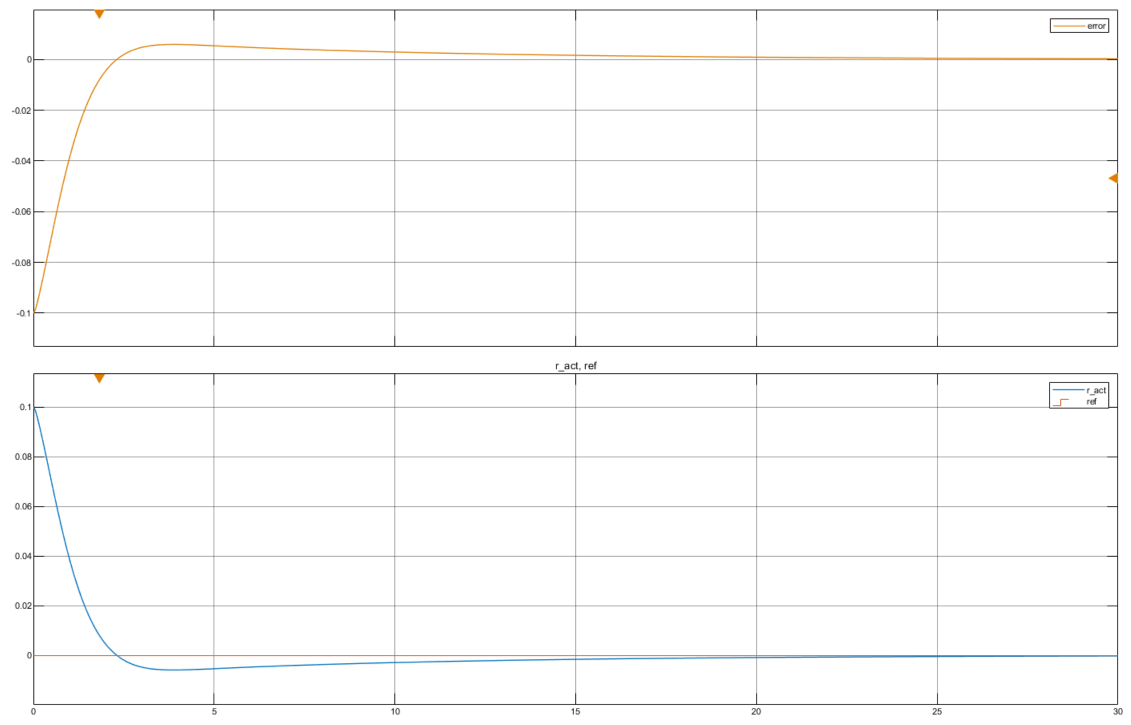


Figure 30: The ball's simulated trajectory for an initial position $r_{ini} = 0.1m$.

It can be seen in the graph that the ball's position will reach the setpoint in around 2 seconds and remain in there stably, moving slowly to the setpoint, where it will arrive after approximately 20 seconds. The result is the desired, as the stationary error tends to zero and the settling time is approximately the desired. However, when it is implemented later it will be possible to tell if its real behaviour matches the theory.

7.1.2 First iteration implementation

Once the PID controller parameters have been calculated for all controllers, the control structure must be implemented into the C code. The first PID to be tested will be the first iteration, whose coefficients are stated in (32).

The PID controller will be coded in the user code section 3 which was described at Chapter 5.4. After having the analog measure from the sensors translated to a distance value in meters, the code will process that value to convert it into a duty cycle value and will send the corresponding pulse to the motor.

The piece of code that performs this task is showed next:

```
dist=dist/1000;

//PID CONTROLLER
error=dist_ref-dist;
error_ant=error_ant;
I_ant=I_ant;
Ts=Ts;
```

```
/////////////////////////WRITE YOUR CODE HERE//////////////////////
P=kp*error;//P component of the output
D=kd*(error-error_ant)/Ts;//D component of the output
I=(ki*Ts/2)*(error+error_ant)+I_ant;//I component
///////////////////////////////////////////////////////////////////
if (I>100) I=100;
else if (I<-100) I=-100;

DutyCycle=530-(P+I+D)*244;

// Safety sat
if (DutyCycle>850) DutyCycle=850;
else if (DutyCycle<250) DutyCycle=250;
//Motor movement
htim2.Instance->CCR1=DutyCycle;
I_ant=I;
error_ant=error;
HAL_Delay(50);
```

This piece of code uses the sampled temporal form of the three controller actions to calculate the error. These forms are obtained by multiplying each signal (the error and each of the actions) by z^{-n}, in which n is the highest index of the z variables in the action equation. It is key to know that multiplying a signal by a negative index n variable z is the equivalent of having the value of the sampled signal delayed n samples. An example is showed next:

$$\frac{K_i T_s}{2} \frac{1+z^{-1}}{1-z^{-1}} E(z) = I(z)\,, \tag{33}$$

which results in the sampled form of the integrative action:

$$I(n) = \frac{K_i T_s}{2}[E(n) + E(n-1)] + I(n-1). \tag{34}$$

The proportional and derivative actions are also stated in this form:

$$P(n) = K_p E(n)\,, \tag{35}$$

$$D(n) = \frac{K_d}{T_s}[E(n) - E(n-1)]. \tag{36}$$

The error at the present iteration is stored at the *error* variable, while the values of the previous iteration error and integrative action are stored at the *error_ant* and *I_ant* ones, respectively.

The integrative part can grow without control under certain circumstances -e.g. if there is an obstacle that prevents the ball from moving to the setpoint or if it does not stand exactly at that point for any other reason-, which can create calculation problems or destabilize the system. To avoid these problems, an anti-windup control mechanism has been coded to limit the values that the I action can take.

The value of the sum of the P, I and D control signals corresponds to a new value of desired angle for the motor's edge in radians at the present iteration, but to move the motor it is necessary to perform a new conversion from the angle value to a duty cycle value which can be sent to the servo-motor. This conversion is performed assuming that the correspondence between these two variables is linear. By testing how some duty cycle values changed the motor's angle it has been seen that the 5.3% duty cycle (or 530 in the code) corresponds to a 0 radian position -the horizontal position for both the motor and the beam angle-. The maximum real duty cycle value -8.2% or 820- corresponds to a $\frac{\pi}{3}$ or 60º position. By establishing the linear equation using these two measures the following expression has been obtained:

$$DC(n) - 530 = 250(P(n) + I(n) + D(n))\,, \tag{37}$$

which equals to the expression of the duty cycle calculation shown at the code.

As it was mentioned in Section 3.1, the real limits of the duty cycle values are not the theoretical 0% and 10% ones, as values outside the 2.5%-8.5% range make the motor get saturated. To protect the motor from damage if too extreme duty cycle values are calculated, another anti-windup mechanism has been introduced to limit the output signal's duty cycle that is sent to it.

It is also worth noting that the code has been structured in a way it is easier to understand by someone with less C language knowledge, at the cost of making the code less optimal, but that has not been considered problematic for the project's purposes.

Implementing the other PID controllers that were calculated at the previous subsection is as simple as changing the K_p, K_i and K_d code variables which were defined at the user code section 0, which, as was stated in Chapter 5, is used to define user variables and auxiliary functions, to each iteration's results.

The controller has been implemented into the board and tested in Debugging Mode following the process described in Chapter 5. The ball has been placed at the $r = 0.15$ position before the test. After observing its behaviour the following conclusions have been obtained:

1. The model has been static the whole test. This is due to the absence of an integrative action that drives the ball to the setpoint even if the system is already stabilized at any other point. With a high certainty it is also due to that the model is not similar at all to the real system and the proportional and derivative actions are not powerful enough to move the ball to the setpoint. The controller needs to be tuned to be adjusted to the reality of the model.

   To test the integrative hypothesis a $K_i = 0.1$ coefficient has been added to the controller. The integrative action is usually really sensible and effective, so the coefficient must not be tested at very high values.

2. After adding this integrative coefficient the system has not been that idle, but the result has been equally unfruitful. The motor eventually drove the ball out of the end of the

beam, but it just rolled to the other end and stayed there.

This behaviour shows that the proportional action must be higher, as it will bring the system a sharper response to a high error signal. The proportional coefficient has now been assigned to a $K_p = 1.5$ value.

3. It is has now been seen that the system is now more sensible to the error, but not fast enough. The ball tended to roll out the end of the beam a lightly faster, but even if the derivative action tried to make it stop at the setpoint it was not powerful enough to do so in an acceptable amount of time.

   The system needs to be more sensible to the error but also to the error fluctuations so it can stop the ball when it moves too much. The proportional action will now be updated to $K_p = 2$ and so will be the derivative one.

4. The last changes to the controller have improved its performance significantly. The ball has moved instantly when the test has begun, and the derivative action has now stopped the ball with a surprising speed and stability. However, the setpoint of the ball has been around 25 millimeters away from the desired setpoint. This circumstance is not critical but it can be easily improved by increasing slightly the integrative action. A $K_i = 0.2$ coefficient has been added.

5. The system has responded as expected and it is has become more sensible to small errors, even too much sensible, which has made the ball's final position unstable. K_i has been lowered to 0.15.

6. This last version has proved to be precise and relatively stable. Naturally, the behaviour is not perfect, but it is considered acceptable taking into account the limitations of the model and how difficult it has been to find a good combination starting from the theoretical controller.

Finally, it is has been seen that the controller does not behave as expected in the real world, and that it is necessary to tune every computer to adjust to the model's limitations. This tuning has not been negligible at all, since it is has been necessary to double the proportional and derivative coefficients and introduce not a small integrative one to make it control the system as desired. However, the calculated controller is not considered wrong or invalid whatsoever, as the simulation back the calculations.

7.2 PID controller iterations

Once the PID calculation process has been described through the first iteration, it will be repeated for new iterations along this chapter.

These iterations have been selected in order to achieve two main goals:

1. First, to test how the PID controller can show adaptability to changes in the specification process, and to test if the real functioning controller obtained through experimentation is sensibly different to the first iteration's one. This would indicate that the model is limiting the controller's specification possibilities, limiting them in a minimum quality limit.

2. The second goal is to test if changing the model's parameters by switching the white ball

for the yellow, heavier ball has a significant impact on the controller's parameters and behaviour. In particular, it is desired to check if the yellow ball controller -for the first iteration- can control the white ball's position and vice-versa. This will show the controller's robustness in both directions -and by directions it is meant to say that if calculating a controller for a heavier ball helps have a better control on lighter balls and the opposite-.

7.2.1 Specification iterations

The specification iterations consist on testing changes on the settling time and the free small pole in order to study their effects on the closed-loop system. These changes cause a variation on the controller's desired characteristic equation, which will change the controller parameters'. If the resulting controller appears to be valid, it will be tested and analyzed the same way as the one in the first iteration.

These iterations are the following:

1. Lowering the settling time to $ts \approx 1s$ and maintaining the small pole at $z = 1$. These specifications result in the following characteristic equation:

   $$D^d(z) = z^4 + (-1.9187 - a)z^3 + (1.9187a + 1.006)z^2 + (-1.006a - 0.08187)z + 0.08187a = 0\,, \tag{38}$$

   in which the pole associated to the settling time is $z = 0.8187$.

   After solving the system it is obtained that $a = 1.002$. This controller can not considered as valid, since the controller would make the closed-loop system be unstable.

2. Keeping the settling time at 1 second and lowering the small pole to $z = 0.05$. The characteristic equation now becomes the following:

   $$D^d(z) = z^4 + (-1.8687 - a)z^3 + (1.8687a + 0.9096)z^2 + (-0.9096a - 0.0409)z + 0.0409a = 0 \tag{39}$$

   Now the system's results are that $a = 1.1009$. This controller would not be valid either. The settling time is already low, and some tests with even lower settling times show that the free pole grows even more. This is an indicator that lowering the settling time too much is not realistic and it makes the system unstable. It has been decided, then, that lowering the settling time more than 1.5 seconds is not worth the time and effort.

3. Now the settling time will be increased to $ts \approx 10s$. The small free pole will be maintained at $z = 0.1$. These specifications' associated characteristic equation is the following:

   $$D^d(z) = z^4 + (-2.0802 - a)z^3 + (2.0802a + 1.1782)z^2 + (-1.782a - 0.09802)z + 0.09802a = 0\,, \tag{40}$$

whose associated free pole is located at $z = 0.8418$, which means that the closed-loop system will be stable. The associated K_p, K_i and K_d of 0.4379, -0.0099 and 1.2893, respectively.

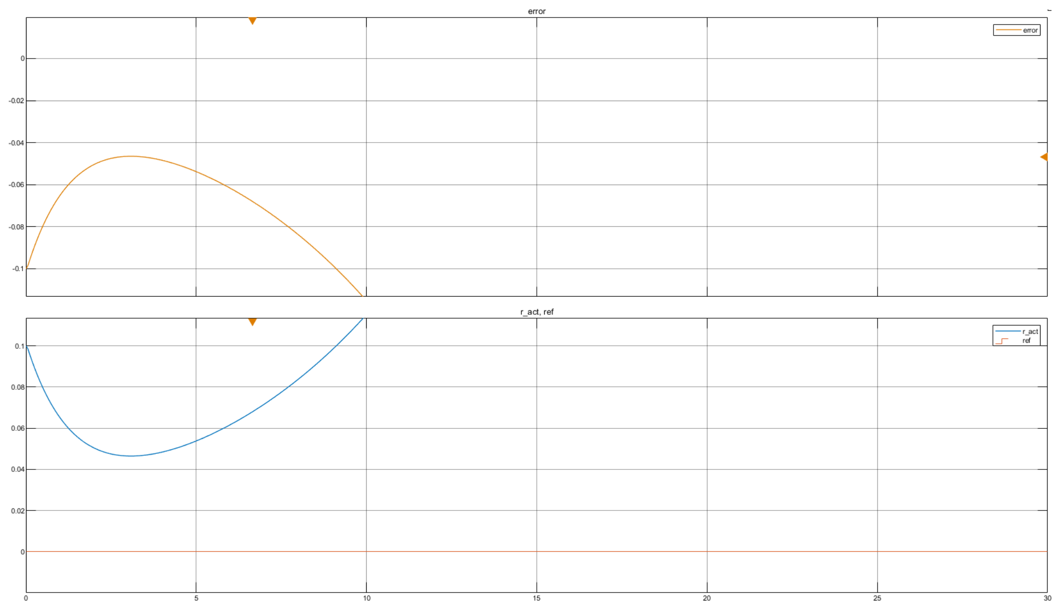


Figure 31: The ball's simulated trajectory for an initial position $r_{ini} = 0.1m$ and the third iteration's specifications.

It can be clearly seen that the system is not valid, since it appears to be unstable. Since the integrative coefficient is negative, which is not really logical, a $K_i = 0.15$ coefficient has been tested. The simulation shows the following graphs:

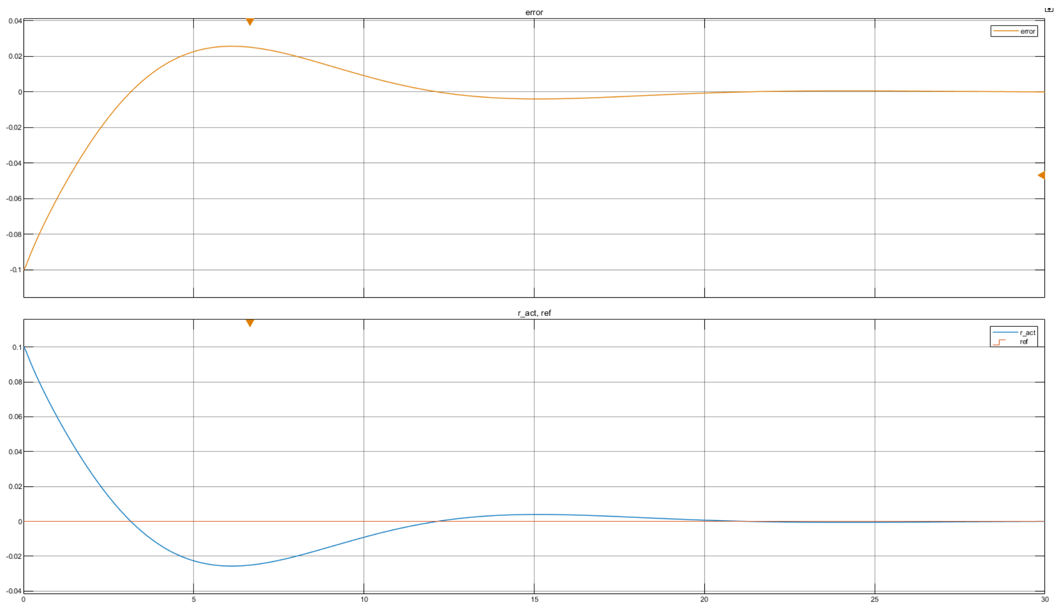


Figure 32: The ball's simulated trajectory after modifying the integrative coefficient.

The simulation shows that the K_i modification has helped making the system stable. It can also be seen that, even though the system presents wider oscillations, the ball reaches the setpoint approximately after 10 seconds, which satisfies the closed-loop specifications that were imposed during this iteration.

These controller parameter values are even lower than the ones at the first iteration, which means that the working values and the system's behaviour are the same as that iteration's. The same results will be obtained with similar small poles, which do not offer many variability to the system's coefficients. After implementing the second simulation's controller, the same problems from the first one have been encountered.

Since analyzing higher settling times configurations would not be of interest, as the system would be too slow, no more settling times will be studied. The real settling time will be considered somewhat constant and equal to the first iteration's one.

4. Setting the small pole to $z = 0.05$ again using the original $ts \approx 1.5s$. The following characteristic equation has been obtained:

$$D^d(z) = z^4 + (-1.9252 - a)z^3 + (1.9252a + 0.969)z^2 + (-0.969a - 0.0438)z + 0.0438a = 0\,, \tag{41}$$

which results in a free pole in $a = 1.0375$. The system, then, is not stable.

The obtained conclusion is that, even if within a reasonable settling time range, the real system will force the closed-loop characteristics, no matter how the controller's specifications are changed. Since the plant's application is purely educational, this is not a serious circumstance, but if it was destined to be implemented in the industry this would be totally unacceptable. The controller experimentation would be more fruitful if the model was refined to be more accurate to theory, but this will remain out of the project's scope.

7.2.2 Model parameter changes

These iterations correspond to the model variation in which the yellow ball is introduced in the system. The system's parameters that change are, then, the ball's mass, radius and inertia moment. These new values are presented in the following Table:

Table 3: The model variables' values that change when the balls are switched.

Constant	Value	Unit
$Ib0$	1.453e-05	$kg \cdot m^2$
R	0.0225	m
m	0.046	kg

These new variables change the system's transfer function, which was presented during Chapter 6. The system's new Laplace and z-transform transfer functions are the following:

$$G(s) = \frac{R(s)}{\Theta(s)} = \frac{2.4163}{s^2 - 0.0669s}\,, \tag{42}$$

$$G(z) = \frac{\alpha z + \alpha}{z^2 - \beta z + \gamma} = \frac{0.0030z + 0.0030}{z^2 - 2.0034z + 1.0034} \tag{43}$$

This last transfer function has been used to extract the controller's parameters in order to obtain the first iteration's closed-loop specifications, following the process that was described in Section 7.1.1.

The following results have been obtained for the controller's coefficients:

$$\begin{bmatrix} K_p \\ K_i \\ K_d \end{bmatrix} = \begin{bmatrix} 1.0762 \\ 0 \\ 1.3754 \end{bmatrix} \tag{44}$$

The undetermined pole has been located at $z = 0.9425$, which makes the controlled mathematically acceptable.

It can be seen at plain sight, by looking at the controller's parameters and at the transfer function, that the new system is not -at least theoretically- far from the one in the first iteration. However, the experimentation has proved that the controller's reality is not similar at all. The closed-loop system -after adding a $K_i = 0.15$ integrative coefficient- has been simulated using the same Simulink model from the first controller with a 30 second simulation time. The resulting error and ball's position to time graphs is attached next:

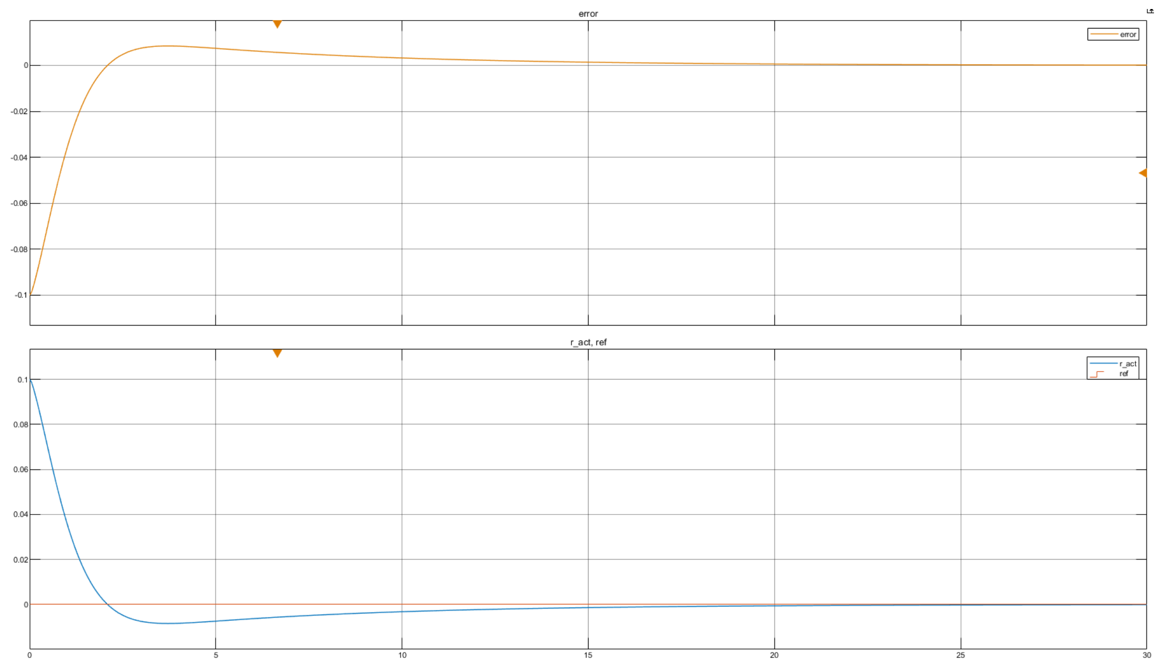


Figure 33: The yellow ball's simulated trajectory for an initial position $r_{ini} = 0.1m$.

The simulation results shows that the system should behave similarly to the first iteration. This result shows that the controller calculation is correct again, as the simulated system behaves as it was specified first.

The controller has been implemented -even though an integrative coefficient $K_i = 0.15$ has been introduced- and tested. The ball has started to move almost immediately after starting the test and has stopped near the setpoint. After a short period of time, the integrative action has driven the ball to the setpoint, where it has stayed with a surprising stability.

It has been deduced that the yellow ball's higher mass has made it easier for the controller to move the ball from the end of the beam, as it tended to roll with more ease when the beam's angle became less plain, but it has also made the setpoint placement more resistant to the integral action's instability, as the ball had less tendency to make sudden position changes when idle.

After completing this test, the first controller has been tested again in order to know whether it required tuning when using the yellow ball just like it happened with the white one or not. A $K_i = 0.15$ integrative coefficient has been added again.

The system's behaviour has been similar to the one at the previous test, but with the ball's movement being slower, as the proportional constant was quite lower. Since the derivative action was lightly lower too, it has been more difficult for the controller to make the ball stop and to be less sensible to the integrative instability. However, it has been proved that the yellow ball system is noticeably more accurate to theory. It is also been proved that controllers designed for lighter balls, which need to have more precision, are also capable of stabilizing the position for heavier balls. The opposite case has not been tested as it was clear that the yellow ball's controller coefficients were too low to make the system behave as desired, as happened in the original tests.

8 Educational application

The ball and beam system's main purpose is to serve as an educational tool about automatic control, specifically about controller calculation and implementation, but also about basic electronics and programming. The goal of this project is to prepare a piece of educational material for ETSEIB students to use, while putting into practice what was learned about discrete time controllers during the Industrial Technologies Engineering degree and discovering how to bring them to the real world. Once the learning phase of the project has been finished, in this chapter a synthesis about what has been done is performed so the students are able to follow this path too.

Before stating what is wanted to be achieved, it is necessary to put some context about what the conditions will be for its usage. In present day automatic control students in ETSEIB have three two hours long groupal laboratory sessions in which they perform some activities about certain concepts that are previously introduced to them in class and at the session guide. In these sessions the students must develop MATLAB codes and graphs in order to solve a set of questions about the session's laboratory systems and concepts. At the end of the semester, they take an exam which contains questions and problems related to what was seen at the three sessions.

However, the aim of the session that is being prepared in this chapter is not only to review the subject's content that is given in classes, but also to give students an opportunity to escape the classes' theoretical and conceptual focus and take a look on how the control systems they are learning about are used to control real life systems of all kinds. Specifically, it is intended to introduce to students to simple, low-cost automatic control platforms and applications which do not require from an industrial or research background to work.

Since the laboratory session guide that has been attached in Annex A is thought to be used only in one of the three sessions, it means that it will be limited in content and length so it is feasible for the students to understand what is being presented to them and solve the session's questions and problems. The session's guide, though, is not restrictive for the conducting professor and is intended to be just supporting material to them, as they are meant to achieve the session's objectives the way they prefer. The guide is also subject to modifications by the Automatic Control Department whenever they consider some changes are required, this is only a first approach to a material proposal to introduce this piece of content to students.

The session's objectives will be based on understanding and doing most of what has been achieved until now in this project. To achieve these objectives, each student group will have a fully configured model, both in software and hardware terms, as it is not intended for them to understand how to prepare the model to work during the session.

1. It is desired for the students to review how to calculate discrete time functions from physical models. They will be supposed to have seen this process in class, either at Automatic Control or at Dynamic of Systems, the Industrial Engineering degree's two main control subjects. In order to achieve this, (7), (8), (10) and (11) will be provided to them.

2. Another of the project's objectives is that the students learn how to implement controllers as the ones they learn about in class, but in a simple and low-cost way. That is why the guide will include a brief explanation of the model's components and their function inside the system.

During the session the implementation software will also be presented. Since this is the most important element of the controller implementation, a good amount of time will be invested in showing how the C codes work and how implementing them into boards like the one used in this model allow to control systems without needing a classical computer. There will be also an explanation of how the code is connected to the model's main elements, the sensors and the motor. This means, an explanation of analogical measuring and output signals, duty cycles and other characteristics of the components presented in Chapter 4 and their influence on how the system works.

3. The students will calculate a PID controller from the system's discrete time transfer function and the desired closed loop poles via the pole assignation method. Since that methodology will be already presented in classes by the time they attend the session, they will be supposed to know how to find the controller constants from the system's transfer function they will have already calculated. A basic MATLAB code will be given to them so they can solve the system faster after they calculate the two denominators.

   The system's desired characteristics for this first controller can be whatever the teacher wants, but in the guide the same iteration order that was followed during Chapter 7 will be followed. However, the other iterations will be introduced later.

4. The students will implement the controller structure after being presented the C code contained in the space shown in Section 7.1.2, except for the three controller actions' calculations. They will be required to transform the PID controller's classical discrete time transfer function -(19)- into each action's difference equation so it can be properly implemented into the program.

   It is important that the students learn about the three actions and what role they have inside the controller, so before implementing the three action controller they will implement one of the actions at a time, first the proportional, then the derivative and finally the integral one. By the time they test the derivative action they will already have implemented the calculation's resulting control. This is a good opportunity to introduce the integrative action and its paper inside the PID controller, also to show how introducing an excessive integrative constant can make the system be unstable.

5. Once the controller has been implemented it is a good moment to introduce how it can adapt to changes in the system's desired parameters or in the model's conditions. Section 7.2's iterations will be introduced to fulfill this purpose.

   The first iterations, as stated previously, are intended to show how changing the desired parameters make the controller's coefficient change to adapt the controller to the new conditions. Students will recalculate the controller's coefficients and test each controller, paying special attention to the undetermined pole's value and making sure its module is strictly lower than the unit. However, it is also a good opportunity to study how the controller can make the system unstable if it is not tuned properly -the cases in which $|a| > 1$-.

   Once the parameter changes iterations are calculated, implemented and tested, the second ping-pong ball will be introduced. It is important that students distinguish between altering the desired system's parameters and changing the model's dynamics, and how this type of controllers adapt to these circumstances. During these iterations it will be necessary not only to recalculate the system by changing the desired characteristic polynomial, but also the system's closed loop transfer function by updating the model's transfer function to the new conditions.

With the session's objectives already described, the session's guide has been created taking as a model the existing Automatic Control session guides. The guide has been reviewed and finally approved, which was one of the project's main objectives. The guide's questions have been conceived so students who attend the session have to perform somewhat similar calculations and reasonings and reach similar conclusions as the ones reached during the course of this project. Their results and conclusions, then, should be almost the same as the ones obtained at Chapters 6 and 7, plus obtaining certain knowledge about what was seen during Chapters 4 and 5.

The laboratory guide is attached at Annex A.

9 Economic, environmental, social and gender analysis

This project is based on creating low-cost, sustainable educational material that can inspire students to enter automatic control from a do-it-yourself (DIY) perspective, in a way they do not necessarily need to spend a significant amount of money to explore projects and applications for the knowledge they get in their studies. This is the reason why every piece of material that has been used has been selected not only for its characteristics or function but also for its price and environmental impact.

9.1 Economic analysis

Since the model which has been used during the project was already made when it started, any economic study of its costs that is done will be based purely on estimations.

The project's costs have been broken down in several categories which are presented next:

1. **Personnel costs:** Since the project has been carried out by an engineering student who is ending his bachelor's degree in Industrial Engineering, no real personnel cost has affected the project. However, in this chapter it will be computed as if the university had developed this project with its own personnel, which adds a junior engineer's fees as personnel costs. Taking a 10€/hour fee, and knowing as was stated in Section 2.3 that 10 hours on average were dedicated to the project during 18 weeks brings out a personnel cost of **1800€**.

2. **License costs:** Since the project has been carried out using free or university sponsored program licenses, no license cost will be computed to the total cost. However, the developer's MATLAB license has had a cost for UPC in reality.

3. **Material costs:** These costs are the main bulk of costs and the most important factor through the project's perspective. They are, too, the most difficult costs to calculate.

   The wood used for the model's pieces will be considered as recycled wood that can be obtained from many sources, and thus having virtually no cost for the project. The other components' costs are presented next:

   Table 4: The model costs broken down by item.

Product (units)	Cost per unit (€/unit)	Total cost (€)
Glue bottle (1)	5	5
Screws, washers and joints (1)	20 (total for all of them)	20
STM32F411RE micro-controllers (2)	22	44
Electronic board and wires (1) kit	15	15
Resistors kit (1)	12	12
Capacitors kit (1)	12	12
Futaba S4004 servo (2)	3	6
Infrared distance sensors (2)	13	26
TOTAL COST:		**140€**

   The different prices have been considered as the common price for each product in a electronic commerce chain. It is also remarkable that some of the products can be used for making more models, making the total cost per model considerably low. Since only one model has been created so far, this will not be taken into account.

4. **Energetic costs:** The energetic costs of the project have been paid by the university since it has been mainly developed within its facilities. Since it is hard to know what energetic contact the university has and how much does the energy cost for each facility, no energetic costs will be considered. The project, though, has been as respectful as possible with nature and economy in this sense, since every device was conveniently disconnected from the electrical network when it was not being used.

The total cost, then, amounts to a total of 1940€, which, as explained before, is probably lower. Making use of the project's advancements would not mean significantly high costs for the school, since no personnel costs would apply. In this sense, the project is considered successful in its goals.

9.2 Environmental analysis

The vast majority of the model's components that have been used are recycled from several electronic or construction parts, and everything that has not been used has been conveniently stored for future use in other projects. This means that the only environmental impact that this project has had on nature has been caused by energy consumption due to computer use or electronic component powering. As it was just said, when any of the project's devices or computers was not being used it was disconnected from the electric network, which significantly reduced their energy consumption. In addition, it is important to remark that the development of this project has been carried out during an energetic crisis due to multiple factors, and in a context of climate change and transition to clean energies.

9.3 Social and gender impact analysis

The fact that this project has had as a goal to create materials with low costs means that it is conceived for any student to use. Any student should be able to experiment and to satisfy their curiosity -in this case, related to automatic control or electronics-, no matter their economic status. This project, then, is considered to be specially sensitive in this matter and, once it is coming to an end, also successful at satisfying this goal.

However, economic status is still one of the most important factors that decide a student's performance in their studies, since students with more resources are more likely to have better results than students which do not have such a privileged position. This is manifested in many areas such as mental health condition or in work-studies conciliation. Gender and sexual identity are also an important factor in these ambits, specially in a discipline like engineering, which has been traditionally occupied by males. Luckily, this is changing over years, but there is still a lot of work to do as a society in this matter.

Conclusions

The objectives that were defined during the project's introduction have been considered achieved for the most part. While it is true that the different controllers are not really consistent in reality and must be adapted to external circumstances, the system has been successfully stabilized by all of them -all the valid ones- after some tuning. These issues, however, have made the controller calculation and implementation process more fruitful from the educational point of view, since it is has been required to invest more time and effort into it in order to get it off the ground.

Another issue that makes the final result a bit unsatisfactory is the fact that it is has not been possible to find relevant differences between the different controllers that have been implemented, as the system has conditioned the results with its inaccuracy. However, the testing and analysis process has been carried out properly, which was what was desired to do, since the project's objective was to **compare** the controllers, even if this comparative did not finally bring out specially interesting results.

This project has been a huge learning experience, since many new knowledge from different areas like electronics, C software development and automatic control has been acquired. This was a personal objective that was proposed since before starting the project, and from a personal point of view, it has been specially successful.

The educational component of the project has been also specially successful, as the content that has been prepared has enough level to be considered as a first draft for official university educational material. This was a first approach to this field of knowledge, and being able to synthesize all the knowledge that has been absorbed through the last months into this laboratory guide brings special personal satisfaction.

To sum up, even though the technical final results were good to some point but not the desired, the process followed to obtained them is correct. The other half of the project, which is its soul and main goal, has been completed successfully. All these results have been obtained through low-cost, sustainable means, which follows the project philosophy as it was proposed during its conception. The overall results for the projects are considered **successful**.

From a personal point of view, this project has been a challenge, since it has required a great amount of time and effort which not always translated into satisfactory results. However, all that has been learned and the fact that in the end a solution has been reached for most of these problems make the final feeling be quite more positive. In my opinion, it has been a fruitful and mostly very positive experience.

Agraïments

All of this would not have been possible without the unconditional support from my family and all my friends, which are my backbone every time I trip. Nothing I can do can bring all this love back to them, all I can do is try my best everyday to improve and to be up to it.

I also want to give special thanks to Santi Prats for his help, advice and patience when I faced hardships. This is also his project and I hope I did enough to follow what was done before me.

A Laboratory session guide

Automatic Control - GETI

LAB SESSION

PID controllers calculation and implementation in low-cost platforms

ETSEIB

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Automatic Control Department (ESAII)

UPC

Objective: *Designing, implementing and testing a series of PID controllers to control the position of a ball in a ball and beam system while satisfying certain closed-loop specifications.*

In this session the student will:

1. Design a PID controller in order to control a ping-pong ball's position in a ball and beam system using one of the methods studied in class -including pole placement, frequency approaches, root locus, algebraic methods, among others-.
2. Implement the controller structure into a micro-controller using a C coding interface.
3. Test the controller's performance experimentally.
4. Test how the controller can be redesigned to adapt to circumstances like a change in the closed-loop desired characteristics or the model conditions.

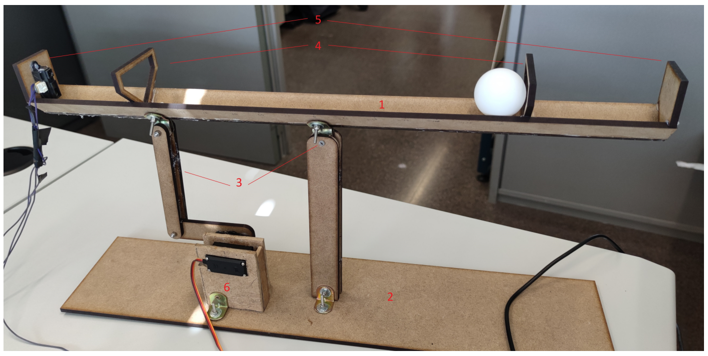
These final goal of the session is not only to review the PID controller's discrete time structure and calculation, but also to show a practical example of simple controller implementation in low-cost platforms that can be easily replicated by students with their own means.

Note: This is the initial version of the guide. Annotations will be included in this format to include comments about how the guide should be used to conduct the session as desired.

A.1 Session materials

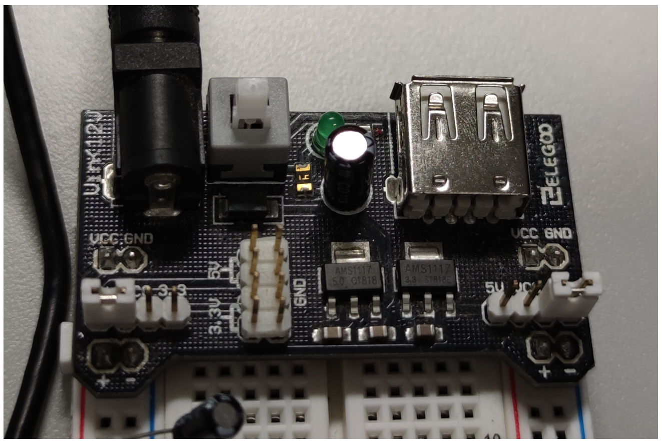
This laboratory session consists in the design of PID controllers using several methods in order to stabilize a ball and beam system so its ball remains at the center of the beam. This central position is known as the **setpoint**.

The ball and beam is one of the most classical automatic control problems and it is widely used for educational or recreational purposes. It consists of a **beam** (1) whose angle to the horizontal axis can be controlled using a **servomotor** (6), and a **ball**, which can slide across the beam. The beam incorporates two infrared distance sensors (5) which measure the ball's relative position periodically. The session's model also includes a wooden base (2) to which the beam is fixed through two wooden articulations (3) and two wooden pieces (4) that limit the ball's movement to 15 centimeters in each direction from the beam's center.



An image of the session's model with numbered parts.

The motor and the sensors are connected to a **STM32F411RE micro-controller** which reads the sensors' measures and processes them through the implemented PID controller in order to send the angle correction signal to the servo-motor. All the plant's components except for the micro-controller are connected to a 5V regulator which can be turned off or on to connect or disconnect the system.



The power regulator. In this figure the power is cut, to connect the system the white button must be pressed.

A.1.1 The SHARP GP2Y0A21 sensors

These sensors measure the ball's distance via infrared beams and convert it to an analog voltage value that is transferred to the micro-controller's A/D converter pins. These voltages are filtered by a hardware low-pass filter in order to avoid interference that may cause the sensors' signal to be unstable. They are one of the system's most critical elements, as if they do not measure the ball's position properly the controller will not work no matter how well designed it is.

A.1.2 The STM32F411RE micro-controller

The micro-controller that is used in this session is the 64-pin STM32F411RET6. Especificaciones.

The micro-controller is used as an A/D converter when reading analog measures from the sensors. The measurements are codified in 12-bit arrays that are converted to distance values in the controller's code in its calibration section. After calibrating the measures (since the analog value does not have a linear relation to the measured distance and thus a correction must be performed), the code processes the measures to correct the beam's angle by sending it a certain pulse signal. This action is equivalent to a D/A conversion.

The micro-controller must be connected to the computer via USB when compiling and debugging the C codes. Even if it is recommended to leave it connected to the computer, it can be connected directly to the network to execute an uploaded code.

Some of the micro-controller's most important features

Feature	Specification
Micro-controller	STM32F411RET6
Recommended power supply	1.7V to 3.6V
Flash Memory	512 kB
SRAM	128 kB
Clock maximum speed	100 MHz

A.1.3 The Futaba S4004 servomotor

The Futaba S4004 is powered by a 5V pulse DC (direct-current) electric input signal which can be modulated in length in order to control the motor's angle and thus change the ball's position. This motor's control system is known as Pulse Width Modulation (PWM).

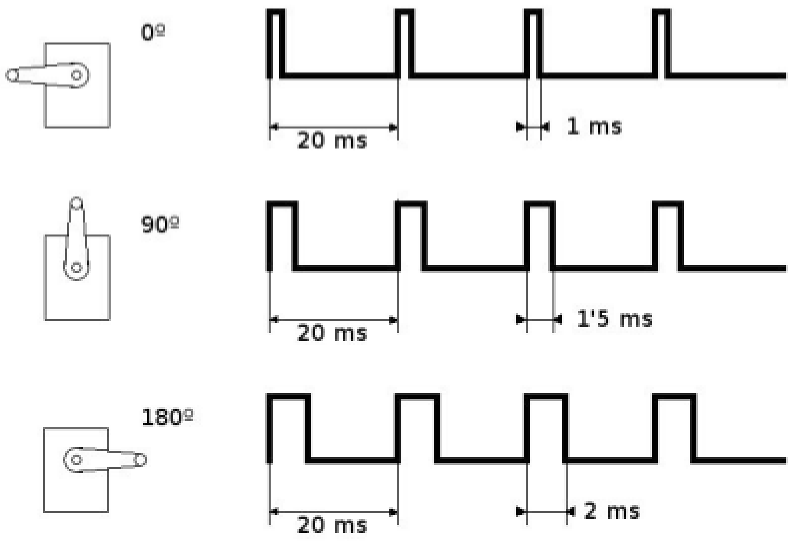


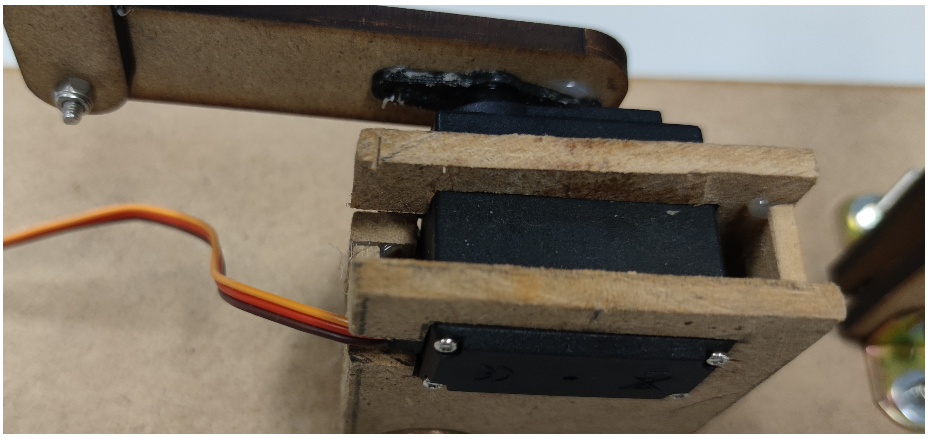
Diagram which shows the relationship between the signal's pulse and the motor's position.

As shown in the previous figure, the PWM pulse's period is 20 milliseconds. However, the signal brings no power for most of it. The quotient between the positive signal's length and the total period is known as **duty cycle**, and it is the variable that is used to change the motor's position. It can be seen that the duty cycle has a linear correspondence to the motor's position, being the 5% duty cycle associated with the 0º position and the 10% one to the 180º position.

However, in reality the motor's duty cycle range is comprised between the 2.5% and the 8.5% values in order to avoid saturation in the motor. This means that comprising the valid duty cycle range between 5% and 8.5% will limit the motor's mobility. That is the reason why the motor has been assembled in order to make the 5% assigned to the 90º position, while the extreme values for the duty cycle will be the 2.5% and 8.5% ones. It is has been experimentally observed that these two values correspond to the 30º and 150º angles, respectively.

Thus, since the correspondence between the motor's angle and the duty cycle is linear, a linear correspondence equation can be obtained.

The motor is powered by the micro-controller through three wires, the positive and negative power wires (corresponding to the red and brown ones at the image below, respectively) and the signal wire (the orange wire). The red wire is connected to a 5 V power supply, while the black is connected to the ground, which is common for all the components and the orange one to the board's PWM output pin.


The motor's and the three wires it is connected to.

A.1.4 Implementation software

In order to design the PID controllers, a MATLAB script will be provided so it is easier to solve the characteristic polynomials' equation system. The resulting controllers will be simulated in a Simulink environment in order to check if the closed-loop system acts as desired in theory.

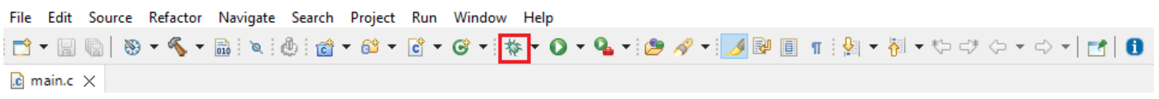
Once each controller has been designed and simulated, it will be implemented into the micro-controller via a C script. These C scripts will be developed using **STM32CubeIDE** a specific integrated development environment (IDE) for STM32 boards.

These C codes contain the PWM and ADC pins' configuration, the measure calibration and translation into distance measures and the duty cycle calculation. This last operation is where the controller structure appears and where the students will work on. There are designed spaces for students to introduced their codes. At the top of the code, inside the "/*USER CODE BEGIN 0*/" section, the PID coefficients will be declared within the K_p, K_i and K_d variables.

The code has been prepared to take distance measures with a 20 Hz **sampling frequency**, while the motor's signal has been programmed to work at a 50 Hz one, as the motor's pulse period is 20 ms.

The analog distance measures are processed through a software moving average filter, which stores an average of a set of a certain number of samples in order to reduce variations on the measuring and translated to distance values by a series of equations obtained experimentally. Once the measures are translated to distances **in meters**, the error signal is calculated by subtracting each measure from the reference setpoint (which is the goal distance, in this case 0).

When the code has been updated and the controller is ready to be launched and tested, press the *Debug* button at the topside toolbar in the STM32CubeIDE interface.


File Edit Source Refactor Navigate Search Project Run Window Help
main.c

The Debug button.

This button will open the debugger, in which the students will be able to run the code as long as desired while monitoring certain variables which will be recursively printed into the *SWV ITM Data Console*.

In order to start the debugging and sending instructions to the plant, the *Resume* button at the topside toolbar will be pressed **after finishing the launching process**. To pause the debugging, press the *Pause* or *Terminate* buttons. These buttons will appear at the toolbar after pressing the *Debug* button.

A.2 Starting up the plant

1. Open MATLAB and select the "BallBeam_Model" file. Do **not** open the "PIDSystem.m" file yet.

2. Open Simulink and select the "PID_Constants" model.

3. Open STM32CubeIDE. The "main.c" file should already appear when the program is opened.

4. Connect the micro-controller is to the computer with the USB cable.

5. Whenever you are about to debug and run a code **check that the power regulator is switched on**.

A.3 First calculations

In the following exercises some questions will be asked about the plant's mathematical representation and the controller's calculation and simulation. Until further notice, you will work with the lighter, common white ping-pong ball.

<u>EXERCISE 1 - Plant's transfer function calculation</u>

The ball's dynamics of the white ball are defined by the following Lagrangian equation:

$$\left(\frac{Ib0}{R^2}+m\right)\ddot{r}+k\dot{r}-mr\dot{\alpha}^2+mg\sin\alpha=0\,, \tag{45}$$

which states the mathematical link between the beam's angle α and the ball's position r. However, it is desired to work with the relationship between the **motor's angle** θ, since θ is the one variable that can be controlled through the motor. Thus, after some approximations, the following Equation is obtained:

$$\left(\frac{Ib0}{R^2}+m\right)\ddot{r}+k\dot{r}+\frac{mgd}{P}\theta=0\,, \tag{46}$$

in which the constants take the following values when studying the white ball:

Constant	Value	Unit
$Ib0$	5.788e-06	kg· m^2
R	0.01988	m
m	0.028	kg
d	0.06	m
P	0.15	m
k	0.05	-

Given the following constants' values, obtain:

a) The plant's Laplace transform transfer function $G(s)=\frac{R(s)}{\Theta(s)}$

b) The plant's discrete time transfer function $G(z)$ considering a sampling period $T_s=0.05s$

EXERCISE 2 - PID controller calculation

a) Calculate a PID controller through the pole placement method in order to satisfy the following closed-loop specifications:

1. A setting time $t_{ss}\approx 1.5s$.
2. Null steady state error for step inputs, $e_{ss}=0$.
3. Use a fast pole in $z=0.1$ and an undetermined pole $z=a$.

This calculation can be done by expressing the PID controller's discrete time TF in the following shape:

$$G(z)=\frac{b_2z^2+b_1z+b_0}{z(z-1)}. \tag{47}$$

After solving the equation system the variable conversion can be reversed to the common PID discrete time structure by applying the following equations:

$$b_2=K_p+\frac{K_iT_s}{2}+\frac{K_d}{T_s}\,, \tag{48}$$

$$b_1=-K_p+\frac{K_iT_s}{2}-\frac{2K_d}{T_s}\,, \tag{49}$$

$$b_0 = \frac{K_d}{T_s}. \tag{50}$$

Reminder: The PID standard TF form is the following:

$$C_{PID}(z) = K_p + \frac{K_i T_s}{2}\frac{z+1}{z-1} + \frac{K_d}{T_s}\frac{z-1}{z}. \tag{51}$$

b) Is the controller that was just calculated valid (not performance related)? Justify your answer.

<u>EXERCISE 3 - Closed-loop simulation with Simulink</u>

Introduce the PID constants in the "BallBeam_Model.m" script and run. Exercise 2's TFs can be checked now at the results in MATLAB's Workspace.

Then, run the Simulink model with a 20ms stop time. In the "Scope31" block the ball's position and the error signal over time to the reference (0 meters) will be plotted for an initial distance of 100 millimeters. The initial distance can be changed at will inside the "r_ini4" block.

a) Analyze the ball's trajectory simulation. Does it correspond to the desired specifications? Find the simulated t_s.

b) Recompute the controller conveniently.

A.4 Controller implementation and analysis

The following exercises consist on implementing and testing the controller in the C code that will be uploaded into the board. Make sure no other changes are make than the required at the questions. Before going any further, initialize the controller constants' values in the designated area inside the "/*USER CODE BEGIN 0*/" section.

<u>EXERCISE 4 - PID controller codification</u>

Check the designed area at the "/*USER CODE BEGIN 3*/" section. The error signal is calculated for every internal clock instant as the difference between the sensors' measured distance and the reference distance, which is 0, as it is the central point of the beam. For every clock instant, the controller must process this error signal as its input and calculate the new duty cycle that is going to be sent to the motor. To fill the blank that is missing in this designated space:

Calculate the difference equation for each of the three signals from the controller's standard TF. Then, write these equations in the code. Some helpful variables that will be used are remarked just before the designated space. Use the variable names P, D and I to store each action's value.

Tip: Take a look at the code sections just above the designed space. Notice that coding in C is relatively similar to coding in MATLAB or even Python.

<u>Note:</u> This exercise might be specially hard to get right for students since they have no experience at coding in C language. Make sure no one gets lost in this part by giving some advice if necessary.

ETSEIB

EXERCISE 5 - Controller implementing and testing

a) Compile and upload the C code as explained in Section A.1.4.

b) Observe the system's behaviour. Does it behave as expected and as it was simulated in Exercise 3? Why?

c) In case the system does not act as expected, reason why it does not. What could be done to improve its functioning? Could the controller that has been implemented be adapted to correct the model's behaviour?

d) In order to test each action and review its function inside the controller test the next controller parameters configurations:

1. Implement $K_p = 2$ without adding any K_d or K_i and test the system. Then, try $K_p = 5$ and $K_p = 10$. What does the proportional action do? What happens if an excessive K_p value is introduced?

2. From now on use $K_p = 5$. Test the controller for K_d values of 2, 5 and 10. What does the derivative action bring to the system? What happens to the system if excessive or insufficient K_d values are introduced?

3. Use $K_d = 2$ for this iteration. Test the controller for K_i values equal to 0.1, 0.25 and 0.4. What paper does the integrative action inside the controller? What happens if an excessive integrative coefficient is introduced in the controller?

e) Are the conclusions that you just obtained consistent with the difference equations you calculated in Exercise 4?

Note: *When students are done with these two exercises is a good moment to do a result recap. It is important that they understand that the controllers they calculate that seem to work in simulations do not necessarily work with the real model, as it is far from ideal and does not correspond to the mathematical model that was previously calculated. This message is important as they might get frustrated or even lost during these two exercises as they will most likely find that the controller is not working, and might come to the conclusion that there is a problem with the code or their calculations.*
Talk about how the physical model has deficiencies from the materials it is made of and its assembly. Talk also about how some electronic configurations, such as the clock speed can be improved in order to make the model more accurate. Also talk about how the mathematical abstraction is not exact, like for example assuming a friction constant that is not known. The students are not meant to know all this but it can not be included in the guide for practical reasons.

A.5 Alternative controllers

Since the controller calculation and implementation process has been already studied, different specification and model variations will be introduced in order to analyze what influences these changes have on the closed-loop system's behaviour.

EXERCISE 6 - Alternative closed-loop specifications

a) First of all, new closed-loop specification combinations will be studied and calculate as done in Exercise 2. For the next combinations, repeat the process done in Exercises 2, 3 and 6:

1. A settling time $t_{ss} \approx 1s$ and a small pole in $z = 0.1$
2. A settling time $t_{ss} \approx 1s$ and a small pole in $z = 0.05$
3. A settling time $t_{ss} \approx 10s$ and a small pole in $z = 0.1$
4. A settling time $t_{ss} \approx 10s$ and a small pole in $z = 0.05$
5. A settling time $t_{ss} \approx 1.5s$ and a small pole in $z = 0.05$

b) Analyze the model's behaviour for every valid controller among the five iterations. What effect does the small pole in $z = 0.1$ or $z = 0.05$ in the system -(both in simulations and in reality)? And the settling time? Do the controller's coefficients grow or decrease with the settling time? Why does this happen?

EXERCISE 7 - Changes on the model's parameters

Repeat Exercises 1, 2, 3 and 6 for the yellow, heavier ball using Exercise 2's original closed-loop specifications. The new ball's parameters are the following:

The model variables' values that change when the balls are switched.

Constant	Value	Unit
$Ib0$	1.453e-05	kg· m^2
R	0.0225	m
m	0.046	kg

The other three parameters will remain the same for this new model.

EXERCISE 8 - Model comparison

In order to study if one of the two mathematical models is more robust than the other, implement the white ball's controller but use the yellow one, and vice-versa. Is one of them more resistant to changes in the physical system? Justify your answer.

B MATLAB and C codes

B.1 MATLAB code - System modelling for simulation

```
%% Ball Beam State Space Equations and Transfer Function
clear all;
close all;
clc;

%% Parameters
m=0.046; %Mass
Ibo=1.453e-05; %Ball inertia
g=-9.81; %Gracity acceleration
k=0.005; %Friction constant
R=0.0225; %Ball radius
b1=0.06; %Bar 1 length
l2=0.150; %Bar 2 length

%% Continous and Discrete models. Control signal: u=Theta
Ts=0.05;
A=[0 1; 0 k/(Ibo/R^2+m)];
B=[0 -0.4*m*g/(Ibo/R^2+m)]';
C=[1 0]; %Determines input-output of the TF
D=0;
cPlant=ss(A,B,C,D); %Continous SS
dPlant=c2d(cPlant,Ts); %Discrete SS
[numc,denc]=ss2tf(A,B,C,D); %Continous TF
[numd,dend]=ss2tf(dPlant.A,dPlant.B,dPlant.C,dPlant.D); %Discrete TF

%% Controlador en TF - Constants trobades amb PID_Constants
Kp=2;
Ki=0.1;
Kd=2;
N=0;
Tf=1/N;
C = pid(Kp,Ki,Kd,0,Ts,'IFormula','BackwardEuler','DFormula','
    BackwardEuler')
dC=tf(C)

Num=dC.Numerator{1};
N1=Num(1);
N2=Num(2);
N3=Num(3);
Den=dC.Denominator{1};
D1=Den(1);
D2=Den(2);
```

B.2 MATLAB code - Controller system solving for the pole placement method

This code's version corresponds to the first iteration's system solving. The other iterations were resolved by changing this program's parameters.

```
alpha=0.0032;
gamma=1.0059;
beta=2.0059;

Ts=0.05;

A=[alpha 0 0 1;
    alpha alpha 0 -1.9252;
    0 alpha alpha 0.969;
    0 0 alpha -0.0438;];

B=[-1.9252+beta+1; 0.969-beta-gamma; -0.0438+gamma; 0;];

X=A\B;

C=[1 0.5*Ts 1/Ts;
    -1 0.5*Ts -2/Ts;
    0 0 1/Ts;];

D=[X(1);X(2);X(3)];

pole=X(4)

constantes=C\D
```

B.3 C code - Final controller code

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the
    LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS
    .
  *
  ******************************************************************************

  */
/* USER CODE END Header */
/* Includes
   ------------------------------------------------------------------
   */
#include "main.h"

/* Private includes
   ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stdio.h" /* standard in out header, inclou el print f */
#include "math.h"
#include "time.h"
/* USER CODE END Includes */

/* Private typedef
   -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define
   ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro
   -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables
   ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;

/* USER CODE BEGIN PV */
uint8_t counter=0;
/* USER CODE END PV */

/* Private function prototypes
   -----------------------------------------------*/
```

```
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_TIM2_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code
   ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

uint16_t adcValArray[2];
double adcMA1=0;
double adcMA2=0;
double adcMA1_ant=0;
double adcMA2_ant=0;
uint16_t len=200;
uint16_t i1=0;
uint16_t i2=0;
uint16_t i3=0;

double dist=0;
double dist1=0;
double dist2=0;

double dist_ant=0;
double dist_ref=0;

/////////////ONLY EDIT THESE PARAMETERS
   ////////////////////////////////
double kp=0.9644;
double ki=0.15;
double kd=1.2919;
//
   //////////////////////////////////////////////////////////////////

double P=0;
double D=0;
double I=0;
double I_ant=0;
double error=0;
double error_ant=0;
double Ts=0.05;//sampling period
double act1=1;
double act2=1;
```

```
uint16_t DutyCycle=530;

int _write(int file, char *ptr, int len) {
         int DataIdx;
         for (DataIdx = 0; DataIdx < len; DataIdx++){
                 ITM_SendChar(*ptr++);
         }
         return len;
 }

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration
     --------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and
     the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_TIM2_Init();
  MX_ADC1_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */

  HAL_ADC_Start_DMA(&hadc1, (uint32_t *)adcValArray, 4);
```

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);

//Start the timer
/*HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
HAL_TIM_Base_Start_IT(&htim3);
//Start ADC as IT*/
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

        //SENSOR 1: Moving average filter

        adcMA1=0;
        i1=0;
        for (i1=0; i1<len; i1++){
        adcMA1=adcMA1+adcValArray[0];
        }
        adcMA1=adcMA1/len;
        //adcMA1=adcValArray[0];

        //SENSOR 1: Measure translation

        if(adcMA1>=1900){
            dist1=0.0473*adcMA1+4.39;
            act1=1;
        }
        else if(adcMA1>=1520 && adcMA1<1900){
                dist1=-441+0.479*adcMA1-0.000105*(adcMA1*adcMA1);
                act1=1;
                }
        else if(adcMA1>=1150 && adcMA1<1520){
                dist1=-332+0.326*adcMA1-0.0000518*(adcMA1*adcMA1);
                act1=1;
        }
        else{
                dist1=0;
                act1=0;
        }

        //SENSOR 2: Moving average filter

        adcMA2=0;
        i2=0;
```

```
for (i2=0; i2<len; i2++){
adcMA2=adcMA2+adcValArray[1];
}
adcMA2=adcMA2/len;
//adcMA2=adcValArray[1];

//SENSOR 2: Measure translation

if(adcMA2>=1850){
    dist2=-0.0404*adcMA2-21.4;
    act2=1;
}
else if(adcMA2>=1420 && adcMA2<1850){
        dist2=-0.0907*adcMA2+80.6;
        act2=1;
}
else if(adcMA2<1420 && adcMA2>=950){
        dist2=354-0.447*adcMA2+0.000116*(adcMA2*adcMA2);
        act2=1;
}
else{
        dist2=0;
        act2=0;
}

//DISTANCE CONFIGURATION

if(act1==0){
        dist1=dist2;
}

else if(act2==0){
        dist2=dist1;
}

if (adcMA1>1500 && adcMA2<1300){
        dist=dist1;
}
else if (adcMA2>1500 && adcMA1<1300){
        dist=dist2;
}
else {
        dist=(dist1+dist2)/2;
}
dist=dist/1000;

//PID CONTROLLER
error=dist_ref-dist;
error_ant=error_ant;
```

```
        I_ant=I_ant;
        Ts=Ts;
        /////////////////////////WRITE YOUR CODE HERE
           ////////////////////////
        P=kp*error;//P component of the output
        D=kd*(error-error_ant)/Ts;//D component of the output
        I=(ki*Ts/2)*(error+error_ant)+I_ant;//I component
        //
           ////////////////////////////////////////////////////////////

        if (I>100) I=100;
        else if (I<-100) I=-100;
        //if (D>0.3) D=0.3;
        //else if (D<-0.3) D=-0.3;


        DutyCycle=530-(P+I+D)*244;


        //PRINTS
        //printf("\n adcValArray[0] %d \n", adcValArray[0]);
    //printf("\n adcValArray[1] %d \n", adcValArray[1]);
        printf("\n adcMA1 %f \n", adcMA1);
        printf("\n adcMA2 %f \n", adcMA2);
        //printf("\n Dist1 %.3f \n", dist1);
        //printf("\n Dist2 %.3f \n", dist2);
        //printf("\n P %.3f \n", P);
        //printf("\n I %.3f \n", I);
        //printf("\n D %.3f \n", D);
         printf("\n Dist %.3f \n", dist);
         //printf("\n DutyCycle %d \n",DutyCycle);
         //printf("\n consigna %f \n", consigna);


        // Safety sat
        if (DutyCycle>850) DutyCycle=850;
        else if (DutyCycle<250) DutyCycle=250;
        //Motor movement
        htim2.Instance->CCR1=DutyCycle;
        I_ant=I;
        error_ant=error;
        HAL_Delay(50);

  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified
     parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|
     RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|
                                 RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) !=
     HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
```

```
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution,
     Data Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV8;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ENABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 2;
  hadc1.Init.DMAContinuousRequests = ENABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its
     corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_480CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its
     corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 2;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 32;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 10000;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig)
      != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) !=
      HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 16;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 50000;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) !=
     HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig)
      != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA2_Stream0_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error
     return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line
     number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
     line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n",
        file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Bibliografia

[1] Aleksandar Haber, *Calibration and noise reduction of distance sensors (SHARP 2Y0A21 infrared sensors), Web tutorial*, Fusion of Engineering, Control, Coding, Machine Learning, and Science

[2] Santosh Anand, Rajkishore Prasad, *Modeling and control of Ball and Beam system*, International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE), Vol 4, Issue 9, September 2017

[3] Benjamin C. Kuo, *Digital control systems*, México: Compañía Editorial Continental, 1997

[4] Ramon Costa Castelló, Enric Fossas Colet, *Sistemes de control en temps discret*, Barcelona: Universitat Politècnica de Catalunya. Iniciativa Digital Politècnica, 2014

[5] Xian-Sheng Cao, *Moment of Inertia of a Ping-Pong Ball*, Changzhou University, Changzhou, People's Republic of China, The Physics Teacher, Vol. 50, May 2012