School of Industrial & ICT Engineering

Development of a real-time system oriented to the control and sensorization of a Ball & Plate system

Degree in Industrial Engineering

End of Degree Project

Author: Jon Garnica Izco

Director: Dr. Gabriel Lera Carreras

Pamplona, June 3rd 2021

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ACKNOWLEDGEMENTS

After four months working on this End of Degree Project, I am writing this acknowledgement chapter to eventually end it up. It has been a time of continuous learning that has helped me to better understand and put into practice most of the main control and electronics concepts that I have learnt during the degree.

I would like to start by thanking my project director Dr. Gabriel Lera Carreras for his commendable help during all these months. It has been truly a pleasure to work alongside him and I feel like I have learnt much more than could have ever expected. He has always been willing to help me solve every problem that may have arisen regardless of the site or the day. Without him it would have been much more difficult to get this job done.

I would also like to thank Dr. Iñaki Arocena Elorza who gave us his selfless help in the hardware field of the project, chiefly when taking the first steps. Thanks to him, I got to understand in depth the inner functioning of the table which turned to be a very important part of the project that aided me to better comprehend all the succeeding steps that I took.

Last, I would like to show my gratitude to the Public University of Navarre and all the professors that I have had during these years, that have made me have the necessary knowledge to carry out this work.

ABSTRACT

The main objective of this End of Degree Project is the development of a real-time system oriented to the control and sensorisation of a Ball & Plate system designed in 1997. It will be tried to adapt the system to the nowadays technology by means of replacing the initial controller card with an Arduino DUE board and substituting the original system's camera by a mobile phone duly programmed to fulfil this task.

In addition, an initialization of the main components of the system will be carried out including in it the definition of the table's reference position and a relation between the samples taken by the camera in pixel units and the real position of the ball in centimetres.

Lastly, there will be taken the first steps in the design of a controller to control the trajectory of a ball on the plate.

KEYWORDS

Ball & Plate

Controller

Open and close loop control

Arduino

RESUMEN

El objetivo principal de este Trabajo Fin de Grado es el desarrollo de un sistema en tiempo real para el control y la sensorización de un sistema Ball & Plate diseñado en 1997. Se tratará de adaptar el sistema a la tecnología actual introduciendo un Arduino DUE como microcontrolador y la cámara de un teléfono móvil como sensor de posición.

Además, se llevará a cabo una inicialización de los principales componentes del sistema en la que se definirá la posición de referencia de la mesa y se establecerá una relación entre las muestras tomadas y la posición real de la bola en centímetros.

Por último, se darán los primeros pasos en el diseño de un controlador que permita el control de la trayectoria de la bola sobre el plato.

PALABRAS CLAVE

Sistema Ball & Plate

Controlador

Control en lazo abierto y cerrado

Arduino

Table of Contents

Table of Figures

Table of Images

Table of Equations

PARAMETER IDENTIFICATION

BALL & PLATE DYNAMICS

Variable	Definition	Units
q_i	i-th generalized coordinate (either position or angle)	m/°/rad
$\dot{q}i$	First derivative of the i-th generalized coordinate by time	$\frac{m}{s}$ / $\frac{°}{s}$ / $\frac{rad}{s}$
K	Kinetic energy of the system	J
V	Potential energy of the system	J
Q_i	i-th generalized force	J
x, y	Position coordinates in the plate	m
α, β	Plate inclination angles	°/rad
r	Ball centre radius vector	m
v	Ball centre velocity vector	m/s
r_b	Ball radius	m
ω	Angular velocity vector of rotating ball	rad/s
Ω	Angular velocity vector of rotating plate	rad/s
I_b	Ball inertia	kg·m^2
I_p	Plate inertia	kg·m^2
m_b	Ball mass	kg

PARAMETER VALUES

Variable	Definition	Value
K_α	Servo system static gain	0.1878 rad/MU
K_b	Ball-plate system gain	4.6 m/rad·s^2
K_x	Ball position sensor constant	5.56 MU/m
K ($=K_\alpha \cdot K_b \cdot K_x$)	Overall system gain	4.803 s^{-2}
$\Delta\alpha$	Plate angle step size	0.000628 rad 0.03598 °
f	Typical driving frequency	400 Hz
$\omega_{\alpha \mid f=400\ Hz}$	Nominal angular velocity of the plate	0.2513 rad/s
α_{max}/-α_{min}	Limit plate angle	0.1878 rad 10.76 °
x_{max}/-x_{min}	Limit ball position	0.18 m
ω	Servo system nominal speed	1.338 s^{-1}
T_m	Servo system equivalent time constant	0.187 s
$d_x \times d_y$	Size of the plate area	0.4 × 0.4 m
l	Pivot-plate margin distance	0.2 m
d	Diameter of a stepper shaft	0.016 m
Δφ	Stepping motor step size	0.0157 rad 0.9°

1. Introduction

The Ball & Plate apparatus is a **MIMO** (multiple inputs-multiple outputs), **open-loop unstable** and **nonlinear system** which englobes the fields of mechanical and electrical engineering, automatic control, and computer science. For this reason, it is a very useful tool for education in mechatronics that can give the students a practical and illustrative view of these fields. Moreover, as it also concerns **real time control**, it can be seen the importance of the integration and communication between all the components to achieve a proper operation of the system. [1]

The system consists of a **table tennis ball** of almost negligible mass, a **table** containing a **plate** whose inclination with respect to both axes can be regulated by the action of a pair of synchronous step motors, and a **camera** which provides the position of the ball in real time to enable the control to be done.

The prototype employed in this project was designed in **1997** using a technology from that time. Nowadays, most of this technology is completely outdated and needs an adaptation to current times to be usable again.

Originally, a **control card** operated from an old-time PC was used to carry out the control task. It was decided to replace it with an **Arduino DUE** microcontroller because it is the best for real-time controlling tasks from the whole Arduino family. To program it, it was used the **Sloeber** interface which is an Eclipse IDE that offers a wider range of possibilities than the Arduino IDE, such as real-time control. [2]

On the other hand, the **camera** was substituted by a **mobile phone** that, thanks to a new Android program development, was able to detect the position of the ball and send it every 33 ms. Thanks to this change, despite it is a task that has not been accomplished because it is out of the scope of this project, it could be possible to establish a wireless connection with the phone (either **Bluetooth or IP connection**) so that the system could be controlled remotely.

This document will follow a similar structure to the one described in the content table. First, there will be described the **objectives** that have been set for this project, stating the steps followed to achieve each of them. Then, it is included a **state-of-the-art research** about marketable and non-marketable prototypes related to the Ball & Plate apparatus that can be found in real life. Another chapter will be dedicated to the **plant's transfer function obtention**, including in it all the calculations and theorical development necessaries to fulfil this task. Finally, to end up the contextualization of the project, it is introduced a **system description** where each of the main components of the system are described in depth (plate, motors, drivers, motherboard, microcontroller, camera…) and where it is explained the **logic** that the table follows.

Further on, to start with the automatic control part of the project, it is opened a new section related to the **initialization of components**. In it, it is described the process followed to find a **horizontal reference position** for the plate, as well as a **relationship** between the position of the ball in **pixels** and in **centimetres**.

Then, having all the necessary components appropriately initialized, it is started the **controller design** in sights of fulfilling the assignment of controlling the ball's trajectory. Nevertheless, for this controller obtention it could not be invested as much time as necessary to define one that fitted in with this task. The final design is only a first approach of what it should eventually do.

2. Objectives

The objective of this End of Degree Project is the start-up of the Ball & Plate system. To accomplish it, it is necessary to follow several steps:

1. Comprehension of the inner functioning of the table.
2. Adaptation of the system to the actual technology.
3. Definition of the plate's horizontal position as reference position for the plate.
4. Demonstration of the method used to convert the camera's samples in pixels to centimetres.
5. Open-loop control of the plate.
6. First approach to the design of a controller to control the trajectory of a ball on the plate.

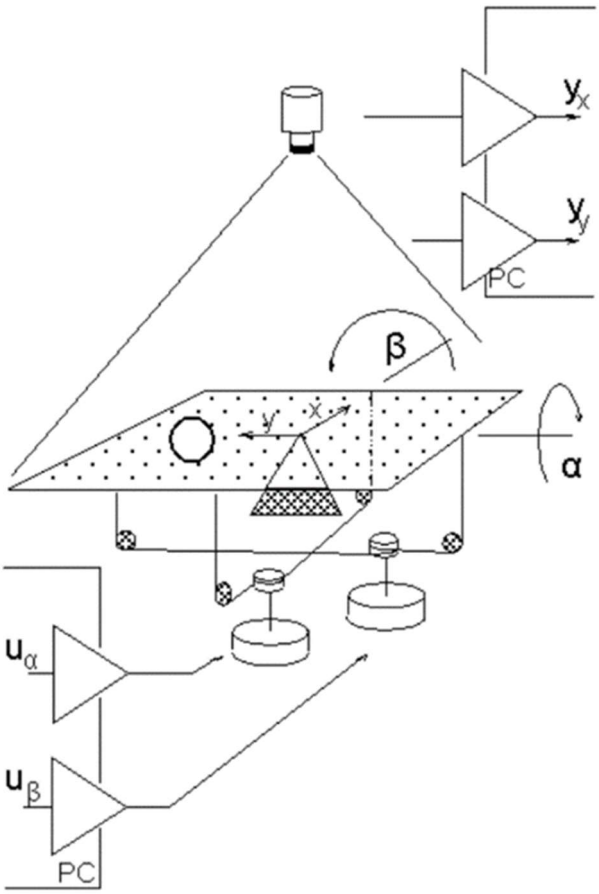

Figure 1: System sketch description [3].

3. State of the art research

The ball and plate system have an eminently educational application, not only for students but also for companies which can employ this example as a guideline to develop other products. It can be taken as a good example because its behaviour is certainly complex as it is a nonlinear, multivariant and open-loop unstable system. [1]

The principal ball and plate systems can be catalogued in two categories: marketable and non-marketable. The first ones are those produced and commercialized by companies, whereas the second ones are those referred to the homemade systems where people who have mechatronic as a hobby develop prototypes that can serve as inspiration to others or even, if they have it patented, sell it to companies for a certain price. [4]

3.1. Marketable prototypes

There are several companies that trade products which are related to or based on the Ball & Plate system.

3.1.1. Leybold GMBH

Leybold is a company that mainly distributes physical, chemical, biological, and technological equipment for almost all of the branches of each field. Inside the *Control and Automation,* they present a large list of prototypes, not only including the *Ball & Plate apparatus*, but also the *Inverted Pendulum* or the *Twin Rotor MIMO* among some others.

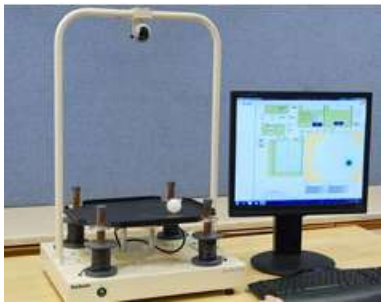

Image 1: Leybold's Ball & Plate [5].

Image 2: Leybold's Inverted Pendulum [6].

Image 3: Leybold's Twin Rotor MIMO [7].

The *Inverted Pendulum* is similar to the *Ball & Plate* apparatus but instead of having two possible direction movements it has only one. The aim of system is to, by moving back and forward the trolley where the pendulum in joint, make it to remain in a vertical position even when receiving a disturbance. It is a stability problem with only one degree of freedom.

On the other hand, the *Twin Rotor MIMO* has a behaviour that resembles a helicopter but having a fixed attack angle of the rotors and controlling the aerodynamic forces by varying the speed of the motors. It shares with the *Ball & Plate* that it is a stability problem and a MIMO system too.

3.1.2. Quanser

Quanser is a company whose main objective is to "provide a modern framework for engineering education through a physical grounding of mathematical theory, manifested in a way that lets students see and feel how the mathematics flow out from the theory and into the physical world" [8].

They develop prototypes for the fields of automatic control, robotics, aerospace, mechatronics, and electrical, earthquake and mechanical engineering. In the control one they have a much broader product offer than Leybold as they have more variants for each model. For example, they have seven types of inverted pendulums, each of them with a different actuator, a robust appearance *Ball & Plate* system, 5 simulators, one gyroscope, one Twin Rotor… [9]

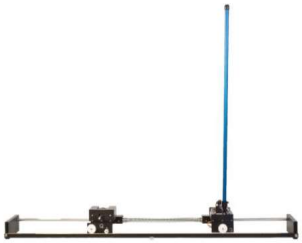

Image 4: Quanser's Inverted Pendulum [10].

Image 5: Quanser's Twin Rotor MIMO [11].

Image 6: Quanser's Ball & Plate [12].

3.2. Non-marketable prototypes

As it has been said before, these are the prototypes developed by people who make homemade designs. Unlike the marketable ones, the cost is much lower, and, in addition, it gives the possibility of designing it to fulfil whichever are the requirements. They can also be a good chance for students in view of learning not only the control point but also the mechanical one.

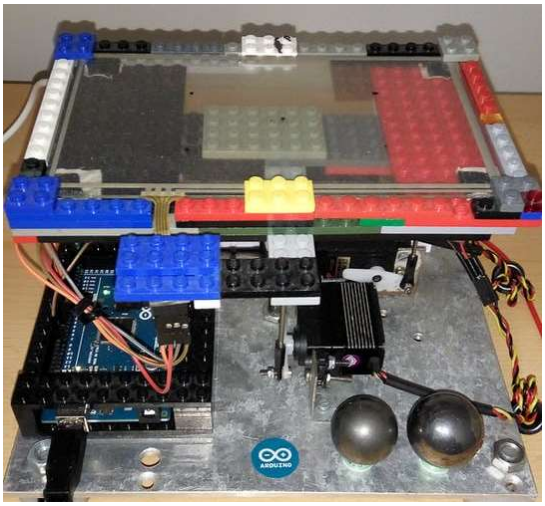

Image 7: Example of homemade Ball & Plate prototype [13].

4. Methodology. Plant's Transfer Function Obtention.

The main objective of this chapter is the **plant's transfer function obtention** as it will be necessary to design a controller that fit in correctly with the system. It has been decided to introduce this chapter because it is important to understand where the plant's expression comes from in order to design an accurate controller and to be able to know how certain modifications in the system may affect the plant's expression.

Before starting, it is important to highlight that during this whole chapter where there are going to be obtained different equations that describe the system's behaviour, it has been followed the **CE 151 Ball & Plate User's Manual** [14]. Each of the variable's meanings can be seen in the PARAMETER IDENTIFICATION table attached above.

In addition, to understand the relation existent between angles α, β and position variables x, y it is included a diagram where this relation can be observed, as well as what it is defined as positive directions for all of them.

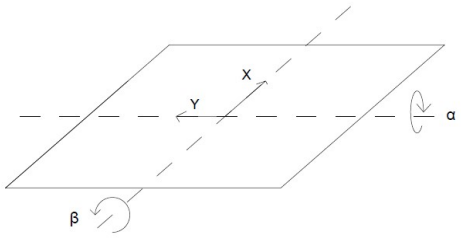

Figure 2: Reference system definition

4.1. Ball & Plate Dynamic Modelling

To obtain a definition of the system's dynamic, it is followed the general form of the Euler-Lagrange equation which will have to be applied to both the ball and the plate.

$$\frac{d}{dt}\frac{\partial K}{\partial \dot{q}i} - \frac{\partial K}{\partial qi} + \frac{\partial V}{\partial qi} = Qi$$

Equation 1: Euler-Lagrange general form equation.

Before starting to define each of the energy terms, it is convenient to define Qi. This variable is referred to the generalized force given by the torque generated by the servo including the transmission system. Therefore, it will be null for the position variables as they do not have influence in it, whereas, for the angular ones, it will have a certain value. These values will be obtained using the following torque equation $\boldsymbol{Q_i = F_i\, d\, \cos(i)}$ for each of the angles, where *d* is the horizontal distance between the pivot (which is coaxial to the motor axis) and the point of the plate where the force is applied. Writing it in a vector form:

$$Q_i = \begin{bmatrix} Q_x & Q_y & Q_\alpha & Q_\beta \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & F_\alpha\, d\, cos\alpha & F_\beta\, d\, cos\beta \end{bmatrix}^T$$

Once defined this vector, the kinetic and potential energy equations of the **ball** are introduced.

$$K_b = \frac{1}{2} m_b\, v^2 + \frac{1}{2} I_b\, \omega^2 = \frac{1}{2}\left[m_b\left(\dot{x}^2 + \dot{y}^2\right) + I_b\left(\frac{\dot{x}^2}{r_b^2} + \frac{\dot{y}^2}{r_b^2}\right)\right]$$

Equation 2: Kinetic energy equation for the ball.

$$V_b = m_b\, g\, h = -m_b\, g\, (x\, sin\alpha + y\, sin\beta)$$

Equation 3: Potential energy equation for the ball.

It has been added a negative sign before the potential energy expression because, as it can be observed in **Error! Reference source not found.**, if either α or β increase following the positive sense defined, the potential energy decreases, whereas if the angles decrease, the potential energy increases. This is solved by adding that negative sign.

Repeating the same process for the **plate**:

$$K_p = \frac{1}{2}\,(I_b + I_p)\left(\dot{\alpha}^2 + \dot{\beta}^2\right) + \frac{1}{2}\, m_b\, |\Omega \times r|^2$$

Where the angular velocity - position cross product presents the following structure:

$$|\Omega \times r|^2 = \left| \begin{bmatrix} -\dot{\beta} \\ \dot{\alpha} \\ 0 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \right|^2 = \dot{\beta}^2 y^2 + \dot{\alpha}^2 x^2 + 2\dot{\alpha}\dot{\beta} xy$$

Introducing the obtained term in the plate's kinetic energy equation:

$$K_p = \frac{1}{2}\,[(I_b + I_p)\left(\dot{\alpha}^2 + \dot{\beta}^2\right) + m_b(\dot{\beta}^2 y^2 + \dot{\alpha}^2 x^2 - 2\dot{\alpha}\dot{\beta} xy)\,]$$

Equation 4: Kinetic energy equation for the plate.

It is not included the **potential energy** term of the **plate** because it remains constant.

Putting together the above equations it is obtained that the kinetic and potential energy equations of the system are:

$$K = K_b + K_p = \frac{1}{2}\left[(I_b + I_p)(\dot{\alpha}^2 + \dot{\beta}) + m_b(\dot{\beta}^2 y^2 + \dot{\alpha}^2 x^2 + 2\dot{\alpha}\dot{\beta} xy) + \left(m_b + \frac{I_b}{r_b^2}\right)(\dot{x}^2 + \dot{y}^2)\right]$$

Equation 5: Kinetic energy equation of the system.

$$V = V_b = m_b\, g\, (x\, sin\alpha + y\, sin\beta)$$

Equation 6: Potential energy equation of the system.

Introducing all these equations in the Euler-Lagrange one and applying each of the derivatives they are finally obtained the equations that describe the system's dynamic.

$$\boldsymbol{x}\text{:}\quad \left(\boldsymbol{m_b} + \frac{\boldsymbol{I_b}}{\boldsymbol{r_b^2}}\right)\ddot{\boldsymbol{x}} - \boldsymbol{m_b}\left(\dot{\boldsymbol{\alpha}}^{\boldsymbol{2}}\boldsymbol{x} + \dot{\boldsymbol{\alpha}}\dot{\boldsymbol{\beta}}\boldsymbol{y}\right) + \boldsymbol{m_b g sin\alpha} = \boldsymbol{0}$$

$$\boldsymbol{y}\text{:}\quad \left(\boldsymbol{m_b} + \frac{\boldsymbol{I_b}}{\boldsymbol{r_b^2}}\right)\ddot{\boldsymbol{y}} - \boldsymbol{m_b}\left(\dot{\boldsymbol{\beta}}^{\boldsymbol{2}}\boldsymbol{x} + \dot{\boldsymbol{\alpha}}\dot{\boldsymbol{\beta}}\boldsymbol{y}\right) + \boldsymbol{m_b g sin\beta} = \boldsymbol{0}$$

$$\boldsymbol{\alpha}\text{:}\quad \left(\boldsymbol{I_p} + \boldsymbol{I_b} + \boldsymbol{m_b x^2}\right)\ddot{\boldsymbol{\alpha}} + \boldsymbol{m_b}\left(\ddot{\boldsymbol{\beta}}\boldsymbol{xy} + \dot{\boldsymbol{\beta}}\dot{\boldsymbol{x}}\boldsymbol{y} + \dot{\boldsymbol{\beta}}\boldsymbol{x}\dot{\boldsymbol{y}} + \boldsymbol{2}\dot{\boldsymbol{\alpha}}\dot{\boldsymbol{x}}\boldsymbol{x}\right) + \boldsymbol{m_b g x cos\alpha} = \boldsymbol{F_\alpha\, d\, cos\alpha}$$

$$\boldsymbol{\beta}\text{:}\quad \left(\boldsymbol{I_p} + \boldsymbol{I_b} + \boldsymbol{m_b y^2}\right)\ddot{\boldsymbol{\beta}} + \boldsymbol{m_b}\left(\ddot{\boldsymbol{\alpha}}\boldsymbol{xy} + \dot{\boldsymbol{\alpha}}\dot{\boldsymbol{x}}\boldsymbol{y} + \dot{\boldsymbol{\alpha}}\boldsymbol{x}\dot{\boldsymbol{y}} + \boldsymbol{2}\dot{\boldsymbol{\beta}}\dot{\boldsymbol{x}}\boldsymbol{x}\right) + \boldsymbol{m_b g x cos\beta} = \boldsymbol{F_\beta\, d\, cos\beta}$$

Equation 7: Expressions that define the system's dynamics.

4.2. Ball & Plate Model Simplification

The aforementioned position expressions are the exact description of the system's position dynamics. Nevertheless, several assumptions can be made to simplify it.

1) The **angular equations** might be **neglected**. As the frequency introduced in the stepper (400 Hz) is below the plate's acceleration limit, **no steps can be lost** and, added up to the fact that the **ball's load moment is almost null** due to its very small weight, the **α and β** plate angles can be considered directly as **inputs** instead of the angular forces.

2) When analysing the position expression, it can be observed that the centrifugal force term $m_b(\dot{\alpha}^2 x + \dot{\alpha}\dot{\beta} y)$, when being compared to the other gravitational ones and fixing the motor frequency to 400 Hz, shows a, roughly, **1:25 ratio** between them. This means that the gravitational terms are 25 times bigger than the centrifugal one. Therefore, this **centrifugal term can be neglected**.

3) As the an**gle variations are low** (≈±5°), the sine function can be mathematically approached to its argument.

$$\boldsymbol{sin\alpha \approx \alpha}$$

4) The ball's inertia is:

$$\boldsymbol{I_b = \frac{2}{5} m_b\, r_b^2}$$

Considering all these possible simplifications, the final model obtained is:

$$\left(\boldsymbol{m_b + \frac{\frac{2}{5} m_b\, r_b^2}{r_b^2}}\right)\ddot{x} - \boldsymbol{m_b g sin\alpha = 0}$$

$$\boldsymbol{\left(1 + \frac{2}{5}\right)\ddot{x} = g sin\alpha}$$

$$\boldsymbol{\ddot{x} = \frac{5}{7}\, g\, sin\alpha \approx K_b \alpha} \qquad \boldsymbol{\ddot{y} = \frac{5}{7}\, g\, sin\beta \approx K_b\, \beta}$$

Equation 8: Ball & Plate simplified position dynamic model.

4.3. Transfer function obtention

Before finally obtaining the plant's transfer function, several steps need to be taken. First, the *state equations* that represent the system's model based on the *state vector* will be defined. This vector contains the three variables on which the system mainly depends: the ball's position (x), the ball's velocity (v) and the plate's slope (α).

$$x = \begin{bmatrix} x \\ v \\ \alpha \end{bmatrix}$$

Applying derivatives to this vector and using the equations expressed in the previous paragraph, the *state equations* can be easily determined. In the first place, if the first vector component (x) is derived, it is trivial to see that it is equal to the second of the vector's components (v).

$$\dot{\boldsymbol{x}} = \boldsymbol{v}$$

Then, the velocity's derivative is equal to the ball's acceleration, whose expression has been deduced before (*Equation 8*). However, as it is being analysed is the overall system, the constant that multiplies the angular term in *Equation 8* will no longer be only referred to the ball but to the whole system. For this reason, the constant (K) will be the product of the ball's constant (Kb), the servo system static gain (Kα) and the ball position sensor constant (Kx). This last two constants are defined in two of the system's description chapter's paragraphs (5.2 Motors, 5.6 Camera).

$$\dot{\boldsymbol{v}} = \ddot{x} = K_b \cdot K_x \cdot K_\alpha \cdot \alpha = \boldsymbol{K} \cdot \boldsymbol{\alpha}$$

Applying the Laplace transform to this expression, it is obtained the first part of the system's transfer function.

$$\mathcal{L}[\,\ddot{x}\,] = \mathcal{L}[\,K \cdot \alpha] \;\rightarrow\; s^2 \cdot x(s) = K \cdot \alpha(s) \;\rightarrow\; \frac{x(s)}{\alpha(s)} = \frac{K}{s^2}$$

Equation 9: Plant's first transfer function term.

Last, to obtain the expression for the angular velocity term, it is previously required to understand the servo system dynamics. Before, in the simplification model paragraph, it has been said that the angular term could be neglected, and that the plate's angles were directly system's inputs. Nevertheless, due to the servo system's dynamics, the response is not instantaneous, so it must be modelled. The problem is that, as it is a nonlinear process, the difficulty of finding an expression that describes this behaviour increases. To avoid this issue, despite making a certain error, it is approximated to a first order response which depends on the first order system time constant (T_m).

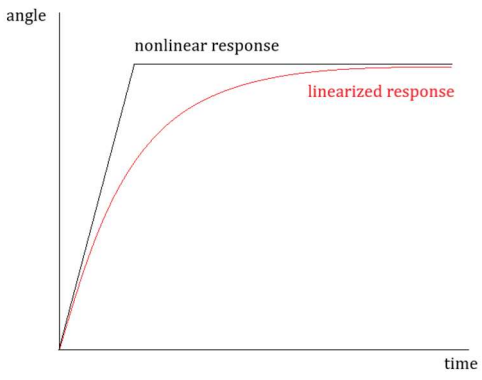

Figure 3: Linearized vs. nonlinearized servo system response.

The angle reached by the motor will depend on the time constant T_m. This constant can take values between 0 and the result of the quotient between the maximum angle amplitude and the servo's minimum angular velocity. Depending on the value chosen, the first-order graph will have a higher or lower slope: if T_m increases, the slope also does, and vice versa.

To obtain the time constant it is used the MU unit. This unit is different depending on if it is referred to angle or distance movements, but its main goal is to represent these variations in decimal fractions, being 1 the maximum variations that they can experiment. For example, if it is said to the plate to rotate 1 MU, what it is really being said is that it must rotate the maximum angle possible (10°), whereas if it is wanted to move the ball 1 MU, what it is being told in metric units is that it is desired to move the ball 18 cm.

$$T_{m,max} = \frac{\alpha_{max}}{\omega_{min}} = \frac{1\ MU}{1.338 \frac{rad}{s}} = 0.747\ s$$

$$T_m \epsilon \left(0\ , \frac{\alpha_{max}}{\omega_{min}} = 0.747\right)$$

Following the user's manual [14], the value chosen for this variable, based on several closed loop simulations looking for the worst possible signal, is $\frac{T_{m,max}}{4} = \mathbf{0.187\ seconds}$. As the servo system's behaviour has been approached to a first order one which depends on the estimated time constant, it will follow the expression:

$$\alpha(s) = \frac{1}{T_m \cdot s + 1}\ u_\alpha(s)\ \rightarrow\ \frac{\alpha(s)}{u_\alpha(s)} = \frac{1}{T_m \cdot s + 1}$$

Equation 10: Plant's second transfer function term.

Where α is the real angle and u_α is the desired one.

Therefore, joining Equation 9 and Equation 10, it is finally obtained the system's global transfer function F(s) in the Laplace domain.

$$\boldsymbol{F(s) = \frac{K}{s^2 \cdot (T_m \cdot s + 1)}}$$

There is a last modification needed to be done before arriving to the final transfer function expression because the units used in the user manual are MU instead of degrees and centimetres. Therefore, there will be needed to apply a series of unit conversions until finally obtaining the expression in the desired units.

$$\boldsymbol{F(s) = \frac{K_b \left[\frac{m}{rad \cdot s^2}\right] \cdot K_x \left[\frac{MU}{m}\right] \cdot K_\alpha \left[\frac{rad}{MU}\right]}{s^2 \cdot (T_m \cdot s + 1)} = \frac{y_x[position\ MU]}{u_\alpha[angular\ MU]}}$$

$$\boldsymbol{F(s) = \frac{K_b \left[\frac{m}{rad \cdot s^2}\right] \cdot \frac{\pi\ rad}{180°} \cdot \frac{100\ cm}{1m}}{s^2 \cdot (T_m \cdot s + 1)} = \frac{\frac{y_x[MU]}{K_x \left[\frac{MU}{m}\right]}}{K_\alpha \left[\frac{rad}{MU}\right] \cdot u_\alpha[MU]}}$$

$$\boldsymbol{F(s) = \frac{K_b \cdot \frac{\pi}{180} \cdot 100}{s^2 \cdot (0.187 \cdot s + 1)} \left[\frac{cm}{°}\right]}$$

Equation 11: System's global transfer function.

5. System's description

5.1. Plate

It consists of a **0.4 x 0.4 m** square piece of metal. Its centre of mass is joint to the base unit through a shaft located on the middle of the plate making the **load moment to be null** with respect to this point and helping to simplify the system dynamics. To adjust its inclination, there are attached a pair of **tight threads** to each of the four plate **edges**. Between both edges, each thread is winded onto a **pivot coaxial to the motors** which, as it has a smaller diameter that the motor, applies to the system a **reduction factor** that makes the plate to rotate at a more adjusted velocity.

As the thread is winded onto the pivot, when it starts to spin, it provokes the emergence of a **pair of opposite tensile forces** in the edges of the plate to which this thread is attached, making the table to start tilting in the desired direction.

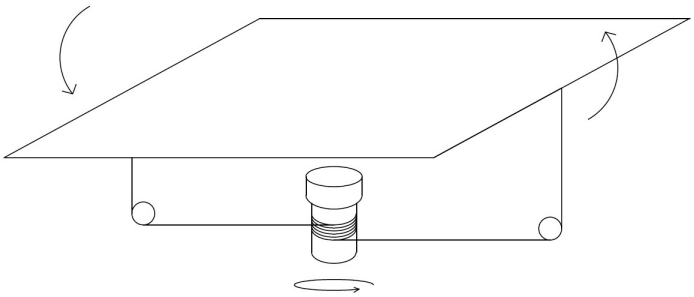

Figure 4: Plate rotation diagram for one of the axes.

5.2. Motors

There are two M061-LE08 synchronous step motors from the Superior Electric house, which operate with a DC voltage of 1.25 V and a current on 3.8 A, and have a relation between the number of steps and revolutions of 200 step/rev. Its angular velocity depends on the frequency of the pulse signal introduced. Typically, this frequency will be fixed to 400 Hz, but it could be modified if desired, always considering the dynamics of the system to avoid reaching excessively high speeds.

Image 8: Motor image.

Stepper motors are composed of a rotor and a stator. The **stator** has **4 coils** joint by **pairs** with **phases A and B**, whereas the **rotor** is a **permanent magnet** with **two poles**. However, as both the stator and the rotor are toothed having a **hundred teeth** (two sections of 50 teeth), it will take the motor to complete a **full revolution 200 steps**. [15]

The functioning of this motor consists of the **consecutive energization of coils**. This way, when a **current** appears in one of the coils, it is generated a magnetic field that makes the rotor spin in that direction. Once it is aligned with this field, the following coil starts to get energized, creating a **new magnetic field** in a **perpendicular direction** to the previous one that begins to attract the rotor in this new direction. [15]

Thanks to this successive coil functioning, it is possible to configure the motor in a **full or half step** setting: if when the next coil starts working, the previous one has already stopped, it works in full step mode; but if when the next coil starts working, the previous one has not stopped yet, it will be created a magnetic field in the bisector direction between them that would make the rotor to take only half a step. [16] In this **project** it has been set in a **half step mode**.

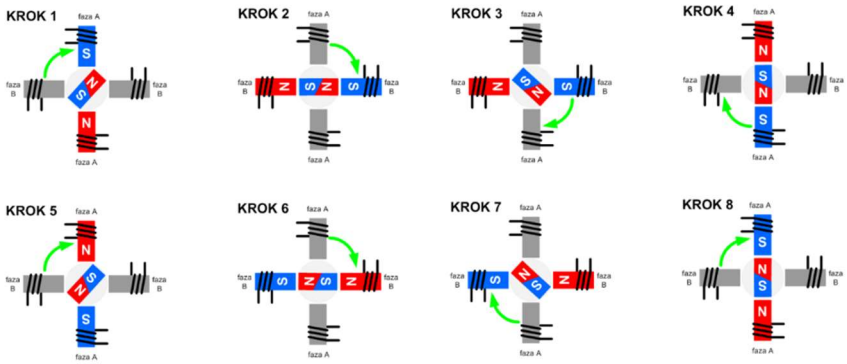

Image 9: Two pole rotor-four coil stator functioning diagram [16].

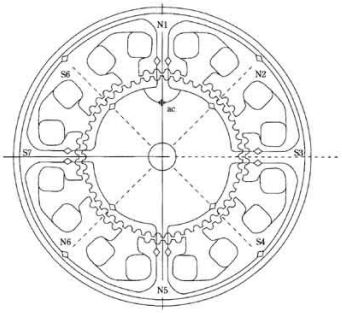

Image 10: Representation of a 100 toothed rotor and stator (the number of phases does not correspond with the project's one) [15].

To create the 400 Hz pulse wave, as the function in charge of creating it could only be called from the main program from millisecond to millisecond, it was only called two out of five times, getting this way globally a 2.5 ms period. This solution solves the wave generation problem, but it may provoke the apparition of some others such as vibration.

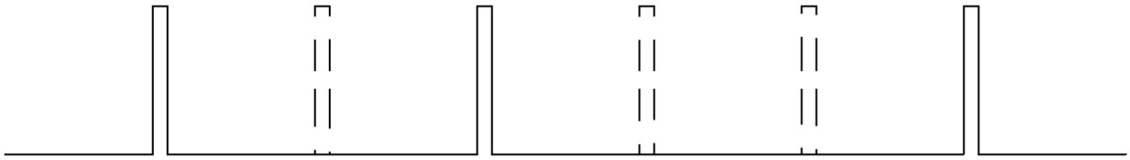

Figure 5: 400 Hz pulse wave construction.

5.3. Drivers

As a need of having external access to the motors, there are included two drivers which link them either to the system's motherboard or directly to the control connector. It is configured with inverse logic, this is, when it receives a '0' it is understood as '*true*' and when it receives a '1' it is understood as '*false*'.

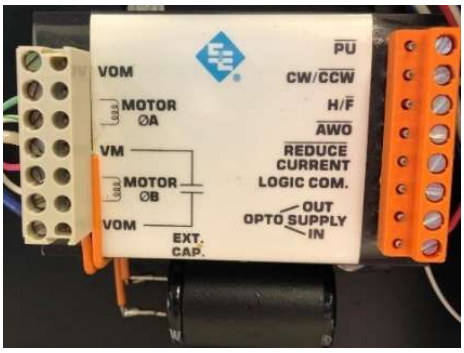

Image 11: Driver.

On the one hand, the left terminals of the image are the ones **connected to the motor**. There exists a voltage difference between Vm and Vom (mass of the system) of **32 V** provided by the power supply. This voltage is the one necessary to **excite the coils** and to generate a **magnetic field** capable of making the rotor to start rotating. However, to make it work, it is also needed a **pulse wave** to make this voltage oscillate between 0 and 32 V and to, this way, excite consecutively coils A and B to make the rotor rotate.

On the other hand, the right terminals are more related with the configuration and, some of them, are connected to the control connector. The following information has been obtained from the driver's datasheet: [17]

- **$\overline{\text{PU}}$:** it is the pin through which it will be introduced the **pulse wave**. As it has been said in the last paragraph, it will be the one in charge of introducing the logic of when must each of the coils be excited. It goes negated because the driver has inverse logic. There is included a **4700 µF capacitor** to filter the signal and avoid misunderstandings in the pulse wave. **It must be always included**.
- **CW/$\overline{\text{CCW}}$:** it tells the **direction of rotation of the motor**. If it receives a '0' it rotates counterclockwise, whereas if it receives a '1' it rotates clockwise.
- **H/$\overline{\text{F}}$:** it defines if the stepper takes **half or full steps**. This pin is not linked to the control connector so there is no possibility of modifying it. It is set as a '1' by default, so it takes half steps instead of full, what makes the plate movement smoother.
- **$\overline{\text{AWO}}$:** "all windings off". When the motor is not receiving pulses and it remains still and it cannot be moved because there is a **holding torque**. If this pin is on a low level, it deactivates this torque, and the plate can be moved manually; but if it is on a high level, it maintains this holding torque.
- **$\overline{\text{REDUCE CURRENT}}$:** it is used to **adjust the current supply** to the motor. In can be configured in two ways: first, in a 0/1 logic – when receiving a '0' it provides 1.0 A, whereas when receiving a '1' it provides 3.8 A; second, if there is introduced a potentiometer, this current can be regulated.
- **LOGIC COMMON**: it is internally joint to Vom. It is the **mass of the circuit**.
- **OPTO SUPPLY OUT**: it enables the user the use of an **internal power supply**. There is no external access to this pin.
- **OPTO SUPPLY IN**: connection for **opto-isolator** power supply. No external access.

5.4. Motherboard

The motherboard's main objectives are **directing the plate** in each of the four possible directions (clockwise or counterclockwise rotation for each axis) and to **indicate when does the plate arrive to one of the limits**, this is, detecting when the end switch, located in the direction that the plate was moving, gets closed. Therefore, to accomplish these goals, there is included a **74LS00 device** which is based on a **NAND logic** and whose connections can be observed in the figure below (Figure 6). From all the 14 pins there will only be external access to **DIR1 and DIR2** as direction inputs, and **SW OUT 1 and SW OUT 2** as end switch outputs, which currently are on a **high-level state** (plus the GND terminal that will be common to the whole system).

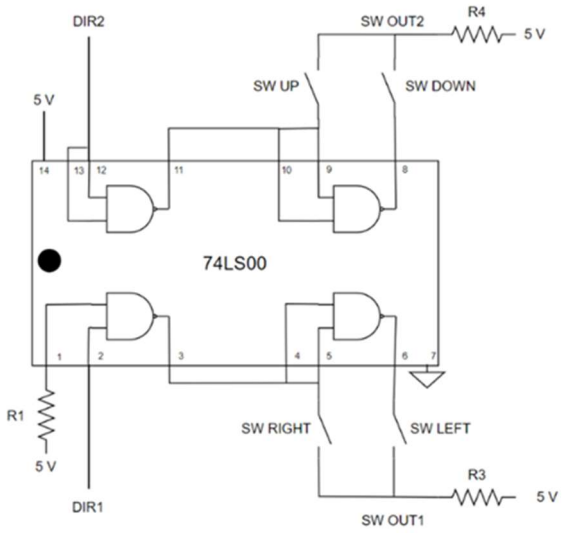

Figure 6: 74LS00 connection.

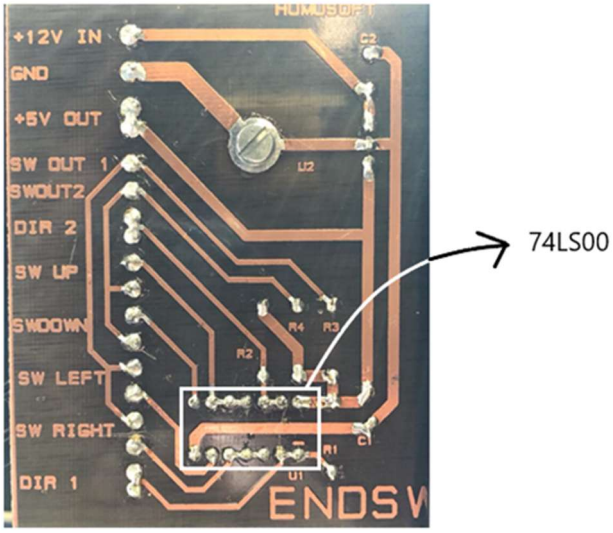

Image 12: System's motherboard.

To fully understand the aforementioned diagram, it is necessary to determine the reference system that has been employed to know what it is referred to when speaking about directions. These assignments have been made with respect to the side where the control and power connections are emplaced.

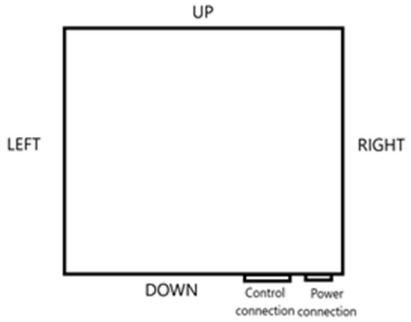

Figure 7: Reference system of the table.

As it can be seen in Figure 6, the only difference between the upper and lower half is the axis with respect to which the action is applied. Therefore, to avoid repetitions, the analysis will only be made for one of them, the lower one.

5.4.1. 74LS00 connection analysis

First, the direction introduced through **DIR1** is compared to a **high state value** ('1'). The logic followed in the direction terminal is that if **DIR1=0**, the plate rotates towards the **left** side, whereas if **DIR1=1**, it goes to the **right** one. As the gate employed is a NAND, the exit value will be the result of a negated boolean multiplication between both entries, and since one of them is always on a high level, the exit will be a negation of the direction entry.

Then, this exit is **short-circuited** with both entries of the second NAND gate and shunted to the right-side end switch, whereas the exit of this gate is linked to the left contact. This makes the functioning of the contacts to **depend on the direction selected**. For example, the left contact will only work if the plate is rotating towards the left side, and so it will do for the right one. For this reason, despite there is not a pin for each of the switches but only one (SW OUT 1), it can be identified which of the four contacts has been closed with only one output.

As an **example**, let us suppose that it is wanted to rotate the plate towards the **left side** until reaching the end switch. It would be introduced a **'0' in DIR1** - to tell the motor that it must move the plate to the left – and it will make a **'1'** to appear in the exit of the first gate and in the inputs of the second one. In terms of voltage, a **'1' is equal to 5 V** so, regardless of whether the right contact is closed or not, there will not be any change in voltage and SW OUT 1 will **remain on a high level**. Here it can be seen that **the switches working directly depends on the direction** in which the plate is moving.

However, after going through the second gate, it would appear a '0' (0 V) on the output of it, which means that, if the left contact gets closed, it will appear a voltage drop that would **provoke a change of state in SW OUT 1 from '1' to '0'**. With this change of state, it is notified to the controller that the plate cannot rotate more in that direction and it will order the motor to **stop**. To avoid creating a short-circuit when this voltage drop happens, it is included the **4.7kΩ R3 resistance**.

To sum up the explanation, there are included two logic tables containing as inputs all the possible **directions** and **end contact** values, and **SW OUT** as an output.

DIR1	SW RIGHT	SW LEFT	SW OUT 1
0	Opened	Opened	1
0	**Opened**	**Closed**	**0**
0	Closed	Opened	1 (not possible)
0	Closed	Closed	0 (not possible)
1	Opened	Opened	1
1	Opened	Closed	1 (not possible)
1	**Closed**	**Opened**	**0**
1	Closed	Closed	0 (not possible)

Table 1: Logic table 1 for directions and switches of the first axis.

DIR2	SW UP	SW DOWN	SW OUT 2
0	Opened	Opened	1
0	**Opened**	**Closed**	**0**
0	Closed	Opened	1 (not possible)
0	Closed	Closed	0 (not possible)
1	Opened	Opened	1
1	Opened	Closed	1 (not possible)
1	**Closed**	**Opened**	**0**
1	Closed	Closed	0 (not possible)

Table 2: Logic table 2 for directions and switches of the second axis.

The cases where "not possible" appears are referred to the ones where a contact from one side is closed when the plate is rotating towards the opposite one. It is not possible for this to happen because the plate is the only component which is able of opening and closing the contacts.

5.4.2. 5 V signal obtention

For the system to work properly it is necessary to power the 74LS00 with 5 V. To obtain them, it is included in the motherboard a 7805 device a voltage regulator that reduces a 12 V input voltage to a 5 V one. Normally, there are added a pair of capacitors in parallel to filter the signal and reduce noise.

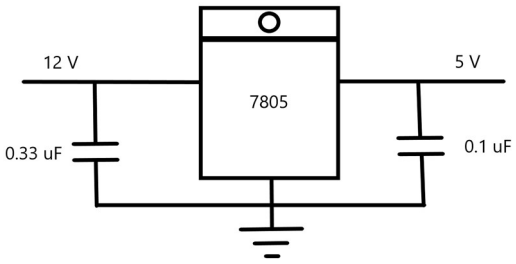

Figure 8: 7805 voltage regulation circuit.

Originally, these 12 V were supplied by the PC and were used to power the camera and the system's circuit. Nevertheless, since this camera was no longer going to be employed, it was necessary to supply this voltage in some other way. After having a look to the system's manual [14], it was found that **one of the pins of the power supply connector** was at a **32 V** voltage (voltage required for the motors to work).

Therefore, this voltage could be employed and brought to the input of the 7805 device after being previously reduced to 12 V by using a **pair of voltage regulators**. There were needed two of them because with only one, due to the 20 V voltage drop, the **temperature** of the device turned to be **excessively high**, and it was convenient to reduce it.

This temperature increment could be quantified having a look to the **7812 datasheet**, where it was found that this device had a **TO-220 encapsulation** meaning that its thermal coefficient was of **20 °C/W**. Knowing the voltage and the current it is easily obtained the **power dissipated** (20V · 71mA = 1.42W) and, with it, the **increment in temperature** (1.42W · 20°C/W = 28.4 °C).

For this reason, instead of using only one to reduce from 32 V to 12 V, it was used a pair of them: one from **32 V to 24 V** (**7824 device**) and another from **24 to 12 V** (**7812 device**) with a silicon heatsink joint to them to reduce the increment in temperature.

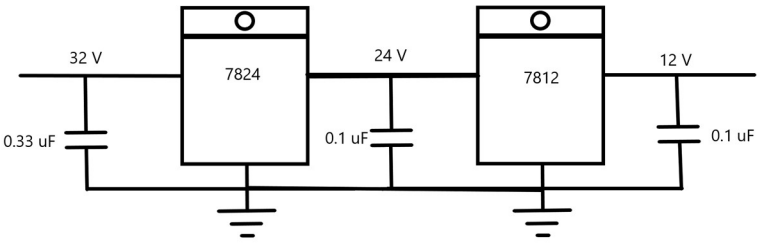

Figure 9: 7824 and 7812 voltage regulation circuits.

This circuit was built in a **PCB** taking as input of the circuit the **VM pin (32 V)** of one of the drivers and welding the output and the mass of the circuit to the **12 V** and **GND pins** of the motherboard, respectively. This way it could be obtained the voltage needed **without adding external components**, getting a much more compact appearance and increasing its ease of use.

5.4.3. Control connector pins

Once the logic that the table follows has been defined, the only remaining point is the **identification of each of the control connector pins and the definition of the signals related to them**. In other words, to find out he relation between the system signals and correspondent control connector pins. From all the 25 available pins there are **only used eight of them** and they are connected in the way described in the below diagram (Figure 10).

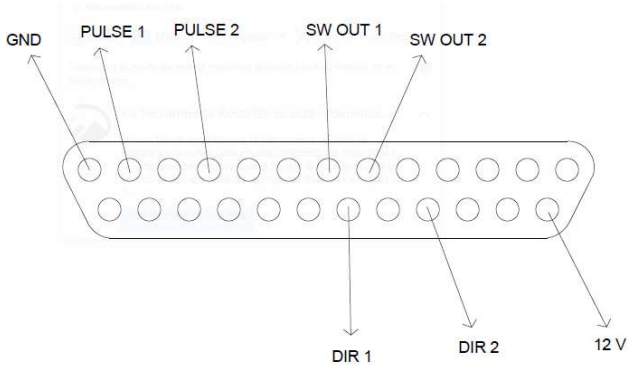

Figure 10: 25 pin control connector description

From the eight signals, four of them are external inputs (DIR1, DIR2, PULSE1, and PULSE2), two are outputs (SWOUT1 and SWOUT2) and the two remaining are the ones related with ground and the 12 V DC voltage.

- **DIR:** they are the terminals through which it will be indicated the direction of rotation in each of the axes.

    - **DIR1 (pin 20)**: if it is introduced a '0' or a low level, the plate will rotate towards the left side, whereas if it is introduced a '1' it will rotate towards the right one.
    - **DIR2 (pin 22)**: if it is introduced a '0' or a low level, the plate will rotate towards the upper side, whereas if it is introduced a '1' it will rotate towards the lower one.

- **PULSE 1 and 2 (pins 2 and 4):** they are the pins through which the pulse wave, necessary for the motors to work, will be introduced. Depending on the period of this wave, the speed of rotation will be higher or lower. According to the system's manual, the maximum frequency recommended for this signal is **400Hz** (2.5 milliseconds period) for both PULSE1 and PULSE2, so it has been the one selected. Apart from that, as it is wanted to spend as little time as possible generating the pulses, it was searched the minimum width that the system could understand as a pulse. For this purpose, a PWM wave with a switching frequency fixed to 400 Hz was introduced, and the duty cycle was reduced until it reached a point where the table stopped working because it was no longer able to recognize the pulses. The duty cycle correspondent to this point was between the 4 and 5% (10 µs – 12.5 µs pulse width), so it was decided to fix this width to 12 µs.

- **SW OUT 1 and 2 (pins 7 and 8):** these pins will notify the controller when the plate arrives to one of the end switches. In normal conditions, they will be at 5V ('1' in digital logic) and they will turn to a 0 V voltage when closing one of the correspondent contacts.

5.4.4. Power connector pins

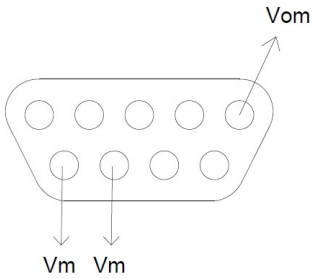

Figure 11: 9 pin power connector description.

The power connector has 9 pins from which most of them remain unused. Originally, the system also employed the second and the fourth pin to supply a 12 V voltage to the camera, but since that camera is no longer being used, they have become useless.

- **Vm (pins 6 and 7):** these pins are at a 32 V voltage and they supply the necessary voltage for both motors to work (one for each). Moreover, as it can be seen in the ***5.4.2 5 V signal obtention*** chapter, they are used to obtain the 12 V that are employed in the motherboard.
- **Vom (pin 5):** it is the ground of the power supply and of the whole system as they are all connected.

5.5. Arduino DUE

As it has been said before, the device selected for the task of controlling the system has been the Arduino DUE. This microcontroller board is based on the Atmel SAM3X8E ARM Cortex-M3 CPU and it is a very appropriate controller to carry-on real-time tasks. [18]

This microcontroller has a special characteristic that is its operating voltage. Instead of being 5 V as most of the Arduino microcontrollers, it is 3.3 V. Given that the table's output pins (SW OUT 1 and SW OUT 2) are at a 5 V voltage, it was needed to introduce a voltage divider to reduce it to 3.3 V when connecting them to the microcontroller. This task has a major importance because if this voltage is not reduced, the chances of irreversibly damaging the Arduino are very high.

To lower it, a 10 kΩ resistor joint to another 20 kΩ one was included, so that the current was not excessively high, and it could not damage the system.

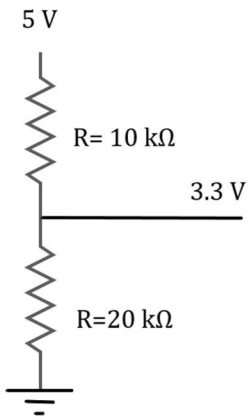

Figure 12: Voltage divider circuit (5V-3.3V).

From all the inputs/output pins that it offers, they have only been used the digital ones because all the signals involved were only digital. Below, they are shown the pins that have been used to connect the table to the Arduino.

- Pin 46: PULSE 1.
- Pin 47: PULSE 2.
- Pin 48: SW OUT 1.
- Pin 49: SW OUT 2.
- Pin 50: DIR 1.
- Pin 51: DIR 2.

For its power supply it is employed the **Arduino's programmable port** connected to the computer. It was wanted to use the 12 V existent between pins 25 and GND of the control connector as input voltage of the microcontroller (it admits voltages between 7 and 12 V as input voltages), but it was observed that if the table remained stopped for a certain time, the voltage in these pins suddenly dropped to around 5 V and the table started having an unexpected behaviour. An attempt was made to discover the reason why this happened, without reaching any result, so it was decided to use the programmable port as power supply and lay aside the other option.

5.6. Camera

The device employed as a **camera** is a Huawei smartphone with an **Android operating** system. There has been developed an application capable of detecting the ball on the plate thanks to the contrast in colours between them (white ball vs. black plate). It is important to try to avoid light bulbs or solar light because, in case of appearing any kind of reflection on the plate, it could not be able of detecting the ball.

The sampling period has been set to **33 ms**, slightly below the one of the original camera (40 ms), but this difference is not enough to make problems related with this question to appear. It also has a **256 x 256 pixels resolution**, which is equal to the initial one. Nevertheless, in case of looking for a better resolution, it could be increased, but for the goal of this project it is enough.

The camera sends every 33 ms the information of the **accumulated number of frames** and the **coordinates in pixels** of the last sample taken from the ball's position. This sample, after being applied a certain relationship between pixel and distance units, is translated to the distance in meters in both axes with respect to the reference system of the plate so that it can be more understandable by the user.

6. Initialization of components

6.1. Table's reference position

First, it is needed to define a **reference position** with respect to which the rotations of the plate will be made. This position will be the one at which the plate is **completely horizontal**, that is, the one at which when placing the ball on the table it remains **still**.

The **regulation** will be first done for one of the axes and then for the other. In the first place and **regardless of where the plate is initially**, it is selected a direction of rotation **(CCW)** and it is indicated to the table to start moving until reaching the correspondent **end switch**. Once arrived, the direction is changed, and it starts rotating towards the opposite switch until **reaching it**. During this second path, it is stored in an auxiliar variable the **number of steps** that the motor has taken until getting to close the contact so that it is known how many steps are there necessary to go **from one side of the table to the other**. Knowing this, it is trivial to find the point at which the plate will remain horizontal. It is just needed to cover **half of the steps** taken to reach this point. After completing the process in this axis, it is repeated for the other one, making the plate to finally end in the desired position**.**

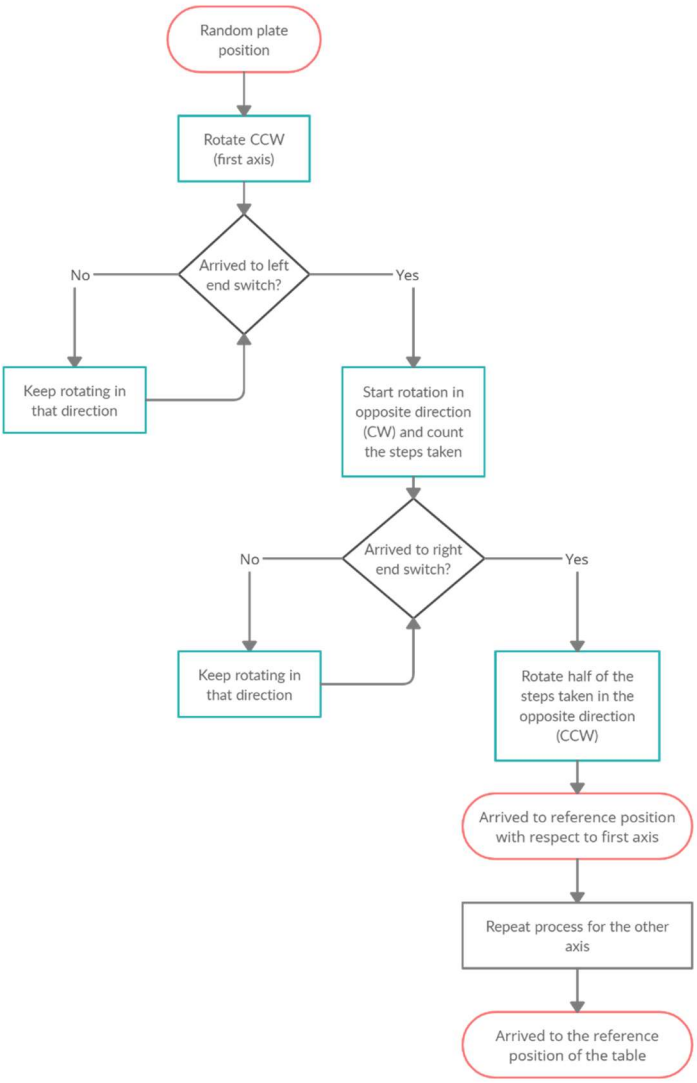

Figure 13: Plate's reference position obtention flowchart.

6.2. Ball's position relationship initialization

The camera sends a string every 33 ms indicating where the ball was the last time it was seen. This string contains the **number of frames** that have been taken since the camera started to take samples and the **ball's coordinates** in pixel units. The importance of knowing the number of frames resides in being capable of determining if any sample has been **lost**. If this value had not be included, it would not be possible to know this.

But in sights of carrying on the control task, the **ball's position** is the value that matters the most. This value is provided in **pixels** so it must be found a way of modifying it to have it as a distance in **metres** with respect to the centre of the plate, which will be taken as reference system.

The main difficulty in defining the relation between the value of the sample in pixels and in centimetres is the **low stability of the camera's mount**. The mount selected was not designed for this purpose, so it is very unstable and, in case of needing to remove the smartphone from it, it is almost impossible to place it again in the same position that it was before removing it. For this reason, it is necessary carry on a calibration every time the system is initialized to know the pixel coordinates of each of the plate edges.

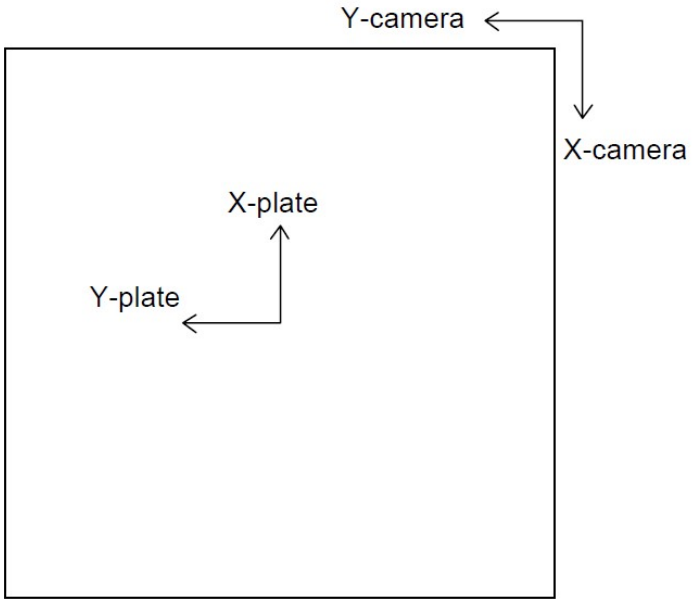

Figure 14: Camera's and plate's reference systems.

The way of getting to do this **without needing to calibrate the camera manually** each time is the following one: first, and after having reach the reference position of the table, it is said to the plate to rotate an angle of **-5°** in both axes, making the ball to move to the **bottom left corner**. Once it has arrived, **samples** of its position are taken until **4 seconds** have passed since the plate moved to that angle. Then, from all the samples, it is only taken the **last** of them and it is stored in a variable.

Afterwards, it is said to the plate to rotate **10°** in the opposite direction in both axes, arriving to a **+5° angle position**, which makes the ball move to the **upper right corner**. Once arrived, it is repeated the sampling process and the **last** of them is stored in another variable. This way, it is known the position in pixels of the ball in both corners and, knowing that in metres these points are (-18 cm, -18 cm) and (18 cm, 18 cm), it can be obtained the relation between the sample in pixels and centimetres. When this task is done, it is told to the plate to go back to its **reference position**.

Nevertheless, the data sent by the camera is a string type and has the following structure:

$$(number\ of\ frames, \quad y\ coordinate, \quad x\ coordinate)$$

Equation 12: Camera's data structure.

To know when does a string finish, it will be searched the "**CR**" bytecode of **carriage return**. This bytecode is coded as 0x0A, which is a 10 in decimal, and it is sent to mark the **ending of a string**. There are other ways of indicating this end (LF o CR/LF) but in this case, CR has been the mode employed. [19] When this byte arrives, it is known that the **string has finished**.

Thus, to make this data usable, it is necessary to convert this string to **six float variables** (three for each corner). To fulfil this task what it will be done is to read the strings and, as the values are separated by ***commas***, look for them so that each of the values can be differentiated and stored in different integer variables. However, this is an operating mode that will work for the **first two values** but not for the third and last of them because there is not a *comma* after it. For this last value, it will not be necessary to look for any character because the string was already **cut before with the CR search**, so when arriving to the end it will stop and store the last value read.

However, the camera will not only send the position of the ball in each moment. There are some scenarios in which instead of a string it will send some data that it is **not desired** to be processed:

- **CE151:** every time that the system is initialized it is sent this starting message, so whenever a string starts with a "C" it must know that it is due to the initialization and that it must not take it.

- **Partial string**: the first sample may not start right in the beginning but at a certain position of the string, which means that, instead of sending a complete sample, it would only send a partial one. Therefore, the first one should never be analysed and always dismissed.

- **-2**: it is unusual that this happens because the camera's height is fixed, but if the camera observes the ball bigger than a certain size it will send a -2. This notification could also be helpful to detect reflections on the plate because for the camera they are seen as big-sized white stains and if any of them interfere with the ball detection, it will be notified.

- **-3**: it will be returned in case of not detecting the ball.

For this reason, there will be included some conditions that must be fulfilled to process the data received, **avoiding all these possible errors**. These processes are shown in the following flowcharts to have a more visual representation of them.

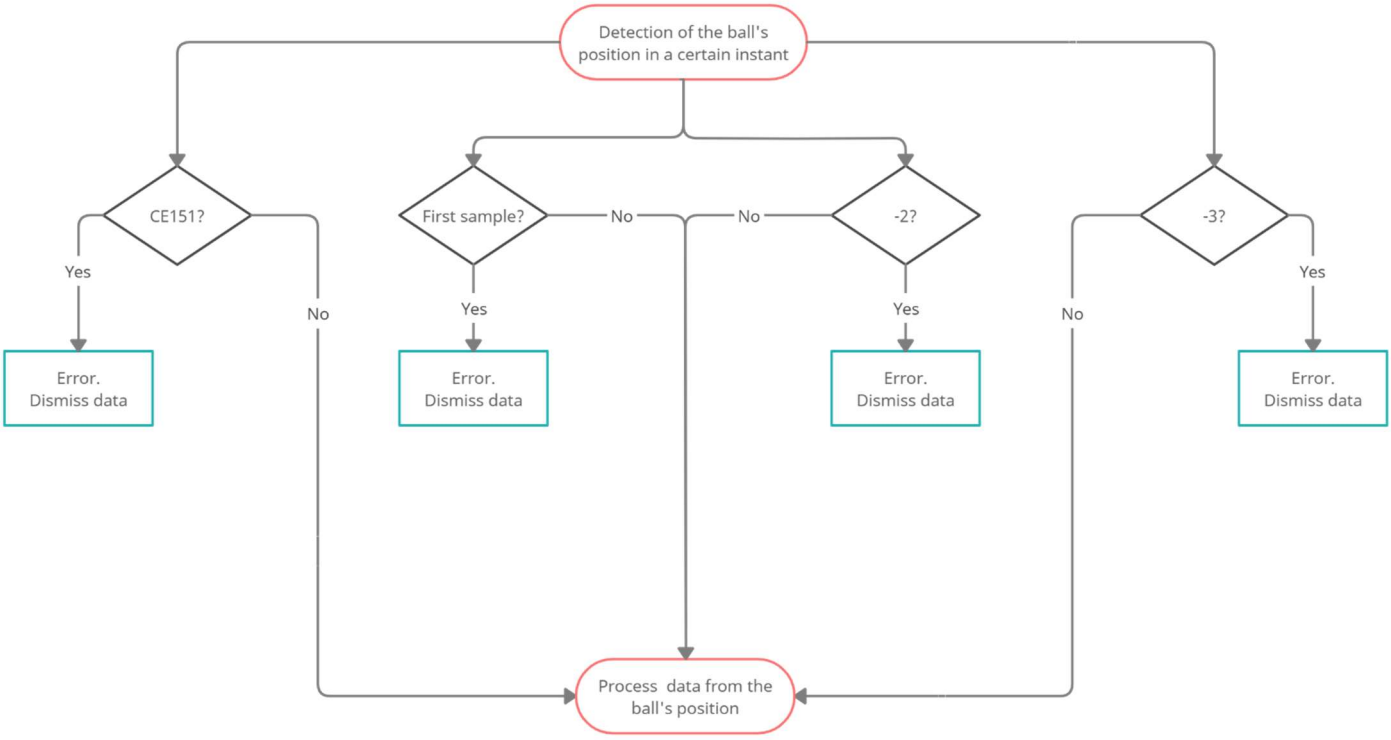

Figure 15: Ball position processing decision flowchart.

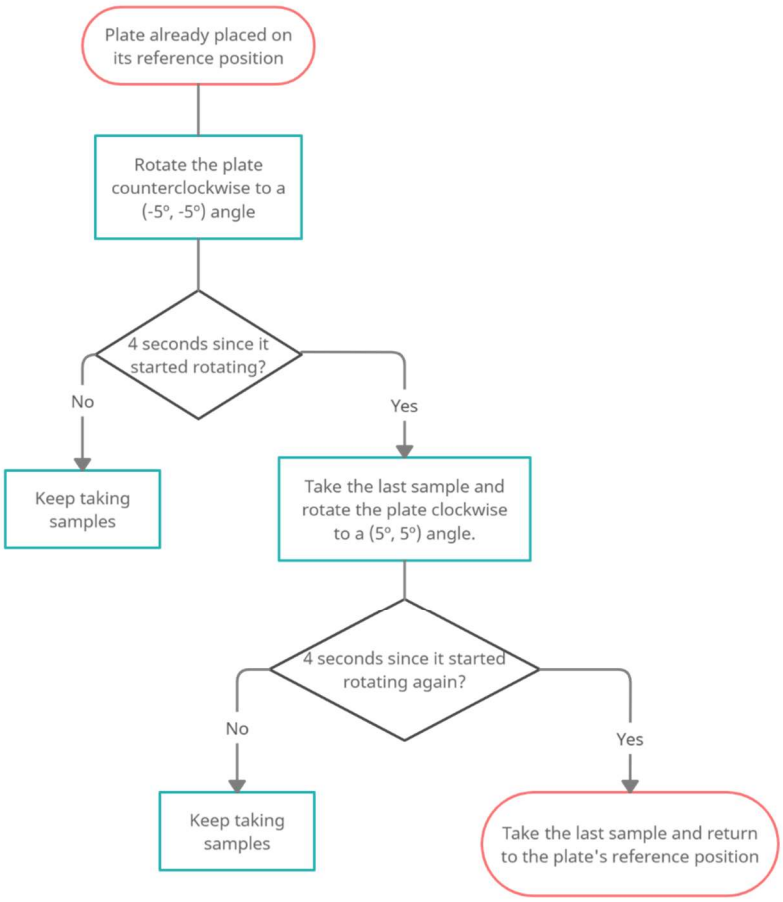

Figure 16: Flowchart of the process carried on to define the relation between the position of the ball in pixels and in centimetres.

The ball positions in the upper and lower corner, after applying the conversion from string to float, are stored in four variables called **xpos1, xpos2, ypos1 and ypos2**, but to make them usable, it is necessary to convert them **from pixels to centimetres**. As this relation is **linear**, they can be written as the following equation system for each of the coordinates, where **'1'** corresponds to the sample of the position of the ball when the plate is on a (-5°, -5°) angle and '2' to the one when the plate is on a (5°, 5°) angle:

- **X-coordinate:**

$$x1\ [cm] = mx \cdot x[pixels] + nx \quad \rightarrow \quad x1\ [cm] = mx \cdot xpos1 + nx$$

$$x2\ [cm] = mx \cdot x[pixels] + nx \quad \rightarrow \quad x2\ [cm] = mx \cdot xpos2 + nx$$

- **Y-coordinate**

$$y1\ [cm] = my \cdot y\ [pixels] + ny \quad \rightarrow \quad y1\ [cm] = my \cdot ypos1 + ny$$

$$y2\ [cm] = my \cdot y\ [pixels] + ny \quad \rightarrow \quad y2\ [cm] = my \cdot ypos2 + ny$$

Solving this equation system, it is obtained that:

$$nx = -18 - \frac{36}{xpos2 - xpos1} \cdot xpos1$$

$$mx = \frac{36}{xpos2 - xpos1}$$

$$ny = 18 + \frac{36}{ypos2 - ypos1} \cdot ypos1$$

$$my = -\frac{36}{ypos2 - ypos1}$$

Substituting in the initial expression it is obtained that the relation between pixels and centimetres is:

$$x\ [cm] = \frac{36}{xpos2 - xpos1} \cdot x\ [pixels] - 18 * \frac{xpos1 + xpos2}{xpos2 - xpos1}$$

$$y\ [cm] = \frac{-36}{(ypos2 - ypos1)} \cdot y\ [pixels] + 18 * \frac{ypos1 + ypos2}{ypos2 - ypos1}$$

Equation 13: Relation between the ball's position in pixels and in centimetres.

7. Controller design

Once the table is already fully initialized, the **control task** can be carried out. The objective is to **move the ball** to a series of specific points of the plate trying to model a **trajectory**. To do it, the system's position reference will be changing from time to time to the different points that it is wanted to move the ball. This time between changes must be **sufficient** for the ball to arrive to each position and stabilize there before moving to the next one.

7.1. Open-loop plate control

First, before introducing the controller it is advisable to start with an **open-loop control** of the system, this is, being able of moving the plate to a desired angle or to make it describe a certain rotation trajectory. The process is the same that the one depicted in the table's reference position obtention with the only difference that now, instead of rotating until reaching a switch, the **angle rotation** will be marked **in advance** and, from it, it will be deduced the **number of steps** that are needed to take to reach that position. To do this conversion it is employed a relationship given by the system's user's manual [14] which relates the **motor step size** with the **angle rotated per each step** ($\Delta\alpha$=0.03598 °/step).

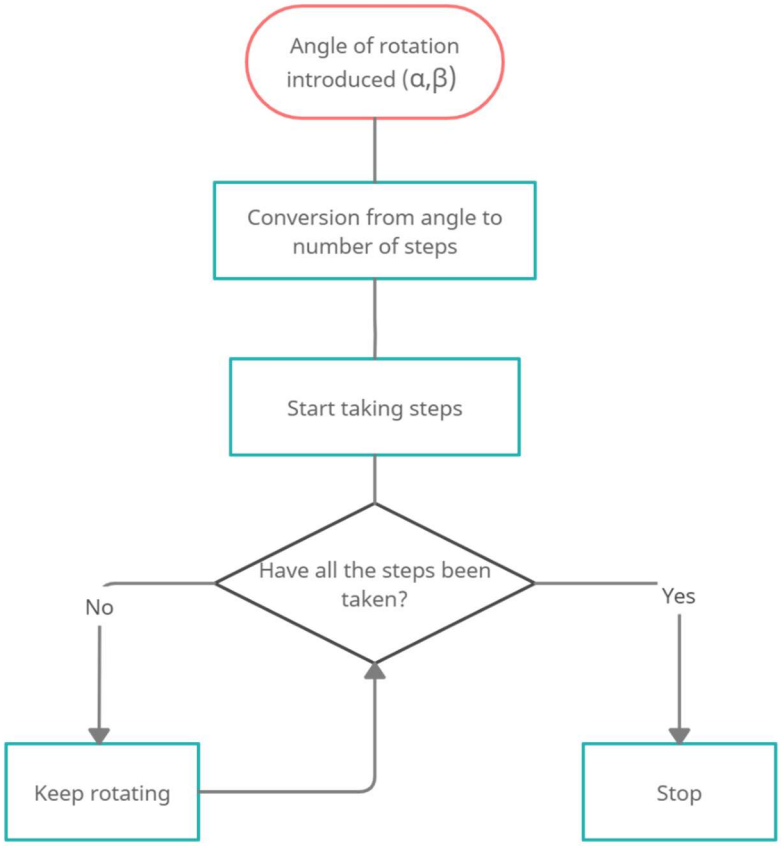

Figure 17: Open-loop control flowchart.

7.2. Controller design

The above approach can be used if it is only wished to control the table's position independently of the ball's position, as the **system plant is unstable**. But as the goal of this system is to define a way to be able to **control the ball's trajectory**, it will be necessary to close the loop and **define a controller** capable of controlling the ball. Apart from that, it is important to highlight that, to carry out all the controller simulations, it has been used the **Scliab environment**.

7.2.1. Plant discretization

Before starting with the controller design, it is mandatory to discretize the continuous plant obtained before and **convert it to the Z-domain**. To do it, instead of carrying on calculations by hand, it has been used the Scliab LTI library which has a specific function that executes the Z-transform *(lti_toDiscrete(function, sample period))* for a certain sample period. In this case, as it has already been said, it has been selected a 33 milliseconds one.

$$Gm(z) = 0.000246193 \cdot \frac{(z + 0.256214) \cdot (z + 3.57341)}{(z-1)^2 \cdot (z - 0.838223)}$$

Equation 14: Plant's discretized transfer function.

7.2.2. Controller obtention using the pole assignment method.

To define this controller, different options were shuffled. Initially, it was thought to use the same PID controller as the one used in the user's manual but modifying certain parameters just to adjust it to the actual system. However, it turned to be more complex and inefficient than what it was first thought, and it was decided to change of idea. Then, it was thought of using the **root locus designing method,** but the controller obtained did not fulfil the requirements as it should. Hence, in the end, it was chosen the **pole assignment designing method** as it enables placing the poles where it is the most convenient to meet the specifications [20]. Regarding the project, the only **specification** that has been imposed is to have a **null position error**, but to accomplish this task it is not needed to apply any extra condition because the plant already has two integers that make this error to be zero. The block diagram of the system can be represented as follows:

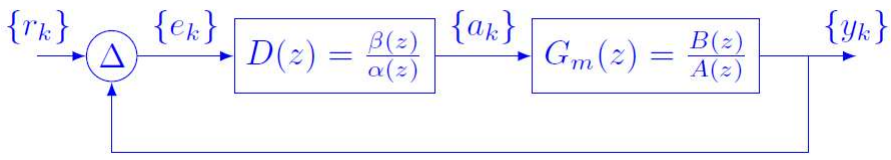

Figure 18: Control structure for pole assignment designing method [20].

Where B(z) and β(z) are the open loop zeros whereas A(z) and α(z) are the poles.

$$B(z) = 0.000246193 \cdot (z + 0.256214) \cdot (z + 3.57341)$$

$$A(z) = (z-1)^2 \cdot (z - 0.838223)$$

To make the calculations easier, the position of the poles and zeros as well as the plant's gain will be substituted by a series of variables.

- K = 0.000246193
- C1 =−0.256214
- C2 =−3.57341
- P = 0.838223

Rewriting the plant's expression using these new variables it results:

$$Gm(z) = K \cdot \frac{(z - C1) \cdot (z - C2)}{(z-1)^2 \cdot (z - P)}$$

It is also convenient to, beforehand, write the expressions of the **controller D(z)** and the **closed loop poles γ(z)** as a function of B(z), β(z), A(z) and α(z) to clear out what it is it going to be pursued.

$$D(z) = \frac{\beta(z)}{\alpha(z)} = \frac{\beta_0 \cdot z^{n_\beta} + \beta_1 \cdot z^{n_\beta - 1} + \beta_2 \cdot z^{n_\beta - 2} + \cdots + \beta_{n_\beta}}{\alpha_0 \cdot z^{n_\alpha} + \alpha_1 \cdot z^{n_\alpha - 1} + \alpha_2 \cdot z^{n_\alpha - 2} + \alpha_{n_\alpha}}$$

$$\gamma(z) = A(z) \cdot \alpha(z) + B(z) \cdot \beta(z) = z^{n_\gamma} + \gamma_1 \cdot z^{n_\gamma - 1} + \gamma_2 \cdot z^{n_\gamma - 2} + \cdots + \gamma_{n_\gamma}$$

Once all these expressions have been defined, the design can be started. First, it is necessary to define the degree of these polynomials. The number closed loop poles, as it has been said before, is equal to the sum of the number of poles of the plant plus the controller ones, so the degree of this polynomial will directly depend on these two.

$$n_\gamma = n_A + n_\alpha$$

The controller, in order **not to have any delays**, must have the **same number of poles than zeros** (if it had more poles than zeros, it would introduce an undesired delay in the control).

$$n_\alpha = n_\beta$$

Last, it must also fulfil a condition related with the number of requirements '*c*' that have been imposed to the system, such as damping, settling time, overshoot… this condition is directly related with the number of closed loop poles and it is mandatory to be accomplished for the assignation problem to have solution.

$$n_\alpha = n_A - 1 + c$$

Getting together the second and third equation, it results the following equation system:

$$n_\gamma = n_A + n_\alpha$$

$$n_\alpha = n_\beta = n_A - 1 + c$$

Equation 15: Pole assignment conditions.

From that system, knowing that the number of conditions is equal to 0 and the number of poles of the plant is equal to 3 it is trivial to know which will be the degree of each of the polynomials.

$$n_\alpha = n_\beta = 2$$

$$n_\gamma = 5$$

Knowing this coefficients, D(z) and γ(z) can be again reformulated in a more compact and defined style.

$$D(z) = \frac{\beta_0 \cdot z^2 + \beta_1 \cdot z + \beta_2}{\alpha_0 \cdot z^2 + \alpha_1 \cdot z + \alpha_2}$$

$$\gamma(z) = z^5 + \gamma_1 \cdot z^4 + \gamma_2 \cdot z^3 + \gamma_3 \cdot z^2 + \gamma_4 \cdot z + \gamma_5$$

The *P* pole of the system's plant (P = 0.838223) might be cancelled to make the calculations easier because it is a non-hazardous pole that will not provoke the apparition of non-desired behaviours on the loop. By cancelling it, it is removed one unknown of the problem, getting more simplified. Instead of it, it will be placed a pole in 0.927 that will not worsen the system and, moreover, will make the system to stabilise in 2 seconds, which is enough time for the ball to arrive and stabilise on a certain point. It will be represented as *zp*.

Furthermore, the closed loop polynomial can also be simplified by adding the pole that will be cancelled (*P)*, the pole that will be added at 0.9 (*zp*) and another three dominated poles (*zd*) which, in order to be dominated by the other one, must satisfy the following condition.

$$|zp|^5 > zd \;\rightarrow\; |0.927|^5 = 0.659 > zd$$

For this reason, the three last poles will be placed in z= - 0.6, z= - 0.61 and z=- 0.62 to make them the slowest possible while still being dominated by *zp*. After carrying on all these calculations, they present the following form:

$$D(z) = \frac{(\beta_0 \cdot z + \beta_1 \cdot z)(z - P)}{\alpha_0 \cdot z^2 + \alpha_1 \cdot z + \alpha_2}$$

$$\gamma(z) = (z - zp)(z - P)(z + 0.6)(z + 0.61)(z + 0.62)$$

Recalling the above expression $\gamma(z) = A(z) \cdot \alpha(z) + B(z) \cdot \beta(z)$ it can be formed an equation system to obtain the five remaining unknown parameters. For this task, it will be used the Scilab program, and the problem will be solved by means of converting it to a matrixial system. It is important to highlight that, as it was expected, the *P* pole will be cancelled.

$$[(z-1)^2 \cdot (z-P)] \cdot [\alpha_0 \cdot z^2 + \alpha_1 \cdot z + \alpha_2] + K \cdot [(z - C1) \cdot (z - C2)] \cdot [(\beta_0 \cdot z + \beta_1 \cdot z)(z - P)] =$$

$$= (z - zp)(z - P)(z + 0.6)(z + 0.61)(z + 0.62) = 0.21033 - 1.26153z + 2.8124z^2 - 2.7569z^3 + z^4$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 1 & 0 \\ 1 & -2 & 1 & -(c1+c2) & 1 \\ 0 & 1 & -2 & c1 \cdot c2 & -(c1+c2) \\ 0 & 0 & 1 & 0 & c1 \cdot c2 \end{bmatrix} \cdot \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ K\beta_0 \\ K\beta_1 \end{bmatrix} = \begin{bmatrix} 1 \\ -2.7569 \\ 2.8124 \\ -1.26153 \\ 0.21033 \end{bmatrix}$$

After solving this system, the coefficients are obtained:

$$\alpha_0 = 1.$$

$$\alpha_1 = -0.7722657$$

$$\alpha_2 = 0.2237037$$

$$K\beta_0 = 0.0153568$$

$$K\beta_1 = -0.0146026$$

Substituting the terms in the controller it is obtained its final expression.

$$\mathbf{D(z) = 62.3772 \cdot \frac{(z - 0.95089) \cdot (z - 0.838223)}{(z\text{^}2 - 0.772266z + 0.223704)}}$$

Equation 16: Controller expression.

Now, having the controller already designed, it can be tested how much does it fit in the system by doing some closed loop simulations. To do it, first it is necessary to obtain the system's closed loop expression.

$$\boldsymbol{T(z) = \frac{D(z) \cdot Gm(z)}{1 + D(z) \cdot Gm(z)} = 0.0153568 \frac{(z - 0.9589) * (z + 0.256214) \cdot (z + 3.57341)}{(z - 0.926909) \cdot (z - 0.62) \cdot (z - 0.61) \cdot (z - 0.6)}}$$

Equation 17: Closed loop system expression.

Before starting to simulate, it is convenient to convert this expression back to the Laplace domain using a zero-order holder to observe not only the samples but all the points between them. In the following diagrams it is shown the response of the system when introducing a step input.

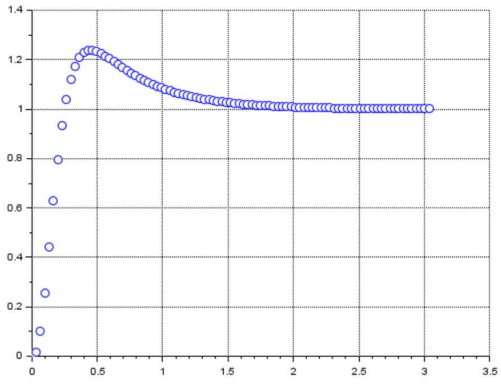

Figure 19: System's discrete step response.

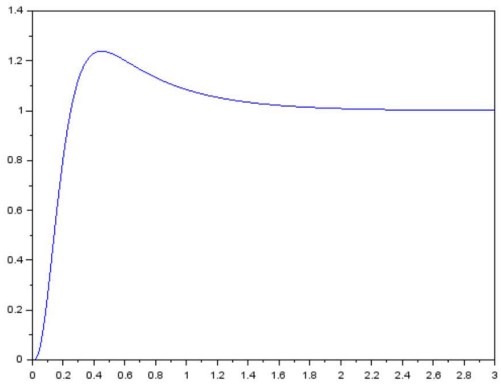

Figure 20: System's continuous step response.

It is observed that the settling time is in line with the previously established one as it starts following the reference after 2 seconds has passed. It can also be seen that a certain overshoot appears that will cause the ball to pass the point where it should go and, afterwards, return to it. This behaviour could be acceptable as the overshoot is not excessively high, but when having a look to the control action it is seen that it cannot be acceptable.

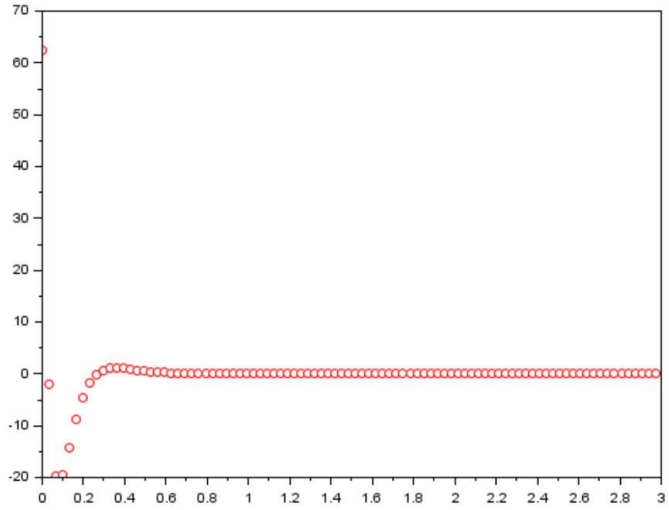

Figure 21: Closed loop control action.

The control action has huge over and undershoots. This means that if, for example, it is wanted to move the ball 1 cm, the angle that will be indicated first to rotate will be of 60º, which is much more than the greatest angle that the system can offer.

To reduce this over and undershoot, a prefilter is included to cancel the closed loop zero at z=0.95089, which is what causes the appearance of this behaviour. It will be also necessary to introduce in the prefilter a certain gain (1-0.95089=0.0491098) to compensate this cancellation.

$$F(z) = \frac{0.0491098 \cdot z}{(z - 0.95089)}$$

Equation 18: Prefilter expression.

$$\boldsymbol{T(z) = 0.00075417 \frac{z \cdot (z + 0.256214) \cdot (z + 3.57341)}{(z - 0.926909) \cdot (z - 0.62) \cdot (z - 0.61) \cdot (z - 0.6)}}$$

Equation 19: Closed loop filtered expression.

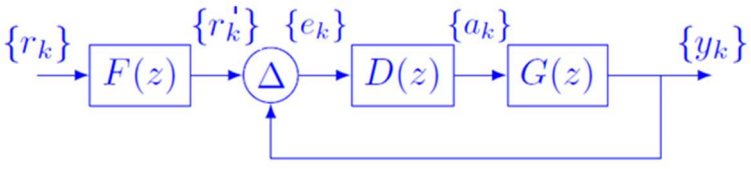

Figure 22: System's diagram including the prefilter. [20]

- r_k: ball's reference position.
- r_k': ball's reference position after being applied the prefilter.
- e_k: position error. Difference between the position of the ball in a certain time instant and its reference.
- a_k: controller's action.
- y_k: system's response. Position of the ball in a certain time instant.

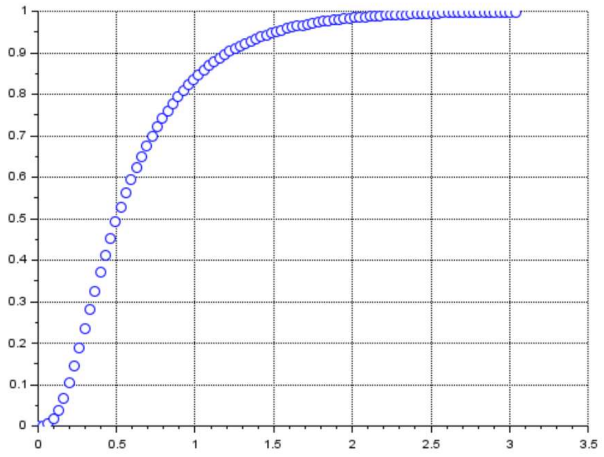

Figure 23: Closed loop prefiltered discrete step response.

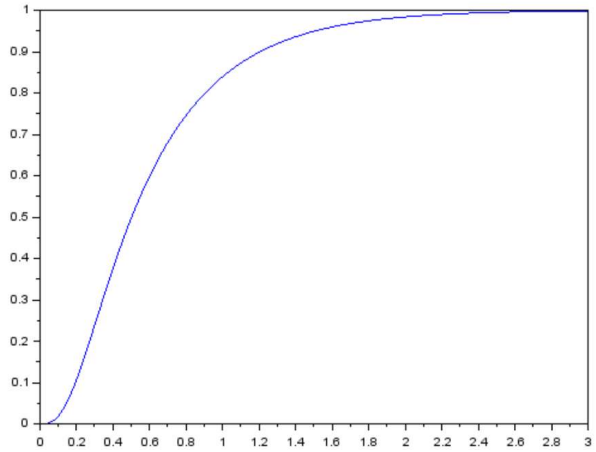

Figure 24: Closed loop prefiltered continuous step response.

Comparing the obtained response with the one above it can be seen that, thanks to the zero cancellation, the response's overshoot has fully disappeared making it much smoother than what it initially was. Nonetheless, it is interesting to highlight that the settling time remains invariant as the dominant closed loop pole responsible of this task has not disappeared from the transfer function.

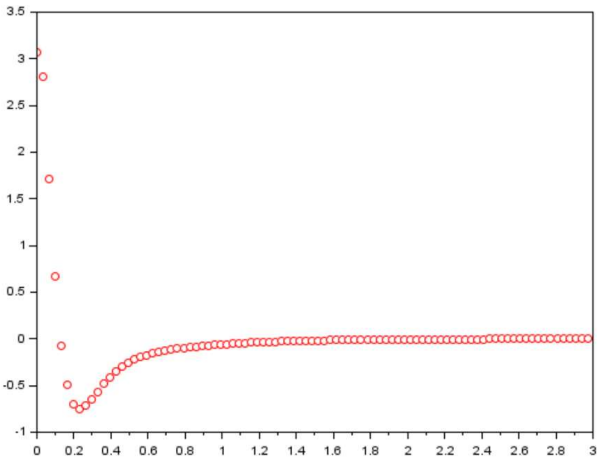

Figure 25: Controller's prefiltered action.

Regarding the **controller's action**, the over and undershoots have been considerably **reduced**. This means that, with this modification, if it is desired to move the ball, for instance, 3 cm, it would be necessary for the plate to rotate 9°, which is less than the maximum admissible for the plate; but if the distance to move would be greater than 3 cm it would reappear the same problem than before but in a smaller magnitude than before. From all the controllers designed, it was the one that performed the best.

To avoid rotations bigger than the maximum admissible for the plate that could endanger the system, it is introduced a **rate limiter** so that, in case of being requested an angle rotation greater than 10°, it would limit it not letting demand one greater than this magnitude. It will affect the settling time in those cases making it greater than what it was expected, but it is a security measure necessary to implement to protect the system.

7.2.3. Controller implementation.

In order to implement the controller in the Arduino board, it is previously needed to apply the inverse Z-transform to the controller and the prefilter to obtain their expressions in a sample form. To facilitate the obtention of these expressions, they are converted to their z^{-1} form.

$$\mathrm{D}(z) = 62.3772 \cdot \frac{(1 - 0.95089\,\mathrm{z}^{-1}) \cdot (1 - 0.838223\,\mathrm{z}^{-1})}{(1 - 0.772266\,\mathrm{z}^{-1} + 0.223704\,\mathrm{z}^{-2})}$$

$$(1 - 0.772266\,\mathrm{z}^{-1} + 0.223704\,\mathrm{z}^{-2}) \cdot \mathrm{D(z)} = 62.3772 \cdot (1 - 1.789113 \cdot z^{-1} + 0.797058 \cdot z^{-2}) \cdot \delta(z)$$

$$d_k - 0.772266\,d_{k-1} + 0.223704\,d_{k-2} = 62.3772 \cdot (\delta_k - 1.789113 \cdot \delta_{k-1} + 0.797058 \cdot \delta_{k-2})$$

$$\boldsymbol{d_k = 0.772266\,d_{k-1} - 0.223704\,d_{k-2} + 62.3772 \cdot (\delta_k - 1.789113 \cdot \delta_{k-1} + 0.7970579 \cdot \delta_{k-2})}$$

Equation 20: Controller samples.

$$F(z) = \frac{0.0491098 \cdot z}{(z - 0.95089)} = \frac{r'(z)}{r(z)}$$

$$(1 - 0.95089 \cdot z^{-1}) \cdot r'(z) = 0.0491098 \cdot z^{-1} \cdot r(z)$$

$$r'_k - 0.95089 \cdot r'_{k-1} = 0.0491098 \cdot \delta_{k-1}$$

$$\boldsymbol{r'_k = 0.95089 \cdot r_{k-1} + 0.0491098 \cdot \delta_{k-1}}$$

Equation 21: Prefilter samples.

Where δ(z) is referred to the input of the system.

As it can be seen, both expressions depend on previous samples of the controller or the prefilter and the input. Therefore, it will be necessary to store in a vector every sample taken and only discard them when they are no longer usable.

Once this implementation has been completed, the only missing step is testing the system's behaviour in a real environment.

8. Conclusion and future plans

After having implemented the designed controller in the Arduino board to test the system's real behaviour, it was observed that the **components initialization was correctly reproduced** both for the table and the ball, but the controller task, as it was expected, was not. The design of this controller was beyond the scope of this project and was only a first approximation of what would be necessary to carry out the control task, since arriving at a correct design requires a lot of time. Despite this fact, all the objectives described in the beginning such as the adaptation of the system to introduce an Arduino as a microcontroller and a phone camera as a position sensor, the initialization of components or the open loop control of the table, have been **fully accomplished,** so the project can be considered as a **success**.

As future plans, there can be inserted some sources of error that may affect to the controller's behaviour when being implemented. For example, a significant **vibration** appears on the plate due to the way that the pulse wave has been created and the type of motors employed that causes the ball to start **bouncing** and **lose contact with the plate's surface**. This contact lose complicates the control of the ball because in those instants where the ball is suspended it is not possible to modify its position. Another source of error may be **controller's action** that, as it has been said before, has an **overshoot** that might demand an excessively high angle when trying to reach a certain point making the action too aggressive and not enabling the ball to smoothly move.

Some proposed solutions to these problems could be designing a **better controller with a softer control action** that took into account the possible disturbances that the system may suffer; finding another way of creating the 400 Hz pulse wave introduced in the **step motors to decrease the plate vibrations**; or **using a heavier ball** instead of a table tennis one to **remove the ball's bouncing** (it must be taken into account that both the change in ball and frequency provokes a change in the system's dynamics, so its transfer function should also be modified).

Apart from designing a better controller, it could also be interesting to **substitute the wired connection** between the camera and the microcontroller by a **Bluetooth or IP connection** so that it could be monitored from any place without needing wires. Moreover, this change would allow the phone's port to be used as a power port instead of a data traffic one, thus eliminating the phone battery problem. Obviously, this kind of connection is less reliable that the other as interferences may have a greater influence, so more caution in the communication will be required.

Moreover, it could also be designed an **interface** containing a series of buttons with **geometrical figures** that, when being pressed, would indicate the system to draw with the ball that shape. For example, if the *triangle* button would be pressed, it would be said to the table to draw a triangular shape with the ball. For all this kind of tasks it would be critical to ensure a correct timing for each of them to avoid endangering the system functioning.

9. Bibliography

[1] A. Kastner, J. Inga, F. Koepf, T. Blauth, M. Flad and S. Hohmann, "Model-Based Control of a Large-Scale Ball-on-Plate System With Experimental Validation," in *IEEE International Conference on Mechatronics*, Ilmenau, March 2019.

[2] R. L. Giacco, "Eclipse," [Online]. Available: http://eclipse.baeyens.it/index.shtml. [Accessed 26 05 26].

[3] Humusoft, "Humusoft - CE151," [Online]. Available: https://www.humusoft.cz/models/ce151/. [Accessed 15 05 2021].

[4] A. A. Puerta, "Diseño y control de un sistema Ball and Beam con," Dep. Ingeniería de Sistemas y Automática, Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla, Sevilla, 2018.

[5] Leybold, "Leybold Ball & Plate," [Online]. Available: https://www.leybold-shop.com/ve6-3-5-9.html. [Accessed 05 05 2021].

[6] Leybold, "Leybold Inverted Pendulum," [Online]. Available: https://www.leybold-shop.com/ve6-3-5-11.html. [Accessed 05 05 2021].

[7] Leybold, "Leybold Twin Rotor MIMO," [Online]. Available: https://www.leybold-shop.com/ve6-3-5-12.html. [Accessed 05 05 2021].

[8] Quanser, "Quanser - Company mission," [Online]. Available: https://www.quanser.com/about/company-values-mission/. [Accessed 8 5 2021].

[9] Quanser, "Quanser - Control Systems," [Online]. Available: https://www.quanser.com/solution/control-systems/. [Accessed 9 5 2021].

[10] Quanser, "Quanser Inverted Pendulum," [Online]. Available: https://www.quanser.com/products/linear-flexible-joint-inverted-pendulum/. [Accessed 05 05 2021].

[11] Quanser, "Quanser MIMO Twin Rotor," [Online]. Available: https://www.quanser.com/products/quanser-aero/. [Accessed 05 05 2021].

[12] Quanser, "Quanser Ball & Plate," [Online]. Available: https://www.quanser.com/products/2-dof-ball-balancer/. [Accessed 05 05 2021].

[13] D. H. Amor, "Arduino Project Hub," Arduino, 21 05 2016. [Online]. Available: . [Accessed 26 05 2021].

[14] T. -. Humusoft, CE 151 BALL & PLATE APPARATUS - USER'S MANUAL., 1992-1996.

[15] Industrial-Electronics.com, "industrial-electronics.com. "The Slo-Syn synchronous step motor".," [Online]. Available: http://www.industrial-electronics.com/emct_2e_5i.html.

[16] T. E. Components, "TME Electronics Components. Motores paso a paso.," 08 09 2020. [Online]. Available: https://www.tme.eu/es/news/library-articles/page/41861/Motor-paso-a-paso-tipos-y-ejemplos-del-uso-de-motores-paso-a-paso/#:~:text=El%20motor%20paso%20a%20paso,eje%20cada%201%2C8%C2%B0.. [Accessed 15 05 2021].

[17] S. Electric, Installation instructions for SLO-SYN model SS2000MD4M-O microstep drive/oscillator, Schneider Electric.

[18] Arduino, "Arduino DUE properties," Arduino, [Online]. Available: https://store.arduino.cc/arduino-due. [Accessed 15 05 2021].

[19] Wikipedia, 10 05 2021. [Online]. Available: https://en.wikipedia.org/wiki/Newline. [Accessed 18 05 2021].

[20] G. L. Carreras, "Apuntes de Control Digital," Pamplona, 2020.

[21] P. Abeles, "CameraPreview.java example from the BoofCV library for Android(C)," 2011-2014.

[22] S. Electric, "bibus.hu," [Online]. Available: https://www.bibus.hu/fileadmin/editors/countries/bihun/product_data/kollmorgen/documents/superiorelectric_series_ss2000md4mo_manual_en.pdf. [Accessed 27 05 2021].

10. Ancillaries

10.1. Camera Android programming code

10.1.1. Camera Preview [21]

```
/*
 * Copyright (c) 2011-2014, Peter Abeles. All Rights Reserved.
 *
 * This file is part of BoofCV (http://boofcv.org).
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.example.ce151;
//package org.boofcv.example.android;

import android.content.Context;
import android.hardware.Camera;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.ViewGroup;

import java.io.IOException;

/**
 * Displays (or hides) the camera preview.  Adjusts the camera preview so that
the
displayed ratio is the same
 * as the input image ratio.
 *
 * @author Peter Abeles
 */
public class CameraPreview extends ViewGroup implements SurfaceHolder.Callback
{
        private final String TAG = "CameraPreview";

        SurfaceView mSurfaceView;
        SurfaceHolder mHolder;
        Camera mCamera;
        Camera.PreviewCallback previewCallback;
        boolean hidden;

        public CameraPreview(Context context, Camera.PreviewCallback
previewCallback,
boolean hidden ) {
                super(context);
                this.previewCallback = previewCallback;
                this.hidden = hidden;

                mSurfaceView = new SurfaceView(context);
                addView(mSurfaceView);
```

```
                // Install a SurfaceHolder.Callback so we get notified when
the
                // underlying surface is created and destroyed.
                mHolder = mSurfaceView.getHolder();
                mHolder.addCallback(this);
                // deprecated setting, but required on Android versions prior
to 3.0
        //        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        }

        public void setCamera(Camera camera) {
                mCamera = camera;
                if (mCamera != null) {
                        // need to start the preview here because it is
possible for it to be paused and
resumed and not
                        // have surfaceChanged called if the orientation
doesn't need to be changed.
Yes, you will have
                        // to start and stop the camera in the more common
situations
                        startPreview();
                        // Need to adjust the layout for the new camera
                        requestLayout();
                }
        }

        @Override
        protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec)
{
                int width,height;
                if( hidden ) {
                        // make the view small, effectively hiding it
                        width=height=2;
                } else {
                        // We purposely disregard child measurements so that
the SurfaceView will center
the camera
                        // preview instead of stretching it.
                        width = resolveSize(getSuggestedMinimumWidth(),
widthMeasureSpec);
                        height = resolveSize(getSuggestedMinimumHeight(),
heightMeasureSpec);
                }
                setMeasuredDimension(width, height);
        }

        @Override
        protected void onLayout(boolean changed, int l, int t, int r, int b) {
                if( mCamera == null )
                        return;

                if (changed && getChildCount() > 0) {
                        final View child = getChildAt(0);

                        final int width = r - l;
                        final int height = b - t;

                        Camera.Size size =
mCamera.getParameters().getPreviewSize();
                        int previewWidth = size.width;
                        int previewHeight = size.height;

                        // Center the child SurfaceView within the parent.
                        if (width * previewHeight > height * previewWidth) {
                                final int scaledChildWidth = previewWidth *
height / previewHeight;
                                l = (width - scaledChildWidth) / 2;
                                t = 0;
                                r = (width + scaledChildWidth) / 2;
                                b = height;
```

```
                        } else {
                                final int scaledChildHeight = previewHeight *
width / previewWidth;
                                l = 0;
                                t = (height - scaledChildHeight) / 2;
                                r = width;
                                b = (height + scaledChildHeight) / 2;
                        }
                        child.layout(l,t,r,b);
                }
        }

        protected void startPreview() {
                try {
                        mCamera.setPreviewDisplay(mHolder);
                        mCamera.setPreviewCallback(previewCallback);
                        mCamera.startPreview();
                } catch (Exception e){
                        Log.d(TAG, "Error starting camera preview: " +
e.getMessage());
                }
        }

        @Override
        public void surfaceCreated(SurfaceHolder holder) {
                if (mCamera == null) {
                        Log.d(TAG, "Camera is null.  Bug else where in code.
");
                        return;
                }

                startPreview();
        }

        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {}

        @Override
        public void surfaceChanged(SurfaceHolder holder, int format, int w,
int h) {}
}
```

10.2. Scilab controller simulations

```
// Control
clear
s=%s;
z=%z;
Tm=0.033;
// Tm=0.1;
ts = 2
// zp=0.9;
zp = exp(-4.6*Tm/ts)

// Modelo planta
Kb = 4.6;
KG = Kb * 100 * %pi /180
Gs = lti_system(KG/((s^2)*(0.187*s+1)))  // cms/grado
//lti_stepPlot(Gs)  // con una referencia de un grado recorre los 18 cms en 2.3
segundos

Gmz = lti_toDiscrete(Gs, Tm)
                cer pol K] = lti_getZpk(Gmz)
c1 = real(cer(1))
c2 = real(cer(2))
p = real(pol(3))

//Cálculo controlador
A=[ 1  0  0   0      0;
  -2  1  0   1      0;
   1 -2  1 -(c1+c2)   1;
   0  1 -2   c1*c2  -(c1+c2);
   0  0  1   0      c1*c2;
]

gammaz = (z-zp)*(z-0.6)*(z-0.61)*(z-0.62)
//cgamma = flipdim(coeff(gammaz),2)(2:$)
cgamma = flipdim(coeff(gammaz),2)

// b=[1; -zp; 0; 0; 0;]
 b = cgamma'
x=inv(A)*b

alfa0=x(1);
alfa1=x(2);
alfa2=x(3);
beta0=x(4)/K;
beta1=x(5)/K;

Beta=(beta0*z+beta1)*(z-p);
Alfa=alfa0*z^2+alfa1*z+alfa2;

Dz=lti_system(Beta/Alfa, Tm)
```

```
Tz = lti_feedback(Dz*Gmz)
lti_stepPlot(Tz)
// Simulación
tfin = 3;
tk = 0: Tm : tfin;
tc = 0: Tm/100 : tfin;

Sa = lti_feedback(Dz,Gmz)
ak = lti_simul(Sa, 'step', tk);
y = lti_simul(Gs, zoh(ak, Tm), tc);
plot(tc,y)
plot(tk, ak, 'ro')

// Prefiltrado
cer = -beta1/beta0
Fz = lti_system((1-cer)*z/(z-cer), Tm)
Tyz=Fz*Tz
lti_stepPlot(Tyz)
rk = lti_simul(Fz,'step',tk);
ak = lti_simul(Sa, rk, tk);
y = lti_simul(Gs, zoh(ak, Tm), tc);
plot(tc,y)
plot(tk, ak, 'ro' )

Siz = lti_system(Gmz,Dz)
Tyr = Fz*Tz
lti_stepPlot(Tyr)

[Dk]=lti_invZeta(Dz)
[Fk]=lti_invZeta(Fz)
```

10.3. Real time control code

```
// Do not remove the include below
#include "CE151.h"

#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "timers.h"

#include "converter.h"
#include "analogSignal.h"

#include <assert.h>

/*************************************************
 *                     Prototypes
 *************************************************/

void controlTask( void *pvParameters );
void commTask(void *pvParameters );

void rtPrint(const char * cad );
void rtPrintFlush( void );
template <typename T> void rtPrint(const char * openStr, const T & x, const
char *
endStr = nullptr );
void rtPrint(const char * openStr, const double & x, const char * endStr =
nullptr );




class MotorPasoAPaso
{
public:
        enum Direccion { CCW = LOW, CW = HIGH };

private:
        Direccion direc { CCW };
        int pFin;
        digitalOutput dir;
        digitalOutput pulso;

        const double pasosGrado {M_PI/(0.000628*180)};
        int nPasosObjetivo {0};
        int nPasos {0};

public:
        // Stablish direction of rotation
        void assignDirection( Direccion d ) { direc = d;
dir.write((bool)direc); }
        // It takes a step at 400 Hz on the initialization stage
        void initStep( void ) { pulso.on();  delayMicroseconds(12);
pulso.off();  delayMicroseconds(2500); }

        MotorPasoAPaso( int pinDir, int pinPulso, int pinFin );
        bool finCarrera( void ) { return (digitalRead(pFin) == LOW); }

        void init( void ); //It initializes moving the plate to a 0° position
```

```
        void setAngle( double angle );  // Fix the objective
        void step( void );  // if it is necessary it takes a step

};

void MotorPasoAPaso::setAngle( double angle )
{
        // *** Here it is coordinated the access to the nPasosObjetivo
variable. Thi function will be called from the control task.
        nPasosObjetivo = (int) (pasosGrado*angle);
};


void MotorPasoAPaso::step( void )
{

        if (nPasos == nPasosObjetivo) return;
        Direccion nuevaDireccion = (nPasos < nPasosObjetivo)? CW : CCW;
        // If we cannot take a step, we return doing nothing
        if ( finCarrera() && (nuevaDireccion == direc) ) return;
        // If the direction of rtotaion has been changed, it is actualised.

        if ( nuevaDireccion != direc )        assignDirection( nuevaDireccion
);
        // We take a step and we actualise the number of steps taken
        pulso.on();
        delayMicroseconds(12);
        pulso.off();
        nPasos += (direc == CW)? 1 : -1;
}


MotorPasoAPaso::MotorPasoAPaso( int pinDir, int pinPulso, int pinFin ) :
dir(pinDir), pulso(pinPulso)
{
        pFin = pinFin;
        pinMode(pFin, INPUT);
}

// It must be called before starting to take real time steps

void MotorPasoAPaso::init( void )
{
        assignDirection(CCW);
        do { initStep(); } while (!finCarrera());

// We have arrived to one of the end switches
// Turn around and start counting steps
        assignDirection(CW);
        int pasos {0};
        do { ++pasos; initStep(); } while (!finCarrera());

// Knowing the number of steps we can place the plate on the reference
position (0°)
        assignDirection(CCW);
        for( int i = 0; i <= pasos/2; i++ )        initStep();
// Now it must be in the desired reference position
        nPasos = 0;
}

MotorPasoAPaso motorX(50, 46, 48);
```

```
MotorPasoAPaso motorY(51, 47, 49);


const int frecPulsos {400};
const int periodoPulsos = 3;
int count {0};

void pulsosMotor( TimerHandle_t )  // It is called every “periodoPulsos”
{
	motorX.step();
	motorY.step();
}


struct commData
{
	double accX, accY;	// Desired position of the plate in each moment
	double bolaX, bolaY;
};

unsigned int limPriority( unsigned int asked )
{
	return tskIDLE_PRIORITY + std::min(asked, (unsigned
int)(configMAX_PRIORITIES - 1) );
}



QueueHandle_t commQueue {nullptr};
SemaphoreHandle_t arduinoSema {nullptr}; // Mutex for function from the
Arduino’s API

void inicializaMesa( void )
{
	motorX.init();
	motorY.init();
}

int leerDatosCamara(int * pframe, int * py, int * px);
void limpiarDatosCamara(void);

float mx,my,nx,ny;

void inicializaBola (void)
{

	motorX.assignDirection(MotorPasoAPaso::CCW);
	motorY.assignDirection(MotorPasoAPaso::CCW);
	for( int n = 0; n < 120; n++)
	{
		motorX.initStep();
		motorY.initStep();
	}
	delay(4000);
	int xpos1, ypos1;
	int frame;
	limpiarDatosCamara();
	int err = leerDatosCamara(&frame, & ypos1, &xpos1);
	if ( err != 0 )
	{
		Serial.println("Error calibrando camara 1");
```

```
            return;
        }

        motorX.assignDirection(MotorPasoAPaso::CW);
        motorY.assignDirection(MotorPasoAPaso::CW);
        for( int n = 0; n < 240; n++)
        {
            motorX.initStep();
            motorY.initStep();
        }
        delay(4000);
        int xpos2, ypos2;
        limpiarDatosCamara();
        err = leerDatosCamara(&frame, & ypos2, &xpos2);
        if ( err != 0 )
        {
            Serial.println("Error calibrando camara 2");
            return;
        }

        motorX.assignDirection(MotorPasoAPaso::CCW);
        motorY.assignDirection(MotorPasoAPaso::CCW);
            for( int n = 0; n < 120; n++)
            {
                motorX.initStep();
                motorY.initStep();
            }



            Serial.print(xpos1);
            Serial.print(", ");
            Serial.println(ypos1);

            Serial.print(xpos2);
            Serial.print(", ");
            Serial.println(ypos2);


        // Ball calibration


        mx=36.00/(xpos2-xpos1);
        nx=-18.00-mx*xpos1;

        my=-36.00/(ypos2-ypos1);
        ny=18.00-my*ypos1;



}

void limpiarDatosCamara(void)
{
    while(SerialUSB.dtr() && SerialUSB.available())
    {
      String xtrans;
      const char term = 10;
      xtrans=SerialUSB.readStringUntil(term);
    }
```

```
}
int leerDatosCamara(int * pframe, int * py, int * px)
{
      String xtrans;

      while(!SerialUSB.dtr() || !SerialUSB.available());

        const char term = 10;
        ººxtrans=SerialUSB.readStringUntil(term); //Read the data received
as a string

        if (xtrans.startsWith("C"))
        {
            Serial.println(xtrans);
              return -1;
        }


            if ( xtrans.startsWith("-"))
            {
                  Serial.print("Error: "); Serial.println(xtrans);
                  return xtrans.toInt();
            }


      int firstComa = xtrans.indexOf(",");
      String frame = xtrans.substring(0, firstComa);

      int secondComa = xtrans.indexOf(",", firstComa+1);

      String xpos = xtrans.substring(firstComa+1,  secondComa);
      String ypos = xtrans.substring(secondComa+1);

      *pframe = frame.toInt();
      *py = ypos.toInt();
      *px = xpos.toInt();
      return 0;
    }


void leerPosBola(float*py, float*px){
      int xpixel, ypixel, frame;
      leerDatosCamara(&frame, &ypixel, &xpixel);
      *px=mx*xpixel+nx;
      *py=my*ypixel+ny;
}

// The setup function runs once when you press reset or power the board
void setup()
{
        Serial.begin(115200);     // Computer (Programming port DUE)
        SerialUSB.begin(115200);  // Camera (Native port DUE)
        while(!Serial);
        Serial.println("[DBG] : A la espera ...");


        Serial.println("[DBG] : Inicializando la mesa");
        inicializaMesa();
        Serial.println("[DBG] : Inicializando la bola");
        inicializaBola();
```

```
        Serial.println("[DBG] : Creando objetos de FreeRTOS");
        Serial.flush();

        arduinoSema = xSemaphoreCreateMutex();

if (!arduinoSema)
      Serial.println("[DBG] : No se pudo crear el mutex Arduino");
      assert( arduinoSema );

 commQueue = xQueueCreate(1, sizeof(commData));

if (!commQueue)
      Serial.println("[DBG] : No se pudo crear la cola de comunicación");
      assert( commQueue );

        Serial.println("[DBG] : Creando tareas");
        Serial.flush();



        TaskHandle_t controlTaskHandle { nullptr };

    // As it is configured, FreeRTOS 10 is the maximum priority

BaseType_t xRet = xTaskCreate(controlTask, "ControlTask", 512, nullptr,
limPriority(1), &controlTaskHandle);

xRet &= xTaskCreate(commTask, "CommTask", 512, (void* )controlTaskHandle,
limPriority(3), nullptr);

        if (xRet == pdPASS)        Serial.println("[DBG] : Tareas creadas");
        else Serial.println("[DBG] : No se pudieron crear las tareas");
        assert( xRet == pdPASS );

    TimerHandle_t

motorTimer = xTimerCreate( "PulsosMotor", pdMS_TO_TICKS(periodoPulsos),
pdTRUE, nullptr, pulsosMotor );

 if ( motorTimer != NULL ) Serial.println("[DBG] : Temporizadores creados");
 else Serial.println("[DBG] : No se pudieron crear los temporizadores");

        Serial.flush();
        assert( motorTimer != NULL );

BaseType_t retMotor= xTimerStart(motorTimer, 0);

if ( retMotor != pdFALSE) Serial.println("[DBG] : Temporizadores iniciados");
else Serial.println("[DBG] : No se pudieron iniciar los temporizadores");

      Serial.flush();
      assert( retMotor );

        // start FreeRTOS
        vTaskStartScheduler();
}
```

```
unsigned long idleCounter {0};
void loop( void ) { ++idleCounter; }


/*-----------------------------------------------------*/
/*---------------------- Tasks ----------------------*/
/*-----------------------------------------------------*/


// Global data...
const unsigned int sampleTime {33}; // milisegundos
const unsigned int commTime {1000}; // milisegundos
constexpr unsigned commPeriod = commTime/sampleTime;


unsigned long lostSamplesNumber {0};
unsigned long badConversionsNumber {0};
unsigned long maxLoopMicros {0};



  void controlTask(void *)  // The control task.
  {
    rtPrint("[DBG] : Control task starts.\n");


    commData eldato;


    TickType_t xpreviousWakeTime = xTaskGetTickCount();
    TickType_t xLastWakeTime {xpreviousWakeTime};


    double postFrefantx {0.00};
    double postFrefanty {0.00};
    double err1x {0.00};
    double err2x {0.00};
    double err1y {0.00};
    double err2y {0.00};
    double accionx1 {0.00};
    double accionx2 {0.00};
    double acciony1 {0.00};
    double acciony2 {0.00};

    limpiarDatosCamara();

    while(true)
    {
         static unsigned long sampleNumber {0};

 // Read ball position

              float x,y;
```

```
            leerPosBola(&y,&x);

    //control task

            float xdeseo=x;
            float ydeseo=y;
            float postFrefx=0.95089*postFrefantx+0.0491098*xdeseo;
            float postFrefy=0.95089*postFrefanty+0.0491098*ydeseo;

            double errx = postFrefx - x;
            double erry = postFrefy - y;

            double accionx = 0.772266 *accionx1-0.223704 *accionx2+62.3772*(errx
    -1.789113 * err1x  +0.7970579 * err2x);
            double acciony = 0.772266 *acciony1-0.223704 *acciony2+62.3772*(erry
    -1.789113 * err1y  +0.7970579 * err2y);

            if (accionx < -10 ) accionx = -10;
            if (accionx >  10 ) accionx =  10;
            if (acciony < -10 ) acciony = -10;
            if (acciony >  10 ) acciony =  10;

            motorX.setAngle(accionx);
            motorY.setAngle(-acciony);

            accionx2=accionx1; accionx1=accionx;
            acciony2=acciony1; acciony1=acciony;
            postFrefantx = postFrefx;
            err2x = err1x;  err1x = errx;
            postFrefanty = postFrefy;
            err2y = err1y;  err1y = erry;


           if (++sampleNumber % commPeriod == 0)
           {
      // Send data to the monitor
                    eldato.accX = accionx;
                    eldato.accY = -acciony;
                    eldato.bolaX = x;
                    eldato.bolaY = y;

      // We are going to wait at most 2 millisconds
            xQueueSend(commQueue, (const void *) & eldato, pdMS_TO_TICKS(2) );
           }
       }
    }



    void commTask(void *pvParameters )
    {
            (void) pvParameters;

        commData elDato;

        while(true)
        {
            xQueueReceive(commQueue, (void * const) & elDato, portMAX_DELAY);
            rtPrint("-->    Pos. bola X: ", elDato.bolaX, " ");
            rtPrint("-->    Pos. bola Y: ", elDato.bolaY, "\n");
            rtPrint("  Angulo solicitado X: ", elDato.accX, " ");
```

```
            rtPrint("  Angulo solicitado Y: ", elDato.accY, "\n");
        }
    }
```

```
// rtPrint functions definition

void rtPrint(const char * cad )
{
        if (Serial && arduinoSema)
        {
                xSemaphoreTake(arduinoSema, portMAX_DELAY );
                Serial.print(cad);
                 xSemaphoreGive(arduinoSema);
        }
}

void rtPrintFlush( void )
{
        if (Serial && arduinoSema)
        {
                xSemaphoreTake(arduinoSema, portMAX_DELAY );
                Serial.flush();
                 xSemaphoreGive(arduinoSema);
        }
}

template <typename T> void rtPrint(const char * openStr, const T & x, const
char *
endStr )
{
        if (Serial && arduinoSema)
        {
                xSemaphoreTake(arduinoSema, portMAX_DELAY );
                Serial.print(openStr); Serial.print(x);
                if ( endStr ) Serial.print(endStr);
                 xSemaphoreGive(arduinoSema);
        }
}

void rtPrint(const char * openStr, const double & x, const char * endStr )
{
        constexpr int nDec {6};
        if (Serial && arduinoSema)
        {
                xSemaphoreTake(arduinoSema, portMAX_DELAY );
                Serial.print(openStr); Serial.print(x, nDec);
                if ( endStr ) Serial.print(endStr);
                 xSemaphoreGive(arduinoSema);
        }
}
```