Département Génie Electrique & Informatique

Tutored research report

**Development of a ball-on-plate system**

Léa Beghin

Timothé Bigot

Emma Couston

Inès Naji

Maxime Oliveira Lopes

Lucie Vorms

4AE-SE

Tutors: Elodie Chantery

Sébastien Di Mercurio

Gwendoline Le Corre

**Abstract**

| Ball-on-plate platform is a well-known system that has been widely studied in the past and well documented in the scientific literature. Stabilizing a ball on a plate is a concrete and major objective in automation. Even though it seems to be a simple action, it can be modeled in many ways and it is possible to apply various types of control laws with a wide level of difficulty. We are here considering a 3 degrees of freedom system based on the work previously done. From the previous model we identified several weaknesses such as a slow reaction time, where the system is not fast enough to stabilize the ball. We are also interested in finding other models that can possibly be used with the existing code and improving on them.The main objective of this project is to have a robust control law to have a better closed loop reaction and to be able to keep the ball on the plate. The final aim is to achieve an efficient feedback on the computer and control it in real-time in order to create a practical work session for AE students so they can apply automation concepts in a concrete manner. Last year, an Arduino board was used to manage the system. We decided to move to Raspberry Pi 4 to avoid communication issues, and use only MATLAB to progressively run the code in order to identify the source of the errors that may occur. We also decided to make simulations of the law control to validate its efficiency. In conclusion, this report presents the work done on this 3DOF ball on plate platform, based on the previous MATLAB/Simulink code of the previous years. An in-depth understanding of the system's functioning allows us to upgrade the code, but finally move to Raspberry and Python code to solve the time response problem in order to keep the ball on the plate. Reducing it with the Raspberry Pi 4 and MATLAB were not enough, and unfortunately we did not have time to test the Python code, supposed to ensure this time issue. |
| --- |

Keywords: Automation, Ball-On-Plate, Multivariable system, Controller, Modelisation

**Acknowledgements**

This project was made possible thanks to the help of several people to whom we would like to express our gratitude.

First of all, we would like to thank our tutors, Elodie Chantery, Gwendoline Le Corre and Sébastien Di Mercurio, for the time they devoted to providing us with the methodological tools we needed to carry out this research.

We would also like to thank José Martin for his patience in answering all our questions.

We also wish to express our gratitude to Vincent Mahout for his support in solving our configuration problem.

Finally, we would like to thank Joseph Shea for his advice on our writing style and presentation, and for his corrections.

**Table of contents**

[**Introduction 1**](#_heading=h.9wwhwjdhk1eb)

[**I. Description of the steps to achieve our goal 2**](#_heading=h.i2t4bqn7ww4i)

[a. Understanding of previous work 2](#_heading=h.cfswhd5qbapu)

[i. Discovery and analysis of the model 2](#_heading=h.s2j221lb88pr)

[ii. Understanding previous code 2](#_heading=h.fs1rcgk0hhm6)

[b. Identifying the problem 3](#_heading=h.kndwkfqaqi7e)

[c. Adapt the model slightly to make it easier to code 3](#_heading=h.p9apq4fnusgc)

[d. Work on the Arduino 4](#_heading=h.cwbnw5ozylqb)

[i. Geometric conceptualization of the problem 4](#_heading=h.374kbbc2skkb)

[ii. Application to the model 5](#_heading=h.1l621f1hh0to)

[e. Work on the Raspberry Pi card 6](#_heading=h.lqovy68y89dq)

[i. Raspberry Pi configuration set up 6](#_heading=h.gdd7f942qilv)

[ii. Converting the Simulink blocks to MATLAB code 7](#_heading=h.g6m57o8p1n55)

[iii. Converting MATLAB code into Python code 8](#_heading=h.aptnq72rvdnr)

[**II. Simulation and experimental results 9**](#_heading=h.9v5hoo7eodi0)

[a. PID simulation with MATLAB 9](#_heading=h.7lvh6ujkco84)

[b. Simulation with the Raspberry Pi 11](#_heading=h.cbo5dmza1ogr)

[**III. The next steps 13**](#_heading=h.xtb7t6gpnngi)

[**Conclusion 14**](#_heading=h.8jhwr2r76m2d)

[**Illustrations table 15**](#_heading=h.pnyw5dsren9m)

[**Figures table 16**](#_heading=h.wdnl808emnpu)

[**Bibliography 17**](#_heading=h.1i2hdtdzx89n)

[**Appendix table 18**](#_heading=h.mpg4see3fhqv)

# 

# 

# Introduction

The ball-on-plate system is an extension of the ball-on-beam system which is a classical nonlinear system with simple or multiple inputs and multiple outputs (also known as a MIMO). This system is common in different scientific fields. The objective of this system is to stabilize a ball on the plate by tracking its position using a camera and a sensor, and adapting the angle of the plate with servomotors, to shift and maintain the ball in the center of the plate, without it falling off the plate. Since this project has been conducted over several years, we took over work that had been previously carried out by another team, and our objective is to finally complete this project. To do so, we will need to address and fix the problem linked to the slow reaction time of the system and also implement a robust corrector to control the system.

Overtime, the ball-on-plate system has gathered significant attention from researchers around the world. This attention is due to the complexity and relevance of this system in robotics and control theory. Several scientific papers have explored various aspects of this system, namely its dynamic behavior, different control strategies, and applications. Researchers have studied multiple methods to stabilize the ball on the plate, ranging from common control techniques to more advanced nonlinear control methods. In addition, researchers have explored real-world applications of the ball-on-plate system in different fields such as industrial automation and medical robotics for example.

Despite the extensive research conducted on the ball-on-plate system and the fact that it is a very common problem in control theory, there still exist several gaps in the current literature that require further investigation. For instance, in existing studies, the control design has usually been based on a simplified model of the ball-ong-plate system by neglecting certain constants and phenomena that only partially influence the behavior of the system or make it nonlinear and consequently difficult to model. Another research gap lies in the development of robust control strategies that can effectively address uncertainties in the model, in the information from the sensors and the actuators, and disturbances in the system.

This study aims to model and simulate the system and its behavior, and to implement different correctors and control the closed-loop system. In addition, the performance of the correction will be evaluated. The main objective of the experiment is to implement a fast and robust system control while making it possible for the tutors to transform this project into practical work for engineering students at INSA.

This report is divided into several parts. First, we start with a description of the several steps we have followed during this project. Then, we present the different simulations and experimental results. We finish this report discussing the next stages of the project that we were not able to complete.

# Description of the steps to achieve our goal

## Understanding of previous work

### Discovery and analysis of the model

First, let us describe the system we are working on. Our model represents a system with 3 degrees of freedom. There is a round plate controlled by three servomotors. The ping pong ball is placed on this plate. In addition, a camera for detecting the ball is connected to an Arduino or Raspberry Pi 4 board. This board is itself connected to a computer, which sends the commands to the servomotors.

The ball-on-plate system works as follows:

When a ping pong ball is placed on the board, the camera detects it and sends the real-time image of the ball to the computer. The computer processes the image and calculates the position of the ball. Then, the control law (PID) sends a command to the three servomotors based on the ball’s position. The servomotors will move more or less to bring the ping pong ball to the center of the platform.



#### Figure 1: Image of our model [1]

### Understanding previous code

Given that we had to resume work previously done by another team, our initial focus was on understanding their progress before proceeding with any new tasks. This step was complicated since the structure of their project was not explained and not sufficiently detailed. We read the report which had guided the previous team to better understand their mathematical reasoning more accurately. Also, in order to work more effectively and understand better what had already been done, we talked directly to a student who had previously worked on the project. From this discussion, it became clear that the detection of the ball already worked and that our focus needed to be on finding a solution to have a fast and coherent reaction from the servomotors. Thus, with a clearer understanding of how the various project components worked together, we were able to run all the code on MATLAB and Simulink.

## Identifying the problem

When testing the work previously done on the model, it was clear there was a too long delay between the calculation of the control law and the actuation of the servomotors. After discussing this problem with our tutors, we assumed that the delay was due to the architecture of the system. In fact, the camera was linked to the computer and the computer was linked to the Arduino which in turn controlled the servomotors. As a result, there is a considerable propagation path for the transmission of information before it reaches the servomotors. We could not connect the camera to the Arduino since the camera was linked by a USB cable and the Arduino board does not have a USB port. As we needed to embed the code in the card to centralize all the information and the calculations, we decided to switch to a Raspberry Pi board which has both a USB port and GPIO ports for the servomotors. We still continued working on the Arduino in parallel while familiarizing ourselves with the Raspberry Pi.

## Adapt the model slightly to make it easier to code

In MATLAB, from the last group we knew that to identify the ball the following parameters must be chosen to make the algorithm work properly: *[2]*

1. Allowable color range: In order to identify the ball, a wide range that takes into account different brightness conditions. They had chosen an interval from hsv(5, 40, 50) to hsv(15, 100, 100), as shown in Figure 2.

#### Figure 2: Allowable color range : from hsv(5,40,50) to hsv(15,100,100) [2]

1. Image pre-processing: A x5 numeric zoom is applied in order to only analyze the platform.
2. Bloc Detection: The allowable area range for the object is set [100,10000] square pixels. The defined interval prevents the system from confusing the ball with very small orange objects (wire, digital noise) .
3. Post-processing: Of all the orange objects that can be identified, only the largest is considered. The function then returns the center of this object.

However, despite these parameters, we still encountered problems with this functionality. When the ball was not in the camera frame, the algorithm tended to detect the support of the model as a ball because its color is similar to brown. This led to a brutal and non-logical reaction from the model because it acted as if the ball was on the plate. The image pre-processing is not sufficient enough to avoid this reaction because the frame is square and the plate is a disc as shown in Figure 3.

#### Figure 3: Sample image taken by the camera and processed [2]

For these reasons, we chose to cover the support of the model with colored sheets of paper very different from the color orange, so that the support can not be detected as the ball.

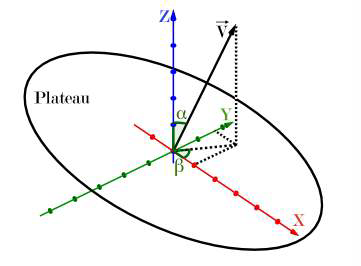
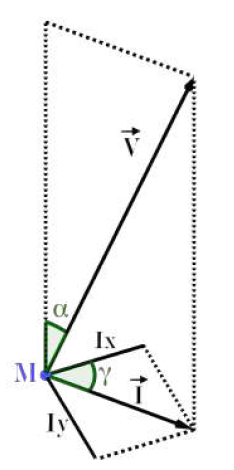
Once we dealt with this problem, and considering that the model was not attached to its support, it was difficult to calibrate the center and be sure how to control which servomotor. Therefore, we decided to glue it to the support after identifying each servomotors. Finally, we calibrated the center.

## Work on the Arduino

### i. Geometric conceptualization of the problem

To model the problem geometrically, we needed to divide it into multiple steps. First of all, we need to correct the position of the ball in relation to 2 axes, the X axis and the Y axis which are in the plate’s plane. We need to correct two directions, so we need two PIDs that give us the Ix and Iy components of the I vector corresponding to the desired direction of the ball to correct its position.

Then, from this vector I, we deduce a unit vector which is always perpendicular to the plate and whose origin corresponds to the center of the plate. The orientation of this vector is defined by the angles 𝛼 and 𝛽. To be more precise, 𝛼 is the angle between the horizontal plane and the plane containing the plate and 𝛽 defines the orientation of the plate.



#### Figure 4: Vector and angle representation [3]

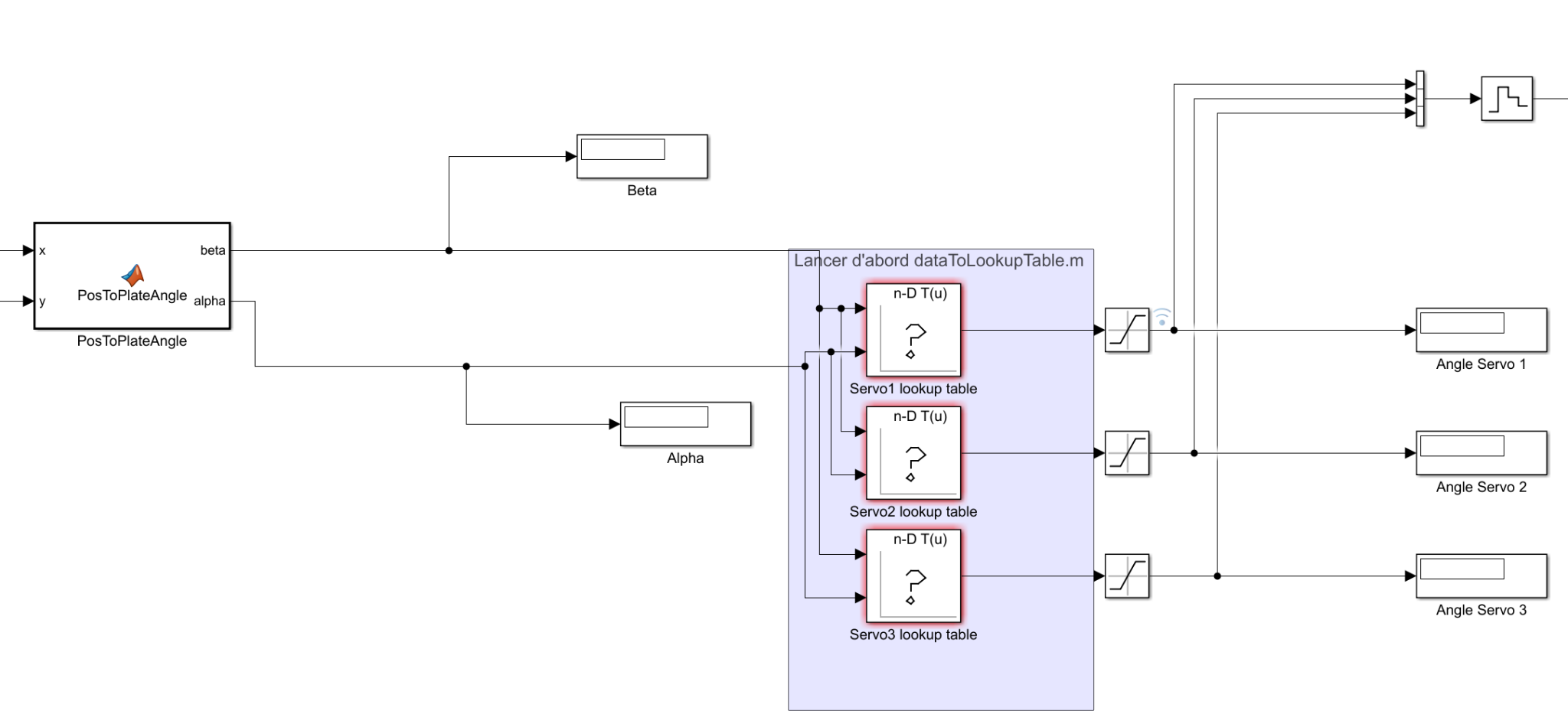
Once we have all of the information, we determine the angle of the servomotors thanks to a matrix of information generated by an equation.

In addition, it is important to note that the model is only capable of tilting the plate a maximum of 35°. Therefore 𝛼 must be inferior or equal to 35° and this condition must be applied in the Simulink model.

### ii. Application to the model

Once we ran the code, we detected some anomalies that seemed not to be linked with the delay. Indeed, only one of the three servomotors moved and the function which calculates the angle of the plate regularly generated an error.

Thanks to the display (where Alpha, Beta and Angle Servo 1, 2, 3 are indicated in figure 5 and 6), we found that the servomotors were not moving because their angle values were too high in comparison to the saturation. First, we wondered if the saturation was too low or the computed angle was wrong. But after comparing the Simulink model with the theory, we realized that the saturation was applied to the wrong angle, it was on the servomotors’ angle instead of the plate’s angle (𝛼).



#### Figure 5: Extract of the Simulink done by the latest group on this project



#### Figure 6: Extract of the Simulink corrected with the saturation on the right angle (𝛼)

This was not the only error because once we corrected it, 𝛼 turned out negative. We took a closer look at the formulas implemented in the code which calculates it and it was not geometrically correct.

Finally, after these corrections were made we obtained a satisfactory result despite the remaining delay. Manual movement of the ball over the plate generated a consistent response in terms of plate inclination. However, as the servomotor angle update times were too slow, the ball did not remain on the plate if we dropped it onto it.

## **e. Work on the Raspberry Pi card**

In order to overcome the problems associated with system slowness, we decided to run the project on a Raspberry Pi 4 board instead. In addition to having a USB port, the Raspberry Pi contains a more powerful processor than the Arduino card, this would mean we would not have to go through the computer when processing the image taken by the camera which was also slowing the response time of the system.

### i. Raspberry Pi configuration set up

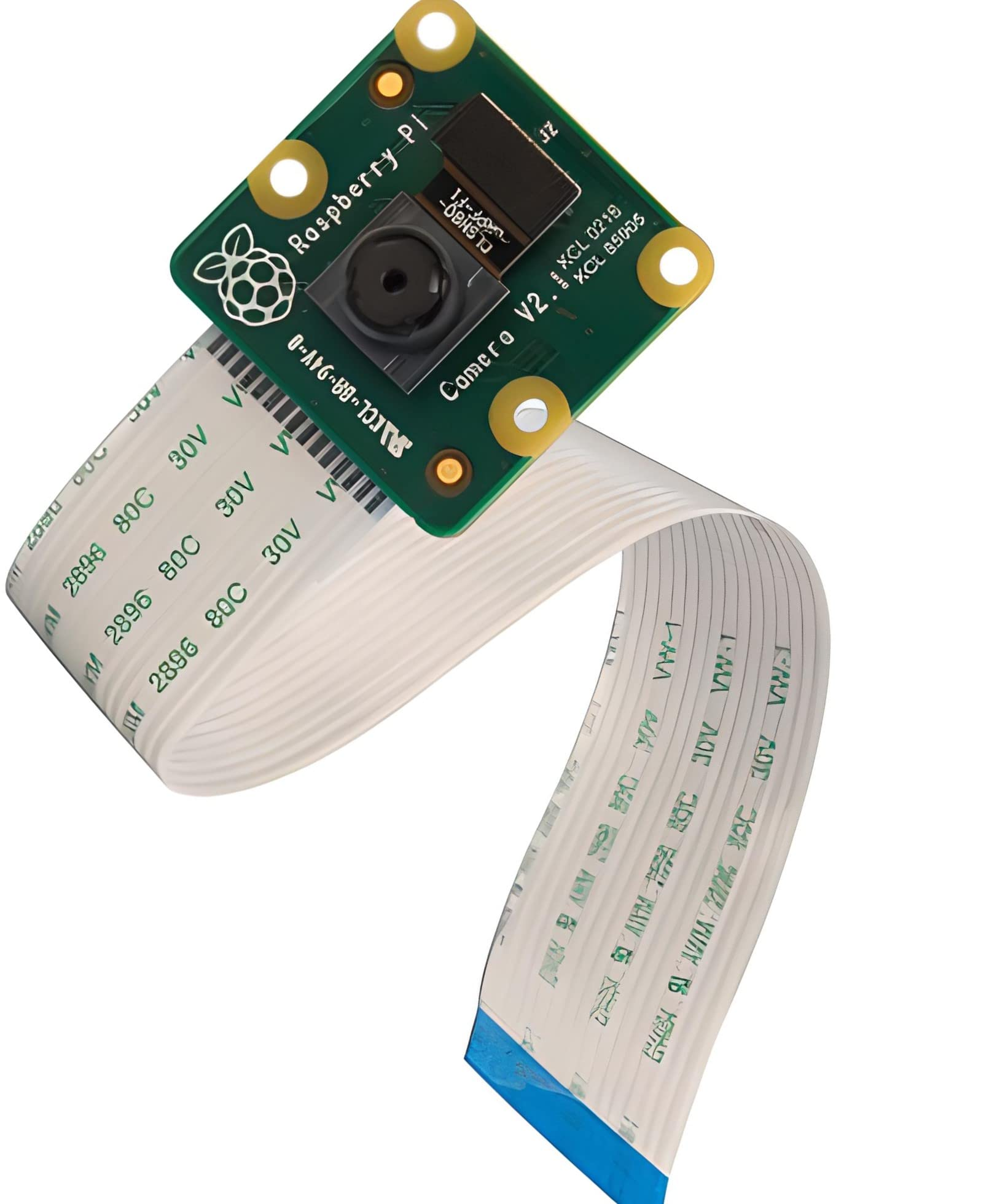
As a first step, we carried out some research to gain a better understanding of the Raspberry Pi 4 card's functionality. We were able to start configuring it by installing the Raspbian MathWorks OS on it. As the aim was to run the board via MATLAB, we also had to download the following packages containing ready-made Raspberry libraries for use with this software: “MATLAB Support Package for Raspberry Pi Hardware” and “Simulink Support Package for Raspberry Pi Hardware”.

Very quickly, we were confronted with connection problems between the card and MATLAB. The same error kept cropping up, but we could not work out what the problem was. Therefore, we started looking in forums for solutions to our problem. We also tried reinstalling the OS and the Raspberry libraries several times to no avail. We even ran the code on the online version of MATLAB. We spent a lot of time trying to understand this mistake. However, it did give us a better understanding of how the card works and the different functions of the libraries.

Our tutors finally solved the problem by finding that the image installed by MATLAB on the board was no longer being updated. In fact, some of the libraries are no longer maintained by Rasberry. This caused the connection to fail.

Once the connection had been made, we tested some basic functions provided by the libraries. For example, we switched on LEDs, set up PWMs and configured the three servomotors. These tests enabled us to check that the board's peripherals were working properly and to check the model's servomotors.

Then, we adapted the Simulink blocks already made for the Arduino board so that they could be implemented on the Raspberry Pi. We first encountered a problem with the USB camera used by the previous group. It was impossible to recover the image taken via a Simulink block, there was no device to process the image in Simulink. Therefore, we opted for the Raspberry Pi camera supplied with the board. However, the same problem occurred. After consultation with our tutors, we decided to script our Simulink blocks in MATLAB because the libraries provided all the functions needed to acquire the image with it.



#### Figure 7: USB camera (left) and Raspberry camera (right) used [4]

### ii. Converting the Simulink blocks to MATLAB code

The aim of this part was to rewrite the Simulink blocks in MATLAB code. The difficulty lay in sorting out the information from the blocks that were useful to us and finding the MATLAB code that had to be used. After creating the MATLAB script, we structured the code to make it easier to read and maintain. Next, our work focused on image acquisition via the Raspberry Pi's camera, to ensure smooth and accurate acquisition for our application.

This transfer work was quite laborious because we had to write the connection links between the blocks that were not materialized in Simulink. We also added the portions of code relating to the configuration of the card and the acquisition of the image taken by the camera. In addition, we wrote the function needed to control the servomotors, using the angles requested to calculate the duty cycle and then update the PWMs of each servomotor.

Once all the blocks had been transcribed, we tested our programme on the system. Unfortunately, the system response time was still slow. We found that configuring the PWM settings and calculating the duty cycles took up the majority of the time. By moving the PWM configurations to the beginning of the code, we were able to reduce the system response time. However, it was still not fast enough to allow the system to control the ball effectively. These tests are described in more detail in part 2.b.

### iii. Converting MATLAB code into Python code

In order to reduce response time as much as possible, we decided to code our program in Python because it has a wide range of libraries and cross-platform compatibility which makes it well-suited for interfacing with hardware and developing embedded systems efficiently. To do this, we used the PyCharm IDE, which is an easy-to-use interface. Since we already had the MATLAB code and based our research on other projects that used Python, it was easier and faster than converting the Simulink blocks to MATLAB code.

We converted each MATLAB block into Python functions by comparing the MATLAB code with the Python code from similar projects [3], [5]. Once we had converted all the main blocks, we tried to run the ones that were linked to physical objects, such as the camera and servomotors, on the Raspberry Pi 4.

The first problem that occurred was an issue with OpenCV, the library we used for video processing. When we tried to capture a video, an error was displayed saying that the system had failed to allocate the required memory. This type of error is unexpected on a Raspberry Pi 4, since the board normally has enough resources to handle basic video processing tasks without encountering any memory allocation problems.

Then we tried to see if the PWMs we had set up were working by looking at the pins concerned on the oscilloscope. However, as we could not see any activity on the oscilloscope, we tried to do a toggle operation on the same GPIO pin, pin 13, to know whether the issue was specifically with the PWMs. Observing a toggle on this pin confirmed that the PWMs should have been operational.

## 

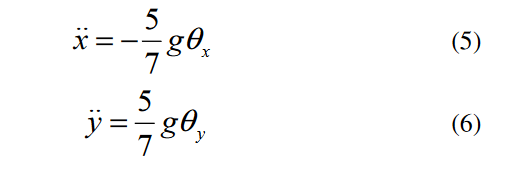
# 

# II. Simulation and experimental results

## PID simulation with MATLAB

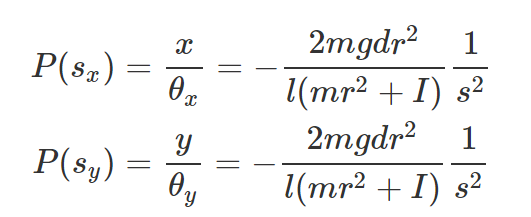
During the configuration of the Raspberry Pi 4, we decided to simulate in parallel the system already initiated for the Arduino board. Consequently, we tested the model for setting the PID in simulation. To do this, we went through the various existing models to help us model our system on MATLAB.

We found various equations for the system and for the transfer function. We regularly found in our sources the same simplification which leads to the same transfer function with a coefficient of 5/7. However, we found it difficult to understand where this simplification came from, given that these transfer functions ought to take into account the system's own parameters. After consultation with our tutors, this coefficient results from the simplification of parameters intrinsic to the system.



#### *Figure* 8*: Transfer function of the system [6]*

We also found another transfer function as shown in figure x. Nevertheless, by using these functions, we obtained a negative numerator, which leads to a completely inconsistent result. In fact, the position curve decreases towards minus infinity.



#### *Figure* 9*: Problematic transfer function [7]*

Next, we simulated the system with different values of the PID parameters. The PID controller can be represented as follows:

with Kp: the proportional gain, tau\_i: the integration time constant and tau\_d: the derivation time constant.

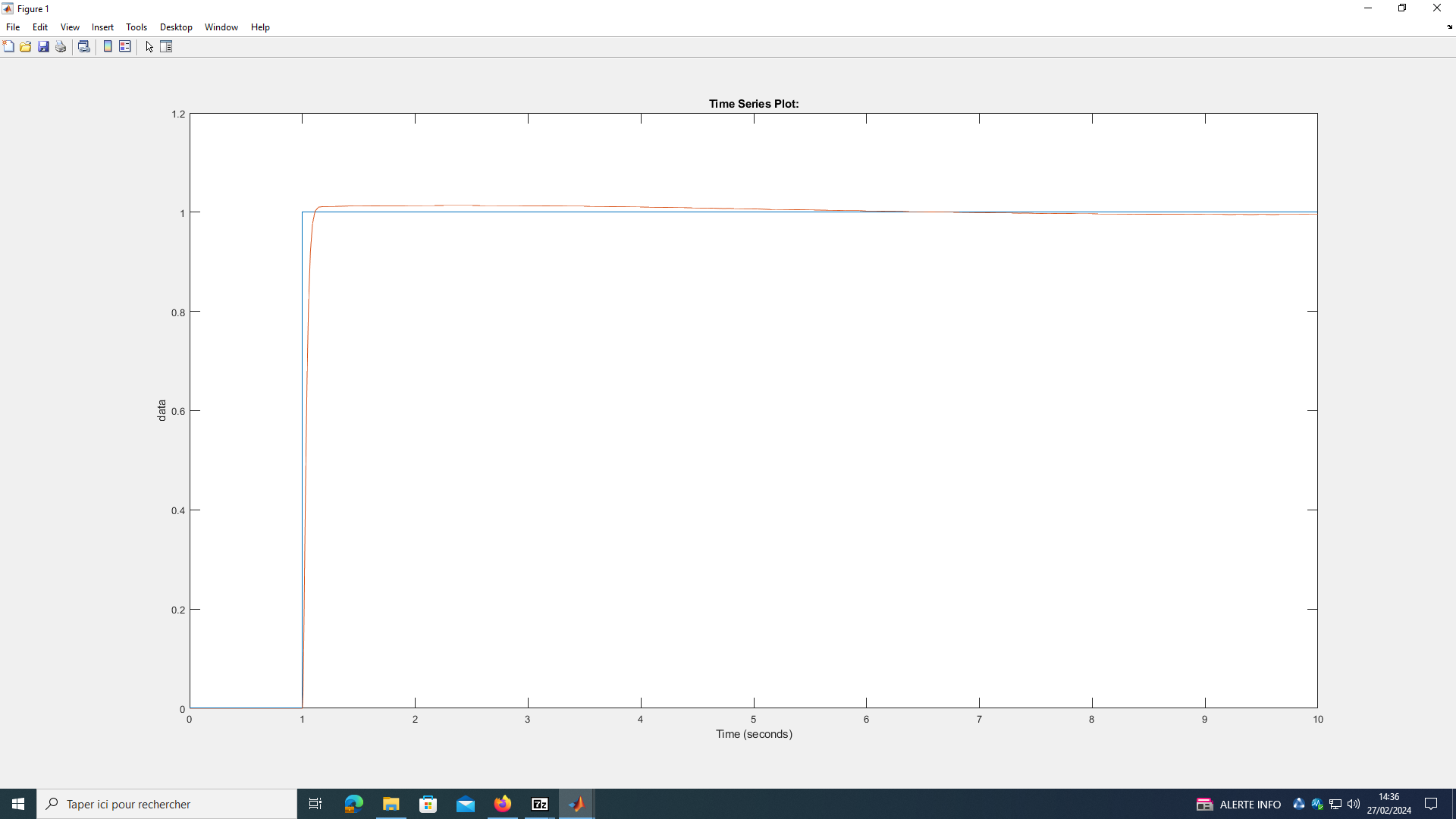
We first observed that without a control law, the system oscillates. By running several simulations, we found that by setting Kp = 1.2, tau\_i = 1.1921 and tau\_d = 0.2980, the static error was corrected thanks to the integrator parameter. However, there is still a significant overshoot of 48.5%. Therefore, the derivator needs to be changed.

We have transcribed our results obtained by varying the value of the derivation parameter, tau\_d, in the following table.

###### Table 1: Overshoot as a function of the value of tau\_d

| tau\_d | 0.298 | 0.5 | 0.8 | 1.1 | 1.5 | 1.8 | 2.2 | 3 | 4 | 5 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Overshoot | 48.5% | 28.7% | 15.4% | 9.4% | 5.6% | 4.1% | 2.8% | 1.7% | 1.1% | 0.8%  and 2.5% |

Analyzing this table, we note that for the value tau\_d = 4, the overshoot is the lowest (1.1%). Furthermore, for values greater than 4, the system shows several overshoots. With this in mind, we have chosen the PID with the following parameters: Kp = 1.2, tau\_i = 1.1921 and tau\_d = 4. The result of the simulation is presented below.



#### Figure 10: Response to a step (blue) in the system controlled by our PID (red)

We tested it in real life with the Arduino and it did not work at all. We did not obtain the same response as in the simulation. The plate behaved randomly. We then tried to remove the derivative component of the PID. The system worked as before, but the delay was still too long.

## Simulation with the Raspberry Pi

The goal was to identify the code sections or instructions that were responsible for the long response time in order to be able to improve them for better system responsiveness. To do this, we connected an oscilloscope probe to the GPIO 6 pin of the Raspberry Pi, to be able to have feedback on the time spent on each section, by sending orders of GPIO high or low state, respectively at the beginning and ending of each section. In this way, it is easy to measure the time on the oscilloscope and identify what causes the slow response.

After setting up the MATLAB code with all the GPIO lines, and connecting the oscilloscope to the Raspberry Pi 4, we were able to begin measurements by running the code and save it (see annex 1, annex 2) in real time when high and low states were detected.

The configuration part is about 4.6ms both in initial and edited code. In this way, we decided to focus on the code that could be improved in the while loop. The following table summarizes the results obtained.

###### Table 2: Time response comparison between initial and optimized code

|  | Initial code | Edited code |
| --- | --- | --- |
| Image acquisition | 21.5 ms | 19.4 ms |
| Detection of the ball | 14 ms | 12.6 ms |
| Distance to the center | 4.3 ms | 3 ms |
| Search into the lookup table | 10.8 ms | 9.8 ms |
| Position to plate angle | 4.2 ms | 2.8 ms |
| PID execution | 3.6 ms | 4.4 ms |
| Updating the actuators | 238.6 ms | 57.4ms |
|  | **301.64 ms** | **114.02 ms** |

We can see that the section of code “Updating the actuators” is the longest one in time execution. So we chose to optimize it as much as possible, especially by moving the declaration lines to the beginning of the MATLAB code, in the configuration part. Concerning the other parts, there is no optimisation possible, as the lines must be executed repetitively in the while loop, so cannot be moved up in the code. Moreover, it is not relevant to try to change the code by itself as we are using enough efficient functions or too specific ones to be substituted by others.

We divided the time execution system by almost a third, which is significant progress. However, it is still not enough to keep the ball on the plate. We confirmed this by running the code and testing it with the ball. After discussing it with our tutors, we decided to finally use Python via Pycharm and OpenCV in order to have a better time response.

# 

# 

# III. The next steps

Unfortunately, we did not have time to solve the problems associated with implementing the Python code on the board and getting our model to work with it. However, we do have a number of ideas for resolving these errors and even improving the system's response time.

Firstly, we used the rpi.gpio library to code our scripts in Python. This contains functions for initializing and modifying peripherals such as GPIOs and so on. It is therefore with rpi.gpio that the PWMs are assigned. So, one possibility would be to test the PWMs with another library to find out where the problem comes from.

Mr. Di Mercurio had managed to test PWMs with his own board and had checked that they worked properly. However, we were not able to connect to his card to see how his code worked and the libraries used. In addition, changing cards would be a good way of finding out whether the memory allocation problem is specific to ours or whether it is linked to this model of card in general.

Once the Python code is working, the aim would be to implement it in the board and run our system to evaluate its behavior. Hoping to obtain a much faster response time, we would then have to modify the PID to control the angle of the servomotors according to the position of the ball.

On the other hand, it might be a good idea to find a way of reducing the execution time of MATLAB code. Even though we have been able to divide it by three, the system is still far too slow. So we need to be able to update the servomotors more quickly.

# Conclusion

In conclusion, we presented the model with the Simulink and MATLAB code which constituted the beginning of our project. To address the delay caused by the considerable propagation path for transmitting information from the camera to the servomotors, we decided to use a Raspberry Pi 4 card instead of an Arduino one. This card was especially chosen because we can directly connect the camera to it and run the code on it, as it is a micro computer.

To execute the code on the Raspberry Pi 4, we needed it to be in MATLAB, so we converted the Simulink model into MATLAB code. We conducted tests to measure the new initial delay without any correction. These tests revealed that the delay was primarily due to the "Updating the actuators" section of the code, specifically the declaration lines within this section. By moving these declaration lines before the infinite loop, we were able to reduce the iteration time by a factor of three.

Finally, we decided to convert the MATLAB code to Python because it is the native language of the Raspberry Pi 4. However, we encountered other problems during the initial intermediate testing of the Python code. Image acquisition generated a memory allocation error, while PWMs were not generated. That is why, in order to finalize this project, the next steps are to solve these problems and debug the rest of the Python code. Once this is done, the PID can be parameterized.

# Illustrations table

[Figure 1: Image of our model 2](#_heading=h.238vg8dmbw5w)

[Figure 2: Allowable color range : from hsv(5,40,50) to hsv(15,100,100) 3](#_heading=h.iu8cs59i334f)

[Figure 3: Sample image taken by the camera and processed 4](#_heading=h.95v61af4rsqe)

[Figure 4: Vector and angle representation 4](#_heading=h.rzj3bysybtor)

[Figure 5: Extract of the Simulink done by the latest group on this project 5](#_heading=h.xwr309t7eghr)

[Figure 6: Extract of the Simulink corrected with the saturation on the right angle (𝛼) 6](#_heading=h.lg1vwd3ueu7o)

[Figure 7: USB camera (left) and Raspberry camera (right) used 7](#_heading=h.n0i0owy0rnye)

[Figure 8: Transfer function of the system 9](#_heading=h.qk66trxdr2lw)

[Figure 9: Problematic transfer function 9](#_heading=h.kxyko0dsfmuq)

[Figure 10: Response to a step (blue) in the system controlled by our PID (red) 10](#_heading=h.1wlsltzgx7kb)

# Figures table

[Table 1: Overshoot as a function of the value of tau\_d 10](#_heading=h.c65e5yhzv55)

[Table 2: Time response comparison between initial and optimized code 11](#_heading=h.mhw8qlmmuaw9)

# Bibliography

[1] « Cours : I4IRPJ11, I4AEPJ11, I4ISPJ11 : Projet d’Initiation à la Recherche | Moodle ». Available: <https://moodle.insa-toulouse.fr/course/view.php?id=1207>. [Accessed: Feb.25, 2024]

[2] ‘PIR\_Ball\_Balancing/Report/Report\_PIR\_ae09\_p03.pdf at main · TacTac315/PIR\_Ball\_Balancing’. Available:<https://github.com/TacTac315/PIR_Ball_Balancing/blob/main/Report/Report_PIR_ae09_p03.pdf>. [Accessed: May 18, 2024]

[3] Link, Johan, « Ball-Balancing-PID-System/TM\_SJF\_version.pdf at master JohanLink/Ball-Balancing-PID-System », *GitHub*. Available:<https://github.com/JohanLink/Ball-Balancing-PID-System/blob/master/TM_SJF_version.pdf>. [Accessed: Dec. 9, 2023]

[4] ‘Raspberry Pi v2.1 8 MP 1080p Module Caméra : Amazon.fr: High-Tech’, *Amazon*. Available: <https://www.amazon.fr/Raspberry-Pi-1080p-Module-Cam%C3%A9ra/dp/B01ER2SKFS> [Accessed: May 18, 2024]

[5] N. Hammje, ‘nicohmje/PID-ballonplate’. May 10, 2024. Available: <https://github.com/nicohmje/PID-ballonplate> [Accessed: May 18, 2024]

[6] Qasrawi. Anas, « Ball and Plate Balancing System », *Palestine Polytechnic University*. Jan. 2018. Available: <https://scholar.ppu.edu/handle/123456789/6308>. [Accessed: Dec. 22, 2023]

[7] F. Zheng, X. Li, S. Wang, et D. Ding, « Position control of ball and plate system based on switching mechanism », in *2011 IEEE International Conference on Automation and Logistics (ICAL)*, Aug. 2011, p. 233‑237. doi: [10.1109/ICAL.2011.6024719](https://doi.org/10.1109/ICAL.2011.6024719). Available:<https://ieeexplore.ieee.org/abstract/document/6024719?casa_token=bfyrrespBkcAAAAA:_3iz0iembGVvkJxWnu6yjXglOSvlakJPItgWz4_lQ285hTZZFTS-X7MvITWlpaE8xH-lLwRclA> [Accessed: Dec. 21, 2023]

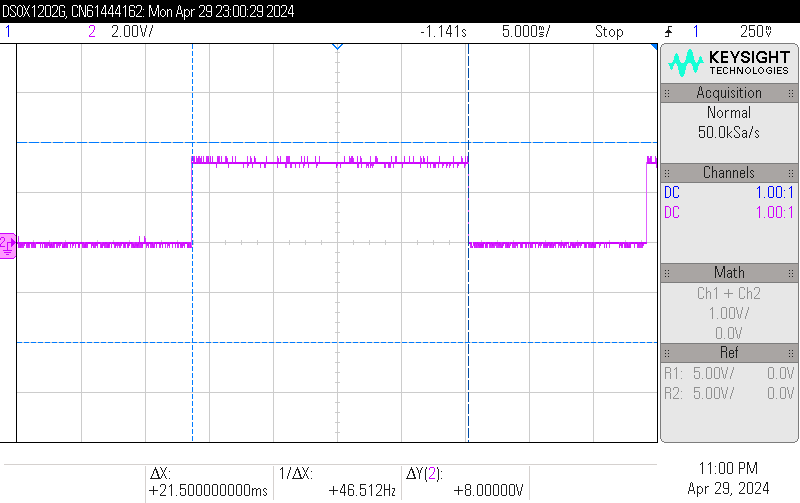
# 

# Appendix table

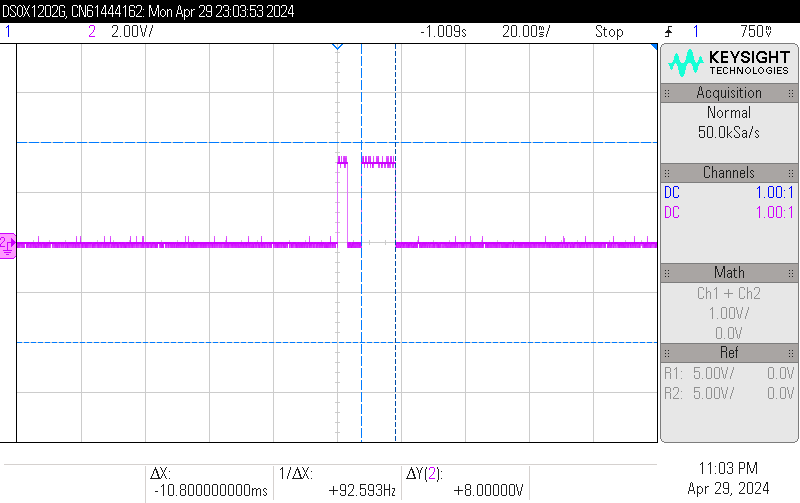
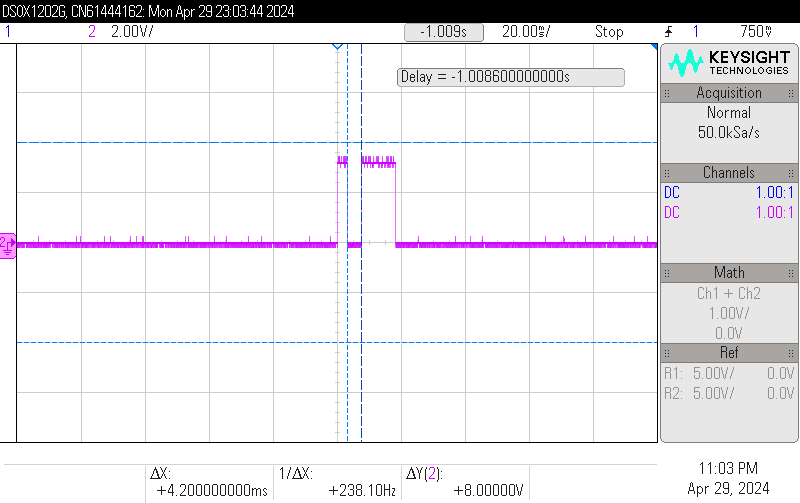
[Appendix 1: Time response of the initial code A1](#_heading=h.bmr8ar4kru64)

[Appendix 2: Time response of the optimized code A2](#_heading=h.fq9hrgdrxfcd)

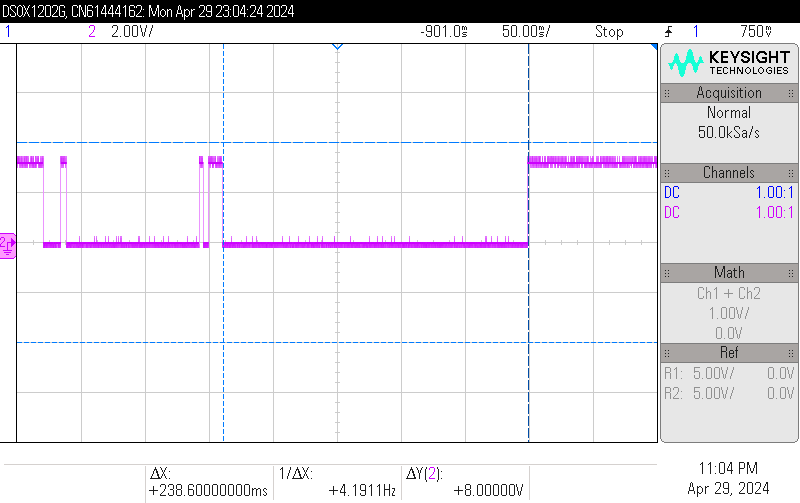
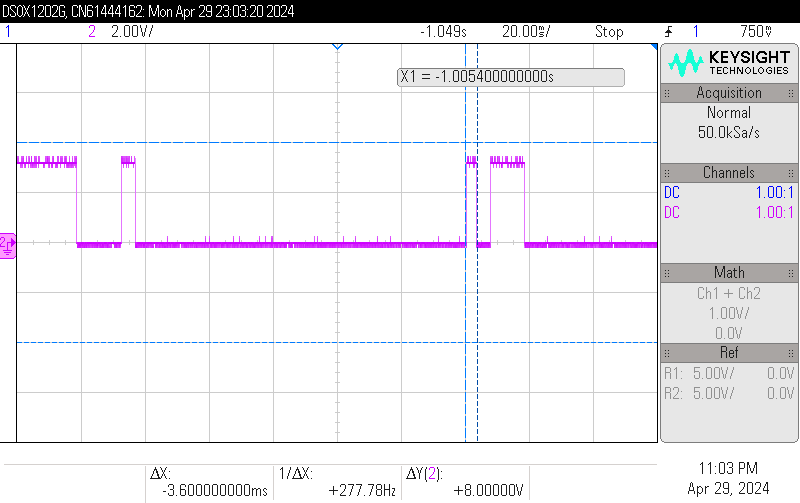
##### Appendix 1: Time response of the initial code



*Figure 1: Image acquisition Figure 2: Detection of the ball*

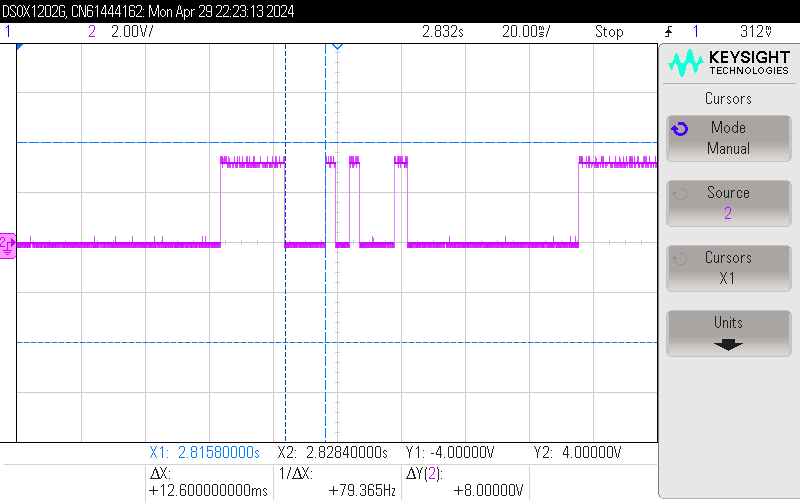
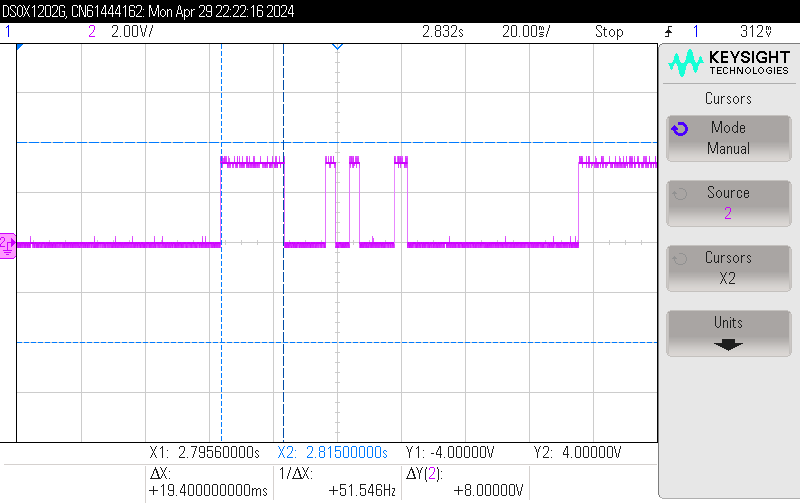


*Figure 3: Position to plate Angle* *Figure 4: Search in the lookup table*

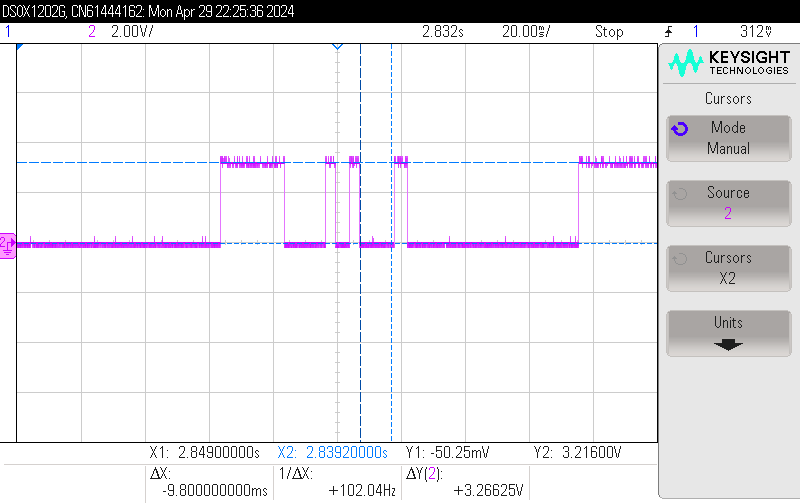
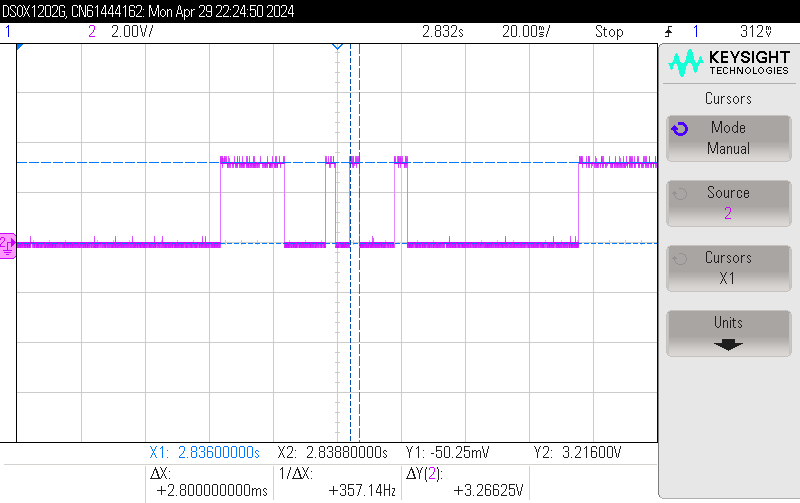


*Figure 5: PID execution Figure 6: Updating the actuators*

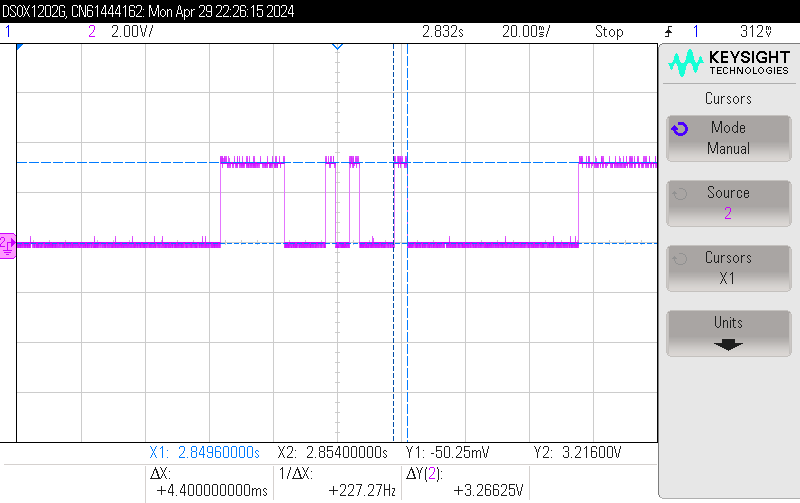
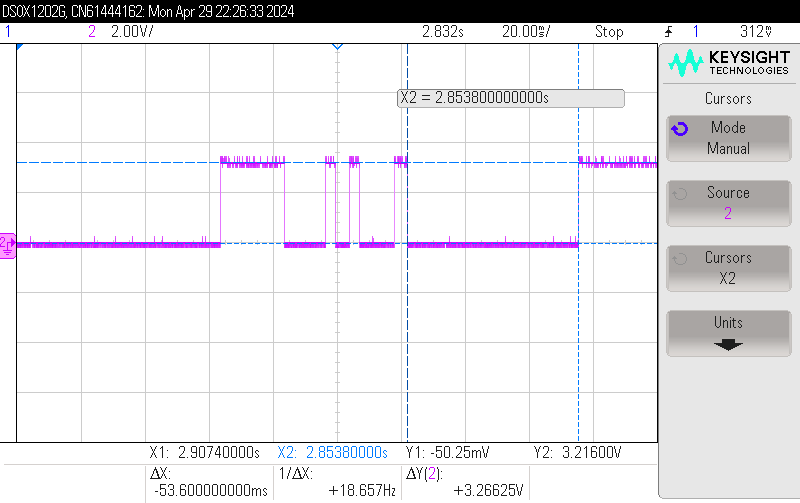
##### Appendix 2: Time response of the optimized code



*Figure 1: Image acquisition* *Figure 2: Detection of the ball*



*Figure 3: Position to plate Angle* *Figure 4: Search in the lookup table*



*Figure 5: PID execution Figure 6: Updating the actuators*