POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

2D Ball Balancing Table

Mechatronics Lab Report

Authors:
Barutta E. 10726132
Milic K. 11012938
Visentin N. 10797203

Lecturers:
Marconi J.
Pozzi M.

Academic Year: 2024/25

Contents

1 Introduction

This report describes the work carried out as part of the Mechatronics Lab course on a ball balancing system with a tiltable table. In particular, the goal was to design, develop and test some suitable control strategies to stabilize a small ball in a certain position or along a certain trajectory on a moving flat surface, by exploiting the feedback on the position of the ball and actuators to tilt the table. At the same time, problem-solving skills and critical thinking were needed to face some software and hardware issues.
In the first chapter, the system's hardware and software setup are described, along with the calibration process and all necessary procedures for preparing the bench for practical testing. Then the mathematical model of the system is presented, which will be essential for design the controls and other components. In chapter 4, different control strategies are explored in detail, focusing on both stabilizing the ball in a fixed position and making it follow a predefined trajectory.

2 Bench setup and calibration

Two different benches were used during the course of the laboratory: the provided one, which exploits a touchpad to sense the position of the ball, and a new one, built by us, that instead makes use of a camera. The decision of introducing an additional setup arises mainly due to three factors:

- The original balancing table is quite old: it relies on many software components that are not up to date and it shows some hardware malfunctioning (unstable cables connections, motors with slipping rods, reduced range of the touchpad, etc).
- The original balancing table works with a microcontroller with a lower computational power.
- The new balancing table relies on a camera to detect the ball position, allowing for testing a different approach to provide the feedback.

Both setup are described in the following sections.

2.1 Original bench

The original configuration is shown in *figure 1*. It includes a stainless steel ball which is free to move on a touchpad panel, hinged in the center, that can be inclined in the two directions by two servomotors powered by a power unit. All the previously mentioned components are part of the Acrome Robotics "2D ball balancing table" set [1]. The sensed position of the ball is read by a myRIO microprocessor board and sent to the PoliArd unit, based on an Arduino microcontroller, which is able to communicate with PoliScope interface on the PC for live feedback visualization. The controller is designed in Simulink and then deployed to the PoliArd. The control action is transmitted to the myRIO board that generates the corresponding PWM signal to feed the servomotors through the power unit. This schematic is resumed in *figure 2*. A dedicated PoliArd library is needed in Simulink to make it communicate both with PoliScope interface and the PoliArd unit.
As previously mentioned, this setup shows some relevant problems or non-optimal configurations:

- PoliArd architecture is quite old and not up to date, requiring an old version of MatLab (2015a) to work. Moreover, it mounts an Arduino microcontroller with a limited computational power, that doesn't allow to implement computationally demanding control logics.
- The presence of the myRIO board seems unnecessary and increases the length and complexity of the hardware connection chain. However, it is required for communicating with the power unit and for reading touchpad data, whose drivers are not accessible.
- Finally, some malfunctioning were noticed during the testing, mainly some defecting cables that needed to be hold by hand and one of the servomotors joint which slipped when too much torque was applied.

Due to the complex and not-so-accessible communication chain that makes difficult to understand how signals are converted, transmitted and processed, the calibration for the "original bench" was carried on by simply comparing measurements on the table with the corresponding signal readings on PoliScope interface in the PC.

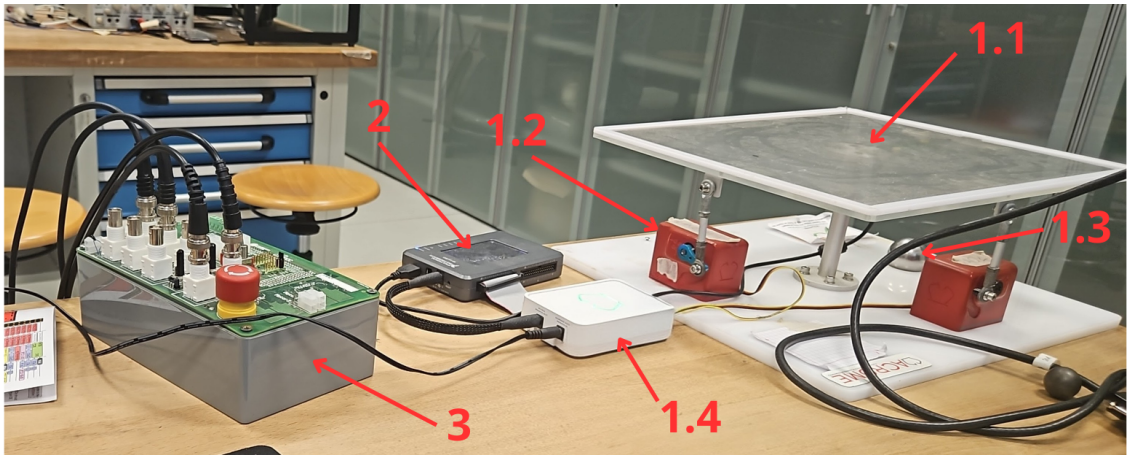

Figure 1: "Original bench" components. (1) *Acrome 2D ball balancing table set with* (1.1) *touchpad surface,* (1.2) *servomotors,* (1.3) *steel ball and* (1.4) *power unit;* (2) *myRIO microprocessor board;* (3) *PoliArd unit (based on Arduino microcontroller).*

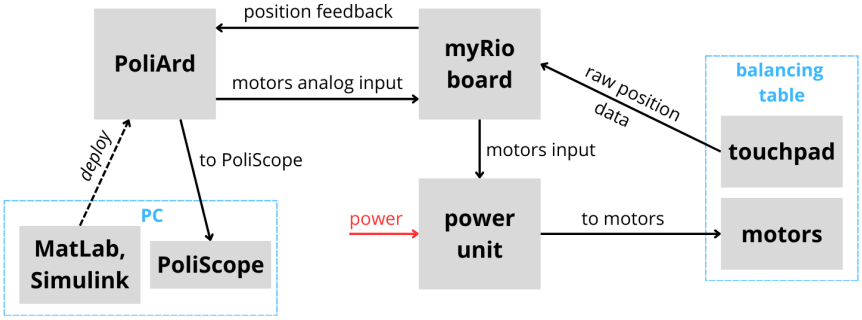

Figure 2: "Original bench" scheme

Touchpad panel calibration Concerning the touchpad calibration, raw position data (x_{bit} and y_{bit}) span in a range between 0 and 4095 representing the voltage that flows from myRIO board to PoliArd ($0 - 10$ V). In order to use these data for a feedback control, we need to turn them into the measurable, physical quantity that they represent (x and y coordinates in meters). To achieve this, a series of measurements were taken by placing the ball at various positions on the table and recording the values displayed on PoliScope for both directions. We realized that the trend of the points was strongly linear, and as a result, we decided to use a first degree polynomial approximation as a tool for converting bits to meters. The conversion formula are the following:

$$x_m = \frac{0.13}{3425 - 2050}(x_{bit} - 2050)$$

$$y_m = \frac{0.10}{3425 - 2050}(y_{bit} - 2050)$$

Notice that the origin of the physical reference frame was set in the center of the table, which corresponded to $x_{bit} = y_{bit} = 2050$. It's also important to mention that the touchpad panel wasn't able to sense the ball in its marginal areas, so we conservatively limited the operative region to $x = \pm 0.13\ m$ and $y = \pm 0.10\ m$.

Servomotors calibration Passing to the calibration of motor angles, we decided to use the equation

$$\alpha = sin^{-1}\left(\frac{h}{b}\right) \rightarrow h = b \cdot sin\alpha$$

derived from the model that will be described in section 3. In that model, the motor angle α is related to the vertical displacement h of the table edge. By varying the input signal with a potentiometer in the PoliArd and comparing its value (from the PoliScope) with the corresponding measurements of h, it was possible to find the relationship between the two, as shown in *figure 3*.

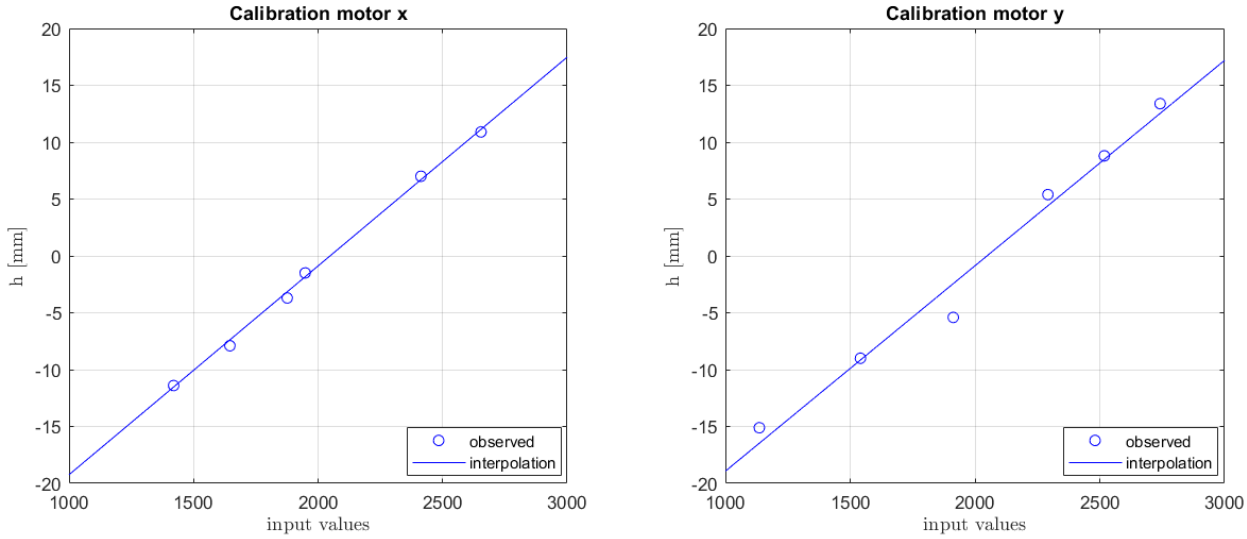

Figure 3: Motors angles calibration data

Again, we decided to use a linear relation as we are aiming at small angles of the table as we will discuss later in the report. Finally, using interpolation data and the previous formula, the following conversions between the motor angle and the required input are obtained:

$$input_x = \frac{b \cdot sin\alpha_x}{1.8358 \cdot 10^{-5}} + 2048$$

$$input_y = \frac{b \cdot sin\alpha_y}{1.8061 \cdot 10^{-5}} + 2048$$

2.2 New bench

The new balancing table (*figure 4*) was built using MDF panels and 3D printed PLA parts. A printed universal joint allows the table to tilt in both directions, while two servomotors control the inclination. This time, the position of the ball is tracked by a camera: a smartphone above the table sends raw video data through USB connection to the PC, exploiting DroidCam application. Here, these data are processed by Python, using OpenCV library, and the coordinates of the ball are sent to the Texas Instruments Launchpad F28069M microcontroller [2], where the control logic has been previously deployed from Simulink. The board outputs the PWM signal for the motors, that are powered by an

external power supply. Notice that while the system is running, the open serial communication between the microcontroller and the PC allows for changing live some parameters directly from Python (such as controller gains, for example), and it is also exploited to send back data (in particular motors input angles) generated by the TI Launchpad, that wouldn't be know otherwise. This schematic is resumed in *figure 5*. The dedicated c2000 Simulink library is needed to configure the TI Launchpad (serial communication, PWM generation, board's pins, etc).

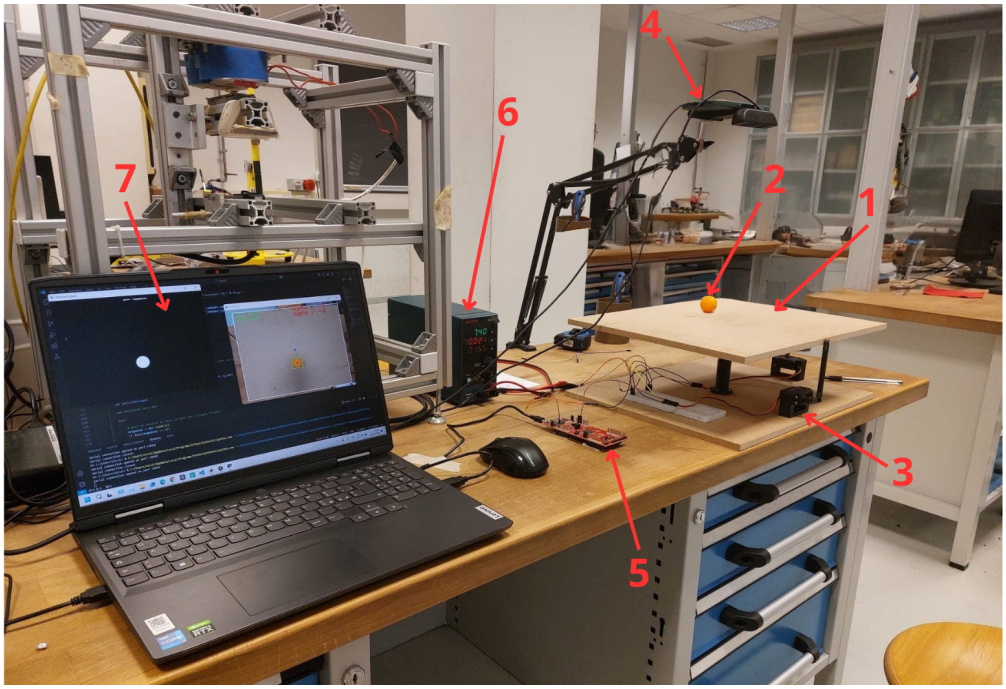

Figure 4: "New bench" components. (1) *MDF table;* (2) *ping pong ball;* (3) *Miuzei servomotors;* (4) *camera (smartphone);* (5) *TI Launchpad F28069M microcontroller;* (6) *power supply;* (7) *PC.*

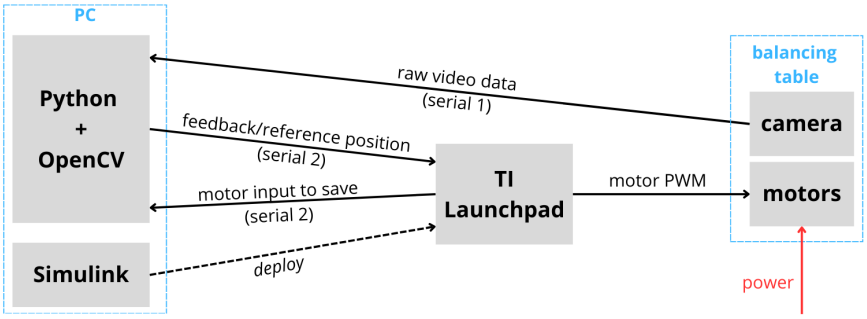

Figure 5: "New bench" scheme

For the "new bench", since we perfectly knew what input/output each component required, it was possible to carry on a more precise calibration.

Servomotors calibration The two servomotors work with a duty cycle between 2.5% and 12.5% on a 50 Hz period, corresponding to -90^o and $+90^o$ motor angles: by setting properly the dedicated

ePWM Simulink blocks of the c2000 toolbox (configuration was based on [3]. Details are not presented here), the TI Launchpad is able to generate and send the required PWM signals (that can also be monitored). Obviously, the correct conversion between the desired angle and the corresponding duty cycle value must be implemented in Simulink as well.

Camera calibration and ball tracking As previously mentioned, a smartphone's camera is exploited in the "new bench" to film the ball while it is moving. The camera is positioned above the table, on a fixed support, and it is manually focused at the correct distance. Raw frames are sent to the PC at 30 FPS with a resolution of 640x480. These parameters were chosen as they showed a good trade off between resolution and data speed (from Python it was possible to monitor the receiving frequency, that was indeed around 30 frames per second).

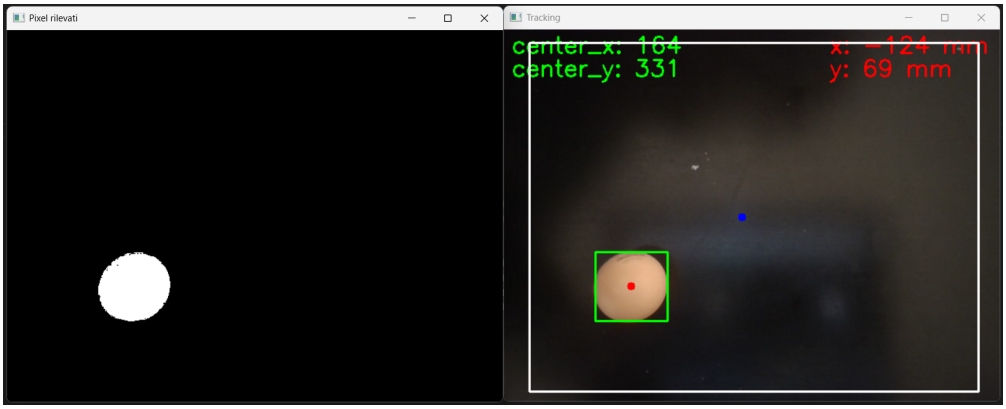

Figure 6: Image processing and ball tracking in the "new bench". On the left, the binary image after the static and the HSV masks have been applied; on the right, tracking results. The physical reference frame (red coordinates) is centered on the blue dot (center of the table) and has positive x going to the right and positive y going down, while the pixel reference frame (green coordinates) is centered on the top-left corner of the whole image. Notice that this was just a calibration test on a black surface: for results on the actual bench see figure 7).

In the PC, Python is used with OpenCV for image processing. First, a static mask is applied to the frame to only take into account for the area where the ball can move: this helps avoiding external objects of the same colour of the ball to be detected and it has to be set by hand, until mask's contour match table's edges. It is also possible to apply an additional Gaussian filter for noise attenuation. After that, image is converted from RGB to HSV, and another mask is defined to extract the desired hue, saturation and value. In our case the ball was orange on a light brown background, but these parameters were also tuned by trying to reduce the effects of shadows and reflections.
The resulting binary image is finally used to find the contours of the ball and compute the coordinates of its center (in terms of pixels) with OpenCV built-in functions. A final conversion of the position from (x_{pixels}, y_{pixels}) to (x_m, y_m) is calibrated by simply positioning the ball in the center of the table and in some other known spots and looking at the corresponding pixel coordinates. Data are now ready to be sent to the microcontroller.
Some passages of this procedure are shown in *figure 6* and *figure 7*.

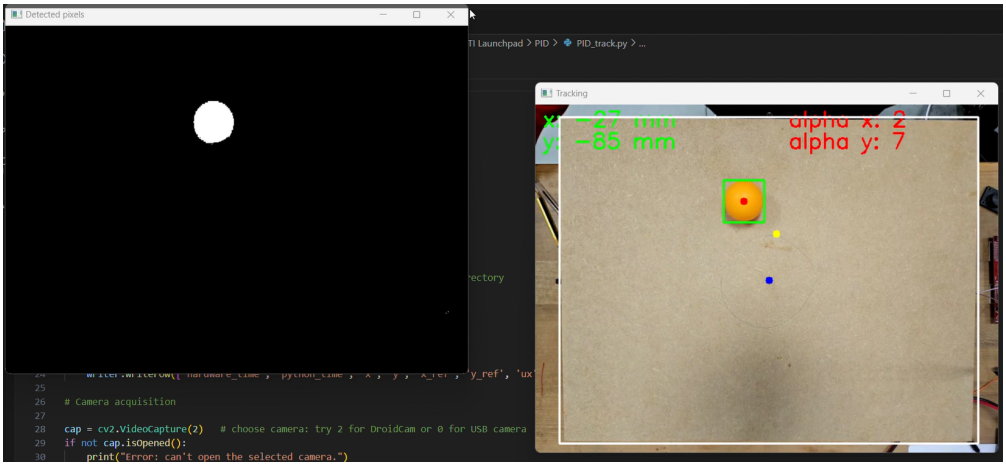

Figure 7: Image processing and ball tracking in the "new bench", here shown during a test, where the ball is trying to follow a moving reference (yellow dot). In this case, the motors angles (in red) are displayed as well.

3 Model of the system

In this section, we present the modeling approach for the two-degree-of-freedom (DoF) system and derive the equations of motion (EoM) using Lagrange's equations, based on the physical characteristics of the system. In the physical setup, the actuators (motor x and motor y) are rigidly mounted to the base plate and connected to the table via links with ball joints on both sides. The table itself is attached to the base plate through a rotary joint that provides two degrees of freedom. The 3D scheme of the system is presented in *figure 8*.

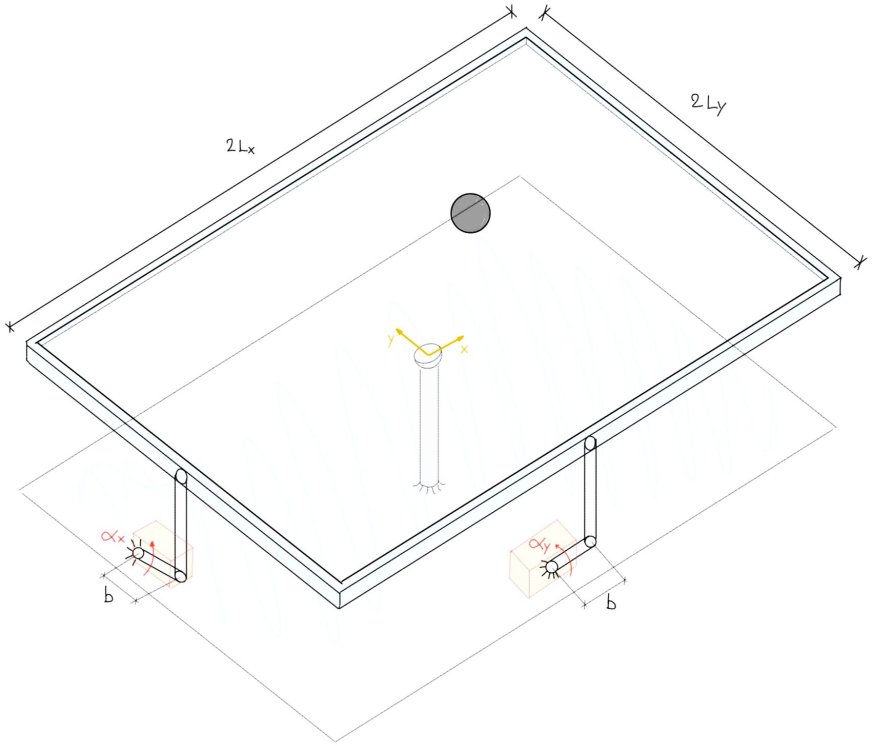

Figure 8: 3D scheme of the balancing table

3.1 Assumptions and data

To derive a mathematical model for this system, we make the following assumptions:

1. The ball can freely roll on the surface of the plate without slipping.
2. Both the table and the links are considered massless.
3. The motion in the x and y directions are fully decoupled.
4. The inclination θ of the table remains small during the motion of the ball.
5. The effects of damping due to the ball rolling on the table and elasticity of the joints are negligible.

Taking advantage of the third hypothesis we proceed to analyze the system and write the EoM of the rigid ball looking sideways so as to isolate individual DoF. The simplified setup scheme is shown in *figure 9*. In this diagram, b represents the motor rod, α_i is the motor angle controlling direction i, h_i is the vertical displacement of the table edge due to the motor rotation and θ_i is the angle of the table with respect the horizon. The steel ball has a mass m, an inertial moment J and a radius R and it is considered to be in the origin when it is above the hinge of the table.

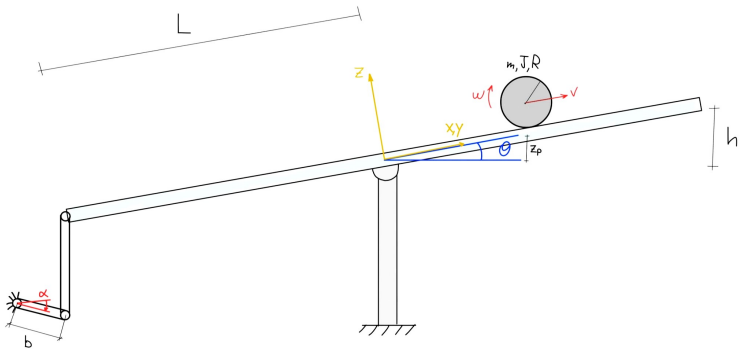

Figure 9: 2D scheme of the balancing table

The system parameters and the values for the two different benches are reported in Table 1.

Symbol	Original bench	New bench
$b\ [m]$	0.0245	0.03
$L_x\ [m]$	0.182	0.225
$L_y\ [m]$	0.150	0.175
$m\ [kg]$	0.264	0.004
$J\ [kg \cdot m^2]$	$4.224 \cdot 10^{-5}$	$2.56 \cdot 10^{-6}$
$R\ [m]$	0.020	0.040

Table 1: Data of the system

3.2 Equation of motion

There are several ways to derive the equations of motion, however, as anticipated, we'll use here Lagrange's equations. The free coordinates of the system are x and y while α_x and α_y are the system inputs. We can start by writing all the energy contributions: the kinetic energy of the ball can be expressed as:

$$T = \frac{1}{2}mv^2 + \frac{1}{2}J\omega^2$$

where the linear velocity of the ball is $v^2 = \dot{x}^2 + \dot{y}^2$ while the angular velocity $\omega^2 = \frac{v^2}{R^2} = \frac{\dot{x}^2+\dot{y}^2}{R^2}$. Thus, the kinetic energy becomes:

$$T = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2) + \frac{1}{2}\frac{J}{R^2}(\dot{x}^2 + \dot{y}^2) = \frac{1}{2}\left(m + \frac{J}{R^2}\right)(\dot{x}^2 + \dot{y}^2)$$

The potential energy can be written as:

$$V = mgz_p$$

where the vertical coordinate of the ball can be expressed as $z_p = x\ sin(\theta_x) + y\ sin(\theta_y)$, from simple geometrical considerations. Thus the potential energy becomes:

$$V = mgx\ sin(\theta_x) + mgy\ sin(\theta_y)$$

The dissipative function, having assumed that the damping due to the roll is negligible, is just $D = 0$. Moreover, being the ball free to move without any force applied, also the lagrangian component of the external forces is null, $Q = 0$. We can then apply the Lagrange equation for the two free coordinates (here only equations in x are reported).

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{x}}\right) - \frac{\partial T}{\partial x} + \frac{\partial V}{\partial x} + \frac{\partial D}{\partial \dot{x}} = Q$$

This results in the following equation of motion:

$$\left(m + \frac{J}{R^2}\right)\ddot{x} + mg\ sin(\theta_x) = 0 \tag{1}$$

Finally, we can link the the table rotation with the angle of the motors using geometrical relations:

$$L_x sin(\theta_x) = -b\ sin(\alpha_x)$$

Thus equation (1) becomes:

$$\left(m + \frac{J}{R^2}\right)\ddot{x} - mg\frac{b}{L_x}sin(\alpha_x) = 0$$

Bringing the input term to the right-hand side of the equation and dividing both members for the coefficient of $\ddot{x}$ we obtain:

$$\ddot{x} = \frac{mg}{\left(m + \frac{J}{R^2}\right)}\frac{b}{L_x}sin(\alpha_x)$$

The same can be done also for the second DoF:

$$\ddot{y} = \frac{mg}{\left(m + \frac{J}{R^2}\right)}\frac{b}{L_y}sin(\alpha_y)$$

To lighten the notation we can define the following coefficients:

$$C_x = \frac{mg}{\left(m + \frac{J}{R^2}\right)}\frac{b}{L_x} \quad \text{and} \quad C_y = \frac{mg}{\left(m + \frac{J}{R^2}\right)}\frac{b}{L_y}$$

Each equation can be rewritten in state space by defining $x_1 = x$, $x_2 = \dot{x}$ and $y_1 = y$, $y_2 = \dot{y}$. The set of equation can be rewritten in matrix form as:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ C_x \end{bmatrix} sin(\alpha_x)$$

$$\begin{bmatrix} \dot{y_1} \\ \dot{y_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ C_y \end{bmatrix} sin(\alpha_y)$$

Thus, the system is linear with respect to the state while it is not linear with respect to the control input. Moreover, defining the following vectors:

$$\mathbf{z_x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{and} \quad \mathbf{z_y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

and the characteristic matrices of the system as

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$\mathbf{B_x} = \begin{bmatrix} 0 \\ C_x \end{bmatrix} \quad \mathbf{B_y} = \begin{bmatrix} 0 \\ C_y \end{bmatrix}$$

we can calculate the reachability and observability matrices:

$$\mathbf{K_{Rx}} = \begin{bmatrix} \mathbf{B_x} & \mathbf{B_x A} \end{bmatrix} = \begin{bmatrix} 0 & C_x \\ C_x & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{K_{Ry}} = \begin{bmatrix} \mathbf{B_y} & \mathbf{B_y A} \end{bmatrix} = \begin{bmatrix} 0 & C_y \\ C_y & 0 \end{bmatrix}$$

$$\mathbf{K_O} = \begin{bmatrix} \mathbf{C}^T & \mathbf{A}^T\mathbf{C}^T \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Being all of them full rank, the system is both reachable and observable for both degrees of freedom. Finally, we can also rewrite the system in the following compact form:

$$\dot{\mathbf{z}}_\mathbf{x} = \mathbf{A}\mathbf{z_x} + \mathbf{B_x} sin(\alpha_x)$$

$$\dot{\mathbf{z}}_\mathbf{y} = \mathbf{A}\mathbf{z_y} + \mathbf{B_y} sin(\alpha_y)$$

4 Control logics

Different controllers were developed during the course of the laboratory. First, PD and PID regulators were used to both stabilize the ball in a certain reference position and also to track a predefined trajectory. Then, different variations of LQRs were tested for the same purposes. Pole placement was implemented as well for stabilizing the ball. The main setup used for these tests was the "original bench", but we have also implemented some of these control logics with the "new bench" for performance comparison. However, unless otherwise specified, we will refer to the old setup as the one used for the experiments.
Notice that, as already mentioned, x and y are decoupled, so separate controllers were used for each degree of freedom. Moreover, all the tuning processes, testing results, ect are only reported for the x coordinate in this report, since the procedures are exactly the same, just with different data. Finally, recall that the available physical outputs (measurements) of the system are positions x and y of the ball.

4.1 PD and PID

First, we tried to develop an output feedback control on the position. During these tests, since the system operates at lower frequencies than the noise, a simple low pass filter was used on the measurements, tuned experimentally as:

$$H(s) = \frac{20}{s + 20}$$

Notice that this is not the optimal filter choice: a more refined approach will be discussed in section 4.3.1. Moreover, a saturation of $\pm 45^o$ was imposed on the motors angles.
It's easy to verify that a simple proportional controller is not able to stabilize the system, as it can be seen from its root locus in *figure 10*, and it can be shown that also a PI regulator is not enough. Thus, we started by tuning a PD.

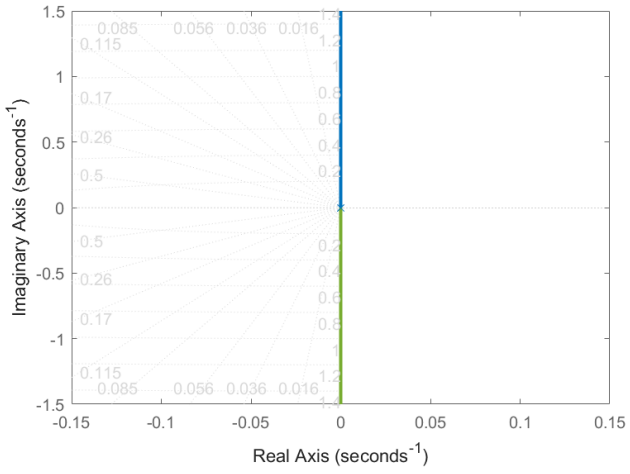

Figure 10: Root locus for the proportional controller

4.1.1 Ball stabilization

At the beginning, an analytical tuning of the gains of the PD controller is performed by imposing a certain limit on settling time and overshoot for the step response. This allows us to obtain a reference set of parameters, which will serve as the starting point for the experimental tuning on the real system. Our goal is to achieve an overshoot below 20% and a settling time under 3 seconds. Moreover, to reduce the effect of noise at high frequency an attenuation of $40\ dB$ is required after $100\ rad/s$.
To formalize these requirements, we can express them mathematically and use them as constraints for the Bode diagram. Let's start with the overshoot:

$$S_{\%} = 100e^{\frac{\pi\xi}{\sqrt{1-\xi^2}}} < 20 \quad \rightarrow \quad \xi < \sqrt{\frac{1}{1+\left(\frac{\pi}{ln(0.2)}\right)^2}} \approx 0.46$$

Furthermore, the damping, ξ, is related to the phase margin, Ψ_m, through the following equation:

$$\xi = sin(\frac{\Psi_m}{2}) < 0.46 \quad \rightarrow \quad \Psi_m > 2sin^{-1}(\xi) \approx 54^\circ$$

Then, we can impose the condition on the settling time:

$$T_{s,\varepsilon} = \frac{4.6}{\xi\omega_c} < 3\ s \quad \rightarrow \quad \omega_c > \frac{4.6}{T_{s,\varepsilon}\xi} \approx 3.4\ rad/s$$

Finally, the condition of noise reduction is enforced:

$$|F| = \left|\frac{L}{1+L}\right| \approx [\omega >> \omega_c] \approx |L| < -40\ dB$$

Once these limits are defined, we can proceed in MatLab with the actual tuning of the PD controller parameters, K_p, K_d and N. To do this, we utilize the functions `asymp(L)` and `margin(L)`, where L represents the open-loop transfer function of the system. These functions allow us to iteratively adjust the gains to meet the desired performance specifications. The open-loop transfer function L is given by the product of the system transfer function, $G = \frac{C_{x,y}}{s^2}$, and the controller transfer function, $R = K_p + \frac{K_d s}{1+\frac{s}{N}}$.
Using this procedure, we obtain the following initial gains: $K_p = 5$, $K_d = 3$ and $N = 15$. The resulting Bode diagram is shown in *figure 11*.

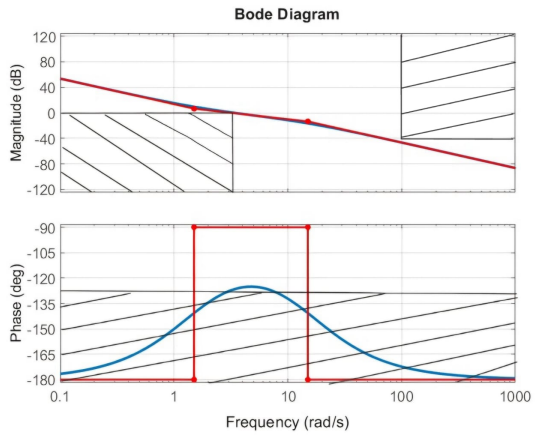

Figure 11: Bode diagram for the analytical tuning. The black areas show the limits due to the imposed overshoot ($< 20\%$) and settling time ($< 3\ s$).

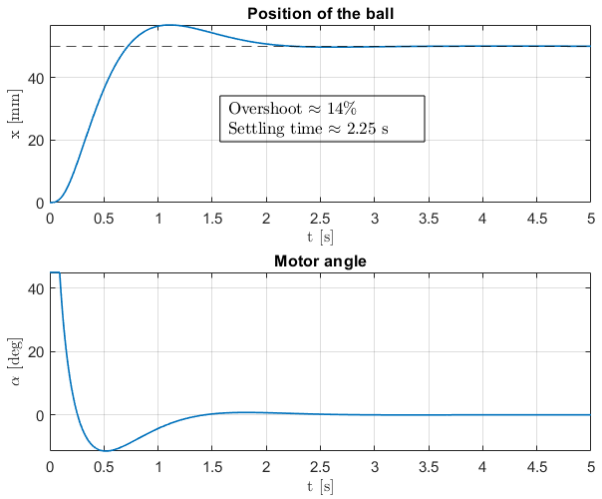

Figure 12: Simulation of the system with a PD controller using the gains tuned analytically: $K_p = 5$ and $K_d = 3$.

These results were simulated in Simulink using the nonlinear model of the system (*figure 12*). As we can see, the overshoot and the settling time are quite close to the expected ones, with possible discrepancies mainly due to the fact that the analytical tuning was assuming a linear system and a non saturated PD controller.

Experimental testing The previously computed gains were used as first guess for the experiments and then refined by performing different tests, also including an integral contribution (PID). The reference position was $x_{ref} = 50\ mm$, starting from the origin, and an anti-windup filter was used when necessary (PID). Some results are shown in *figures 13,14,15,16*.

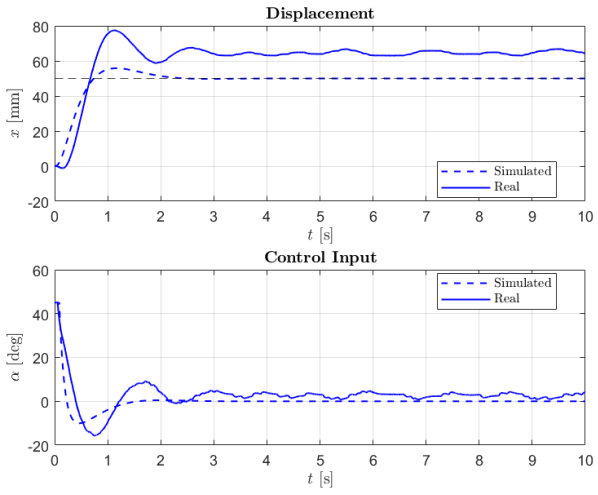

Figure 13: PD with $K_p = 5$, $K_d = 3$

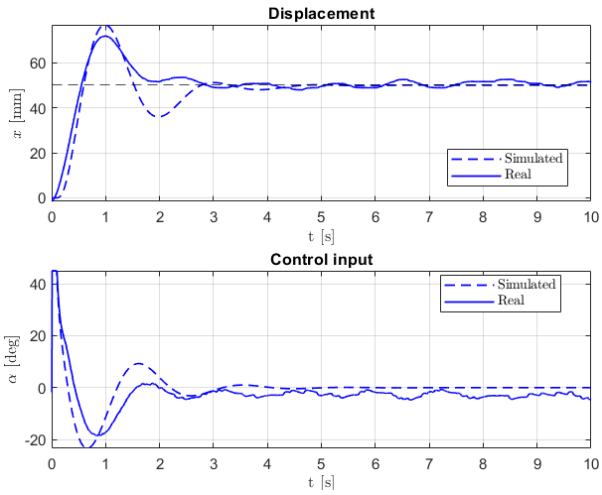

Figure 14: PID with $K_p = 5$, $K_i = 3$, $K_d = 3$

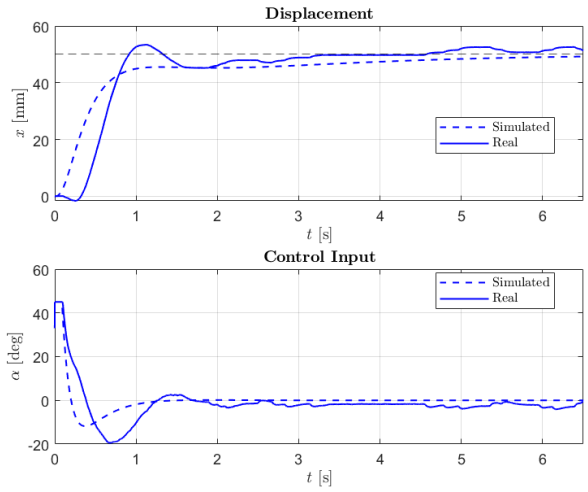

Figure 15: PID with $K_p = 4.5$, $Ki = 1.5$, $K_d = 3.5$

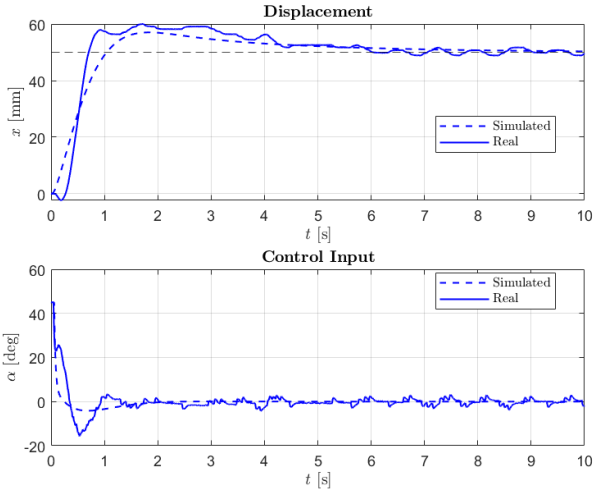

Figure 16: PID with $K_p = 6.4$, $K_i = 1.8$, $K_d = 4$

From the results of the tests, it can be noticed that the ball is successfully stabilized in all cases. However, some differences between the simulated and real system can be observed, and these are primarily due to disturbances, uncertainties and delays that are present in the real system, but not accounted in the model.
The PD regulator has a steady state error, as expected from theory, which is not present in the simulation since there we are assuming ideal conditions (*figure 13*). Adding an integral action eliminates the steady-state error observed in the PD controller. However, this improvement comes with a trade-off: the integral increases the overshoot and settling time, generating a slower response. Comparing the results from *figure 14* with *figure 15* we can notice that the first exhibits a higher overshoot and lower settling time, due to more aggressive K_p and K_i. The role of the derivative gain K_d is evident in *figure 16*, as increasing its value causes a reduction of the oscillations by enhancing the system damping, but also gives a more "scattered" control input (derivation amplifies noise).

Ball stabilization on the new bench Some tests were performed also on the new setup. The firmware deployed into the Launchpad board is shown in *figure 17*, while the Python code that communicates with it through serial port is not reported here.

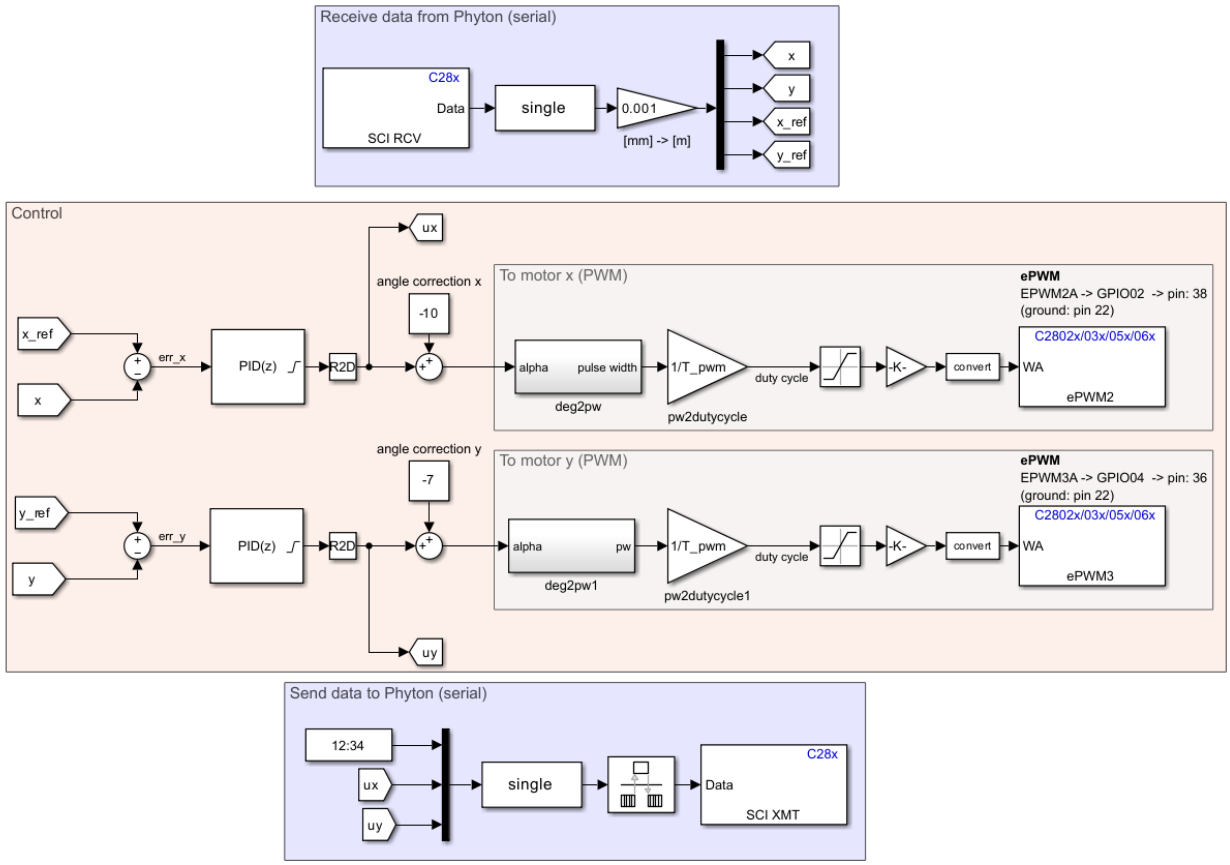

Figure 17: Simulink code for PID control to be compiled and deployed in the Launchpad board.

Also in this case, the gains for the PID have initially been chosen close to the ones used in the PoliArd setup, and then refined through experiments (even if not too much, due to lack of time). Some results, with the corresponding parameters, are shown in *figures 18, 19, 20, 21.*

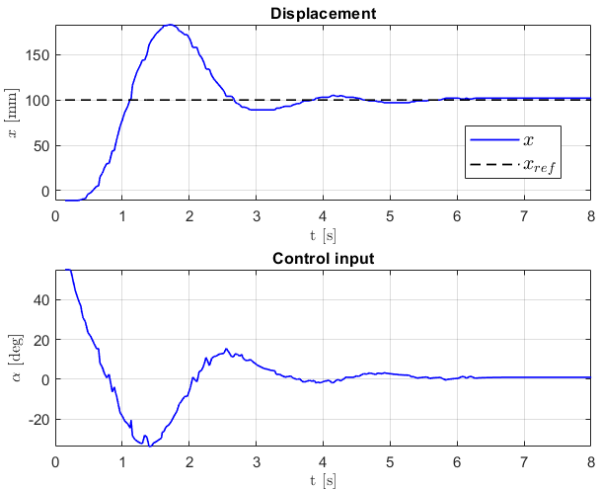

Figure 18: PID with $K_p = 2.6$, $K_i = 0.7$, $K_d = 0.9$

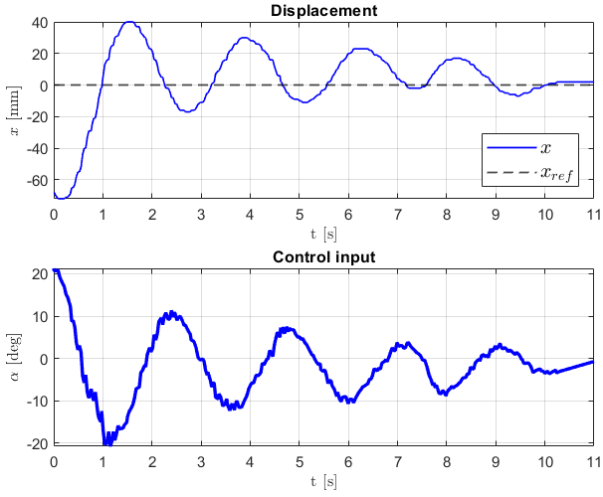

Figure 19: PID with $K_p = 2.6$, $K_i = 0.9$, $K_d = 1.75$

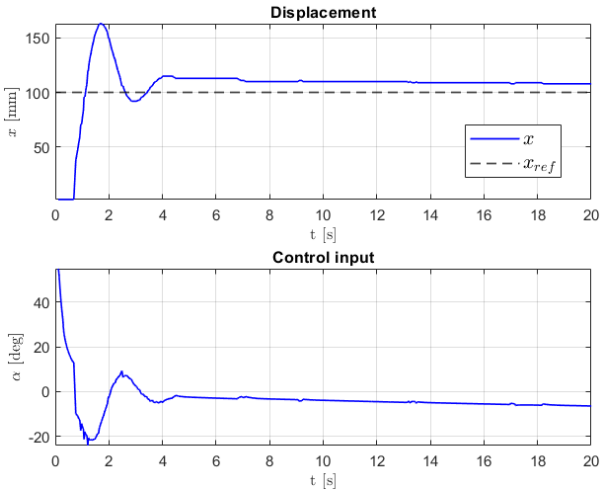

Figure 20: PID with $K_p = 2.6$, $K_i = 0.5$, $K_d = 0.9$

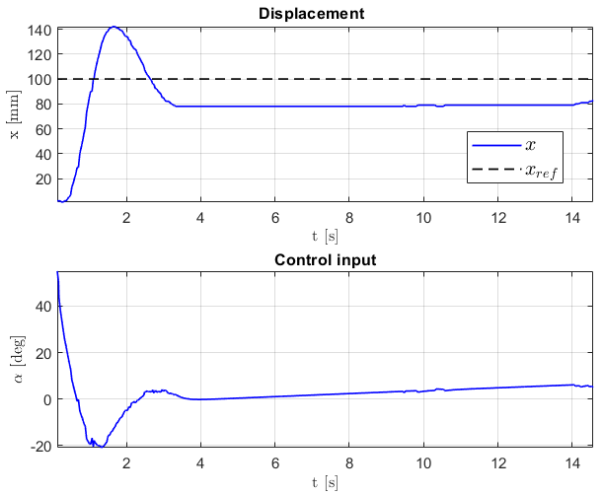

Figure 21: PD with $K_p = 2.6$, $K_d = 0.7$

The same considerations we did for the original bench still hold: for example, from *figure 19* it's clear that an high derivative gain amplifies noise, and that an high integral action slows down the response, increasing oscillations. Or again, the PD regulator in *figure 21* does not remove the steady state error. However, there is an additional observation that is worth mentioning. During the tests, we noticed that the ping pong ball was often stopping next to the desired position, but not exactly on it, even if the integral action was there. This behavior is due to the rough surface of the wooden panel where the ball moves, which is less smooth than the touchpad of the original bench. Looking at *figure 20*, it's clear that the ball doesn't exactly stops on the reference, and even if the integral action accumulates the error (in fact the motor angle slowly goes down), it won't move for a very long time (because it's "stuck"). One way to improve this, consists in giving the ball a very fast spinning motion on its vertical axis, reducing the friction with the table: plot in *figure 18* was obtained like this, and it shows a smoother displacement, that doesn't "stop" anywhere, having basically the same regulator as the one in *figure 20*.
As we can see, the performance doesn't appear to have improved, compared to the old setup (*figures 13, 14, 15, 16*), however there are few things we need to keep in mind. First of all, the new bench is bigger and heavier, enhancing nonlinear effects due to geometry. Then, as we will better explain in section 5, this new system has undergone less testing and tuning, and even the signals themselves are handled more roughly (no filters are used here, for example). Nevertheless, the ball still manages to stabilize in its desired position, the new bench seems to be more stable in general and it has never been observed to loose data or to slow down significantly, differently to the old setup. Finally, the position feedback through a camera allows for controlling the ball even if it is not touching the platform: the system, in fact, has been successfully tested many times with a bouncing ball.

4.1.2 Trajectory tracking

A second set of experiments is now analyzed, where the aim was to make the ball follow a predefined trajectory. The strategy simply consisted in passing a series of points to the controller as reference. These points were generated by the deployed code in the microcontroller, with the proper frequency and position.

At first, a circular path with radius $50\ mm$ and period $T = 6\ s$ was chosen, and different gains for the controller were changed live during the experiment, using potentiometers on the PoliArd (*figure 22*). Then, we performed a test on a similar circle but with $T = 10\ s$, fixing the gains to values that seemed to give the best performance, and we compared the result with a Simulink simulation (*figure 23*). In particular, we noticed that:

- The values of the gains to assure a decent tracking were slightly different from the ones found with the step response, and also they strongly depended on the specific experiment (frequency / distance at which the reference-points were provided, period of the trajectory, etc).
- For "slower" trajectories (higher period to complete one cycle), tracking was more accurate, as can be noticed comparing *figure 23* $(T = 10\ s)$ with *figure 22* $(T = 6\ s)$. We also observed that when the reference trajectory was "too slow", the ball tended to maintain a very irregular speed, visibly "jumping" to the next reference. This behavior could be reduced by changing the way reference points were provided.
- As we saw in section 4.1.1, derivative gain helps reducing the settling time, while integral action slows down the response. Thus, for this type of tracking problem, the first one improves the result (within the limits of stability), while the second one is not so desired. This can be easily observed in *figure 22*.

In general, as expected, PID control is not really meant for tracking, in particular with this very basic implementation. This is also confirmed by the simulation visible in *figure 23*, that is very close to the experimental results, but far from the reference circle, even if it doesn't include any noise or disturbance.

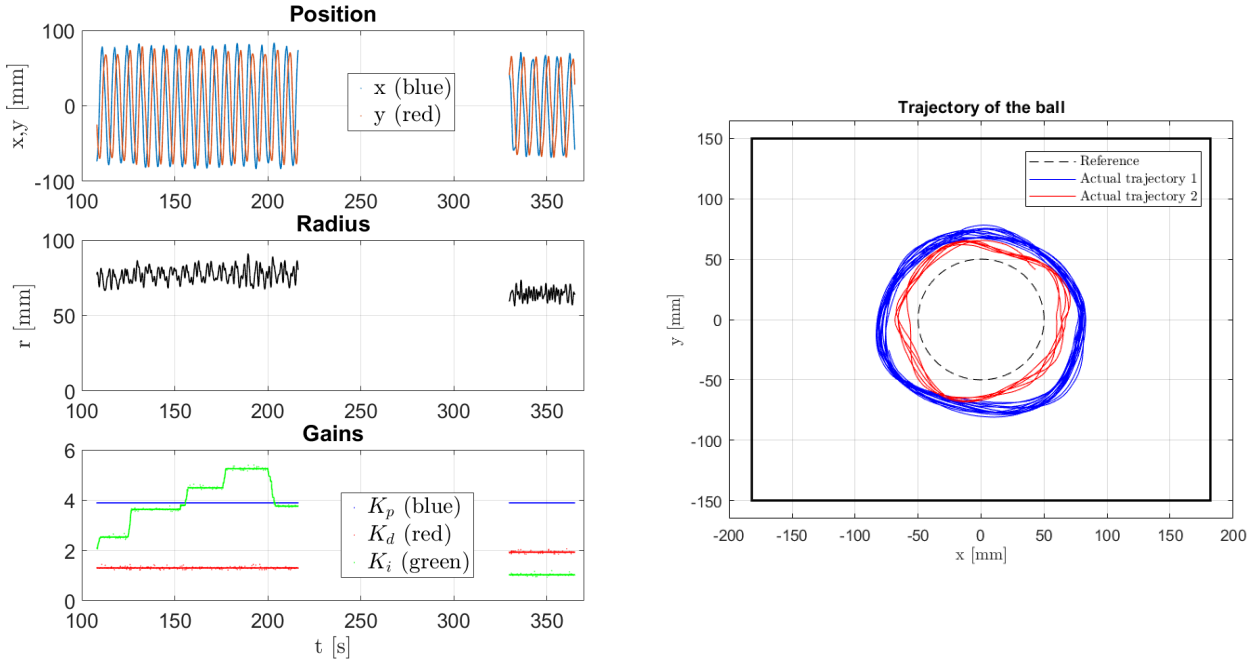

Figure 22: Circle trajectory tracking with original bench (extracted from a longer experiment). Period $T = 6\ s$. PID controller with various gains. In the figure on the right, the blue trajectory corresponds to the time history from 110 to 220 seconds, while the red one to the remaining part (from 330 to 370 seconds).

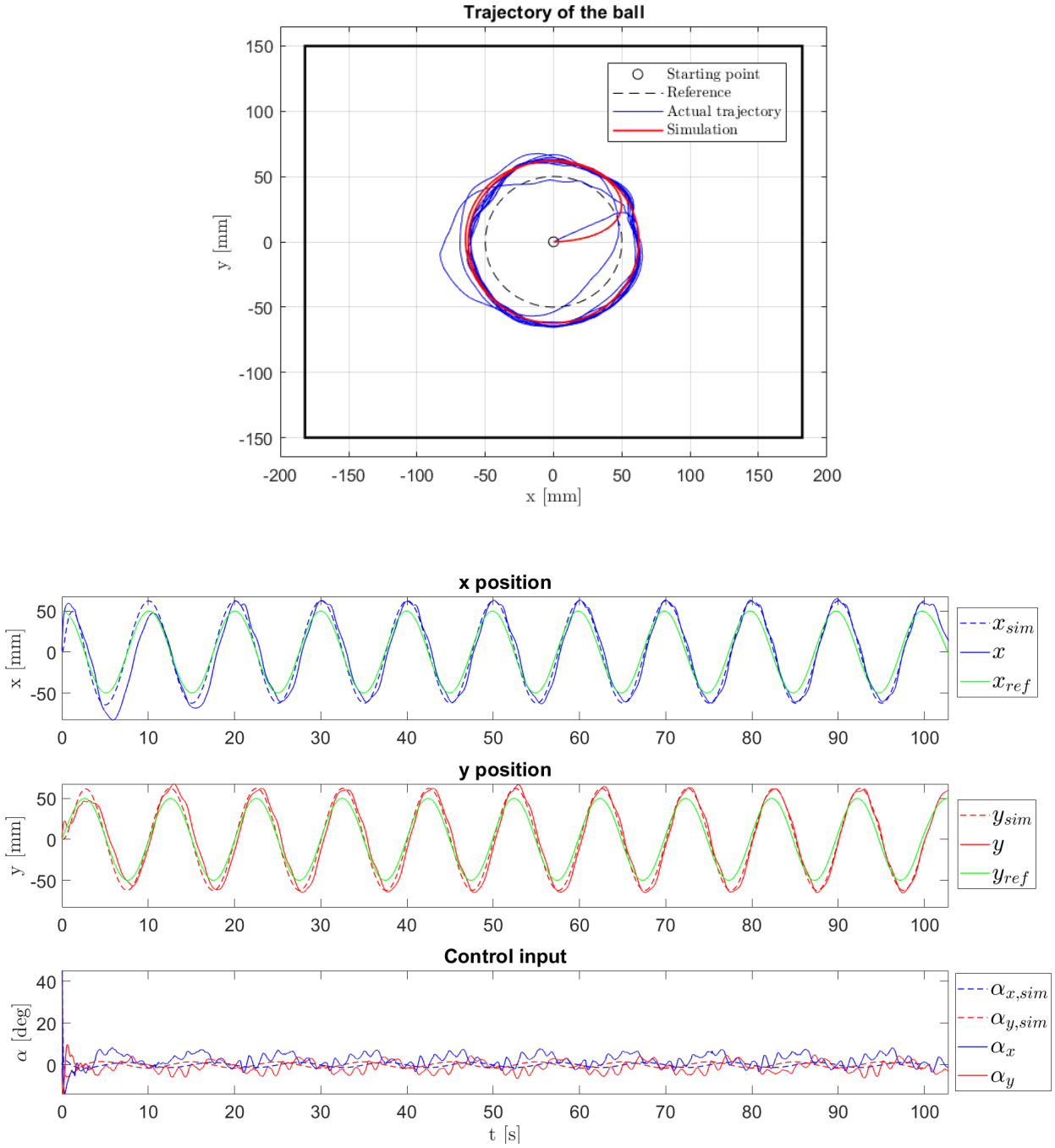

Figure 23: Circle trajectory tracking with original bench. Period $T = 10$ s. PID controller $K_p = 2$, $K_i = 0.5$, $K_d = 1.8$

As an additional option, we also implemented the possibility for the user to freely draw a certain trajectory, using `ginput` in MatLab, deciding the duration, and deploying it in the microcontroller. Different tests were made, but the outcome and the possible considerations was always very similar to what we already said for the circle. For the sake of curiosity, in *figures 24* and 25 two of those experiments are reported, both of them on the same "fish-shaped" trajectory. Once again, an higher derivative gain (and also proportional in this case), helps improving the tracking, which remains not

very accurate.

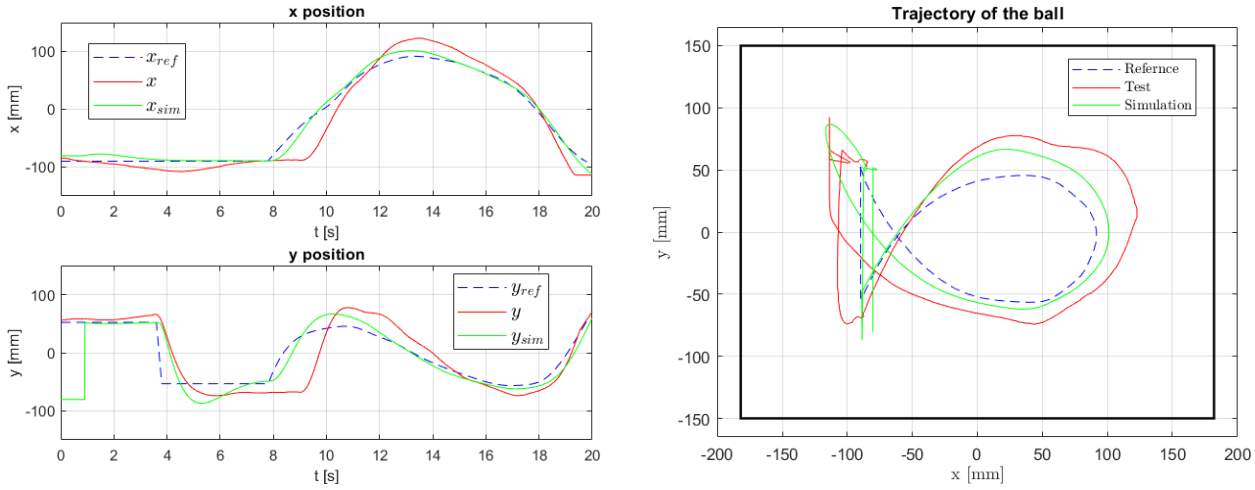

Figure 24: "Fish" trajectory tracking with original bench. Duration $T = 20$ s. PID controller $K_p = 2$, $K_i = 0.5$, $K_d = 1.8$

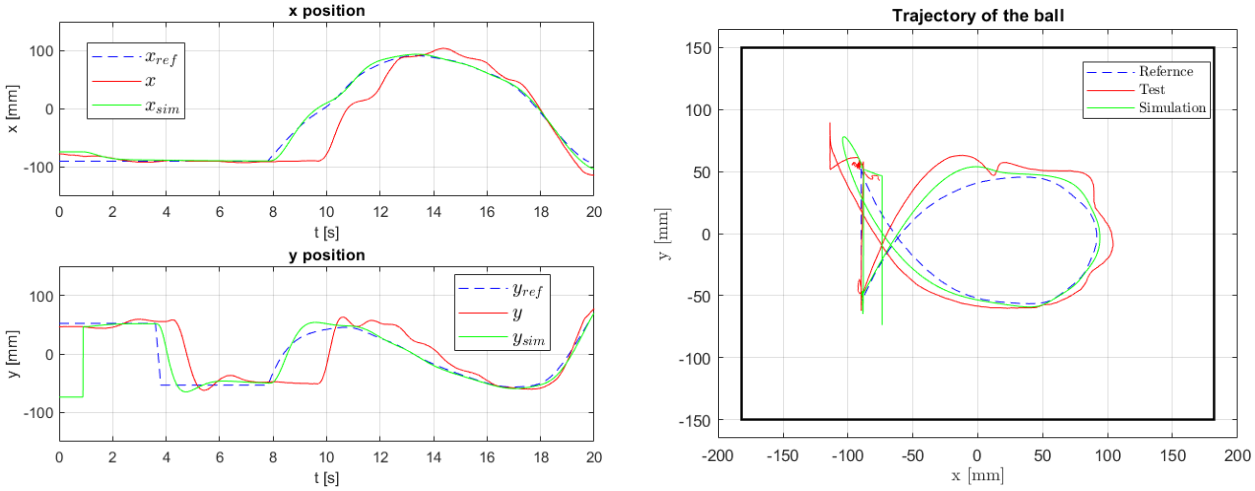

Figure 25: "Fish" trajectory tracking with original bench. Duration $T = 20$ s. PID controller $K_p = 5$, $K_i = 1$, $K_d = 3$. Notice there is a remarkable delay around $t = 8$ s, probably due to the loss of some data.

Trajectory tracking on the new bench The circular trajectory, with radius 50 mm and period 5 s, was also tested with the new bench setup, still using a PID controller. The firmware deployed into the Launchpad board is the same as in *figure 17*, while the Python code is different, but still not reported here.
The results of two experiments with different sets of gains are shown in *figures 26* and *27*. As we can see, similarly to what happened with the step response, also the tracking performance doesn't seem to be better than before (*figure 23*). Despite this, this bench seems to be slightly more robust with respect to sudden and intense perturbations. Finally, in this case the varying reference for the PID

track control is externally sent by Pyhton, and not internally generated by the microprocessor itself, providing a signal with a much more stable frequency.

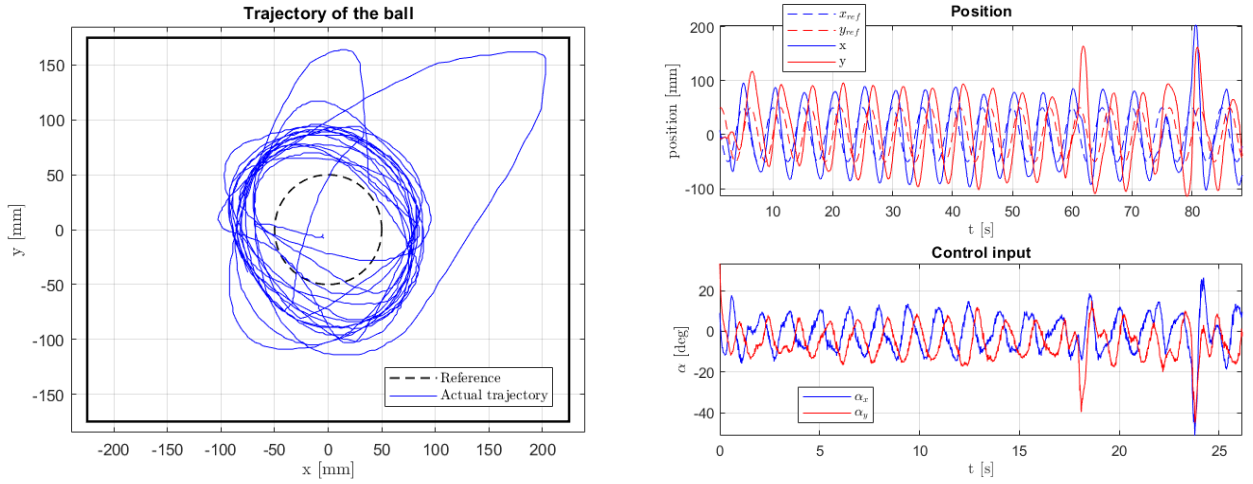

Figure 26: Circle trajectory tracking with new bench. Period $T = 5$ s. PID controller $K_p = 2.6$, $K_i = 0.7$, $K_d = 0.9$. Notice that at time $t = 18$ s and $t = 23$ s the ball was deliberately pushed, to test disturbances rejection and robustness of the system.

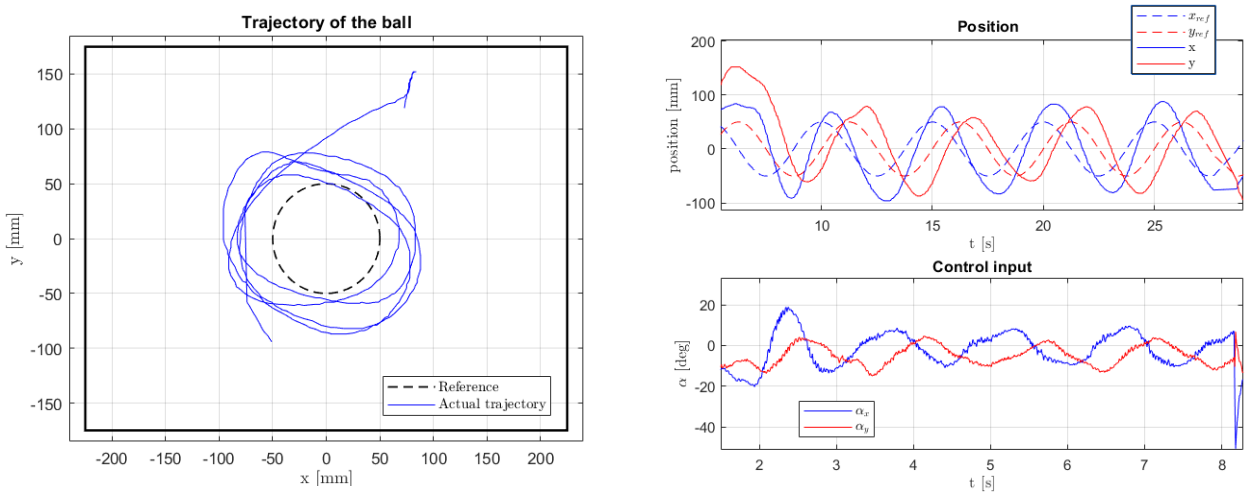

Figure 27: Circle trajectory tracking with new bench. Period $T = 5$ s. PID controller $K_p = 1.75$, $K_i = 0.6$, $K_d = 1.75$

4.2 Pole placement

Full State Feedback (FSF), also known as pole placement, is a control method that enables us to position the closed-loop poles of a system at desired locations in the complex plane.
To eliminate the steady-state error as well, we can consider an extended system, represented by the

following matrices:

$$\mathbf{A_{ex}} = \begin{bmatrix} \mathbf{A} & \begin{bmatrix}0\\0\end{bmatrix} \\ \mathbf{C} & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \mathbf{B_{x,ex}} = \begin{bmatrix} \mathbf{B_x} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ C_x \\ 0 \end{bmatrix} \quad \mathbf{B_{y,ex}} = \begin{bmatrix} \mathbf{B_y} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ C_y \\ 0 \end{bmatrix}$$

To apply the pole placement method, we must first verify the controllability of the extended system. This requires computing the controllability matrices:

$$\mathbf{K_{Rx}} = \begin{bmatrix} \mathbf{B_{x,ex}} & \mathbf{B_{x,ex}A_{ex}} \end{bmatrix} = \begin{bmatrix} 0 & C_x & 0 \\ C_x & 0 & 0 \\ 0 & 0 & C_x \end{bmatrix} \quad \text{and} \quad \mathbf{K_{Ry}} = \begin{bmatrix} \mathbf{B_y} & \mathbf{B_yA} \end{bmatrix} = \begin{bmatrix} 0 & C_y & 0 \\ C_y & 0 & 0 \\ 0 & 0 & C_y \end{bmatrix}$$

Clearly, the rank of the matrices is 3, thus the system is controllable. Moreover, we need also to check the equilibrium condition:

$$\Sigma = \begin{bmatrix} \mathbf{A} & \mathbf{B_{x,y}} \\ \mathbf{C} & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & C_{x,y} \\ 1 & 0 & 0 \end{bmatrix}$$

Again the rank is 3, thus the equilibrium equation is satisfied for every constant reference and the pole placement technique can be applied. Before proceeding further, we first check how the poles of the system have been shifted by the application of the analytically tuned PD controller. The results are illustrated in the following figure:

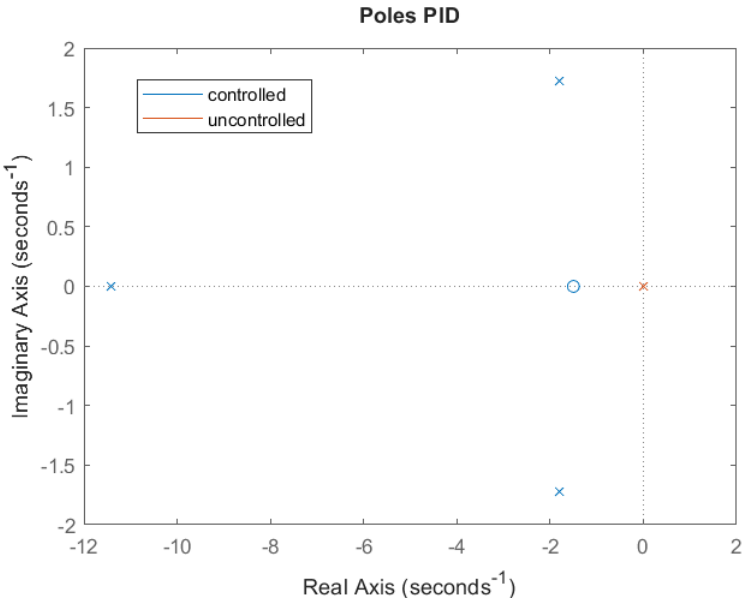

Figure 28: Poles of the PD controller

As we can see the poles have been moved from the zero to left hand side of the complex plane to guarantee stability. Moreover, two of them are complex and conjugates and this is coherent with the second order response obtained in the previous paragraphs. We were interested in understanding whether it was possible to achieve a response similar to that of a first-order system, rather than the

oscillatory behavior typically seen with PID controllers. To do this, we attempted to place all the poles along the negative real axis. However, through testing, we discovered that this was only possible by limiting the speed of the poles, which resulted in a slower system. After some trial and error, we obtained the following set of poles:

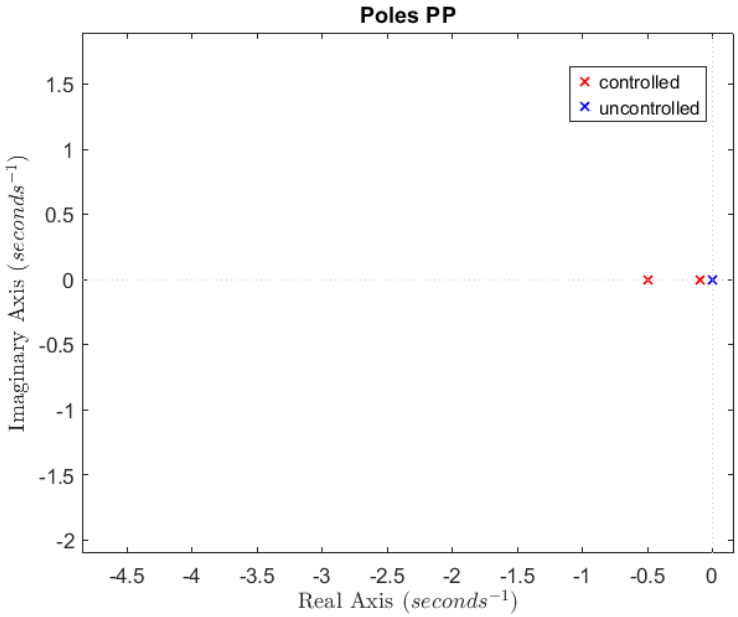

Figure 29: Poles of the pole placement controller

Using the MATLAB command `place`, it is possible to shift the poles toward the desired configuration and determine the appropriate gain to be used in the feedback control.Note that, since Pole Placement is a full-state feedback controller, an estimate of the velocity is also required for it to function properly. A Kalman filter was used for this purpose. As the Kalman filter is extensively used in the LQR controller as well, its design will be discussed in section 4.3.1. The result of one of the tests is shown in figure 30

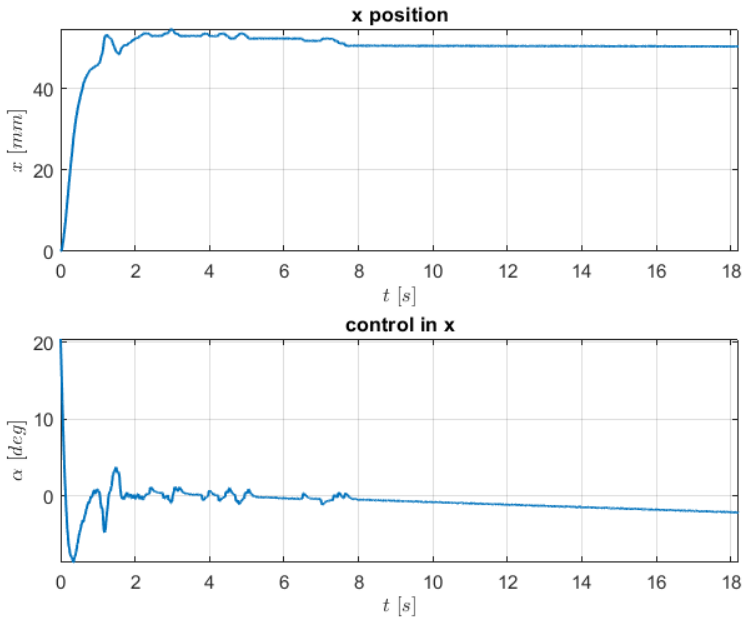

Figure 30: Pole Placement with gains $K_x = [3.05\ 5.59\ 0.25]$

As we can see, both the rise time and the settling time are longer than the one experienced with the PID controllers while the control action requires less effort for the system with respect the PIDs.

4.3 LQR

Another type of control logic we have implemented is the Linear Quadratic Regulator. At the beginning, an infinite-time LQR control was developed for stabilizing the ball around a certain position. Then we introduced an integral action (LQRI) to remove the steady-state error. Finally, trajectory tracking was tested as well with new type of regulator.
LQR is a full-state feedback controller, so we also need to estimate the velocity to make it work. A Kalman filter was designed for this purpose, since it gives the optimal estimation of the states ($x, \dot{x}$ and $y, \dot{y}$ in our case), also filtering noise on the measurements.

4.3.1 Kalman filter

As already mentioned, by using a Kalman filter we achieved more accurate estimates of the noisy measurements compared to the low-pass filter. Furthermore, it provides better estimations when handling disturbances. The estimator was designed as a steady-state Kalman-Bucy filter (here shown for the x direction, so $\mathbf{z} = [x \quad \dot{x}]^T$ is the state and x is the measurement):

$$\dot{\hat{\mathbf{z}}} = \mathbf{A}\hat{\mathbf{z}} + \mathbf{B}u + \mathbf{K_{obs}}\left(x - \mathbf{C}\hat{\mathbf{z}}\right)$$

$$\dot{\hat{\mathbf{z}}} = \mathbf{A}\hat{\mathbf{z}} + \begin{bmatrix}\mathbf{B} & \mathbf{K_{obs}}\end{bmatrix}\begin{bmatrix}u \\ x - \mathbf{C}\hat{\mathbf{z}}\end{bmatrix} \tag{2}$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are our linearized system's matrices, u the control input and the apex $\hat{}$ marks the estimated quantities. Kalman gain is found as

$$\mathbf{K_{obs}} = \mathbf{P_{obs}} \mathbf{C^T} \mathbf{R_{obs}^{-1}}$$

by solving the Algebraic Riccati Equation for $\mathbf{P_{obs}}$ (this can be done offline):

$$\mathbf{Q_{obs}} + \mathbf{A}\mathbf{P_{obs}} + \mathbf{P_{obs}}\mathbf{A^T} - \mathbf{P_{obs}}\mathbf{C^T}\mathbf{R_{obs}^{-1}}\mathbf{C}\mathbf{P_{obs}} = \mathbf{0}$$

with the following weighting matrices:

$$\mathbf{Q_{obs}} = \begin{bmatrix} 100 & 0 \\ 0 & 1000 \end{bmatrix} \qquad \mathbf{R_{obs}} = \begin{bmatrix} 0.5 \end{bmatrix}$$

The process noise covariance matrix $\mathbf{Q_{obs}}$, was intentionally set to higher values to account for the presence of unknown disturbances and uncertainties inherent in the system model. The measurement noise covariance matrix $\mathbf{R_{obs}}$ was set to an intermediate value, reflecting a balance between reliable and unreliable measurements.
Simulink implementation of the filter, consisting in Equation (2), is shown in *figure 32*. Notice that the LQR control requires $\mathbf{\Delta z} = \mathbf{z} - \mathbf{z_{ref}}$ as input, as we will see later, so the filter presented in the previous equations is intended to work with $\mathbf{\Delta z}$, Δu, Δx, etc (just a different notation).

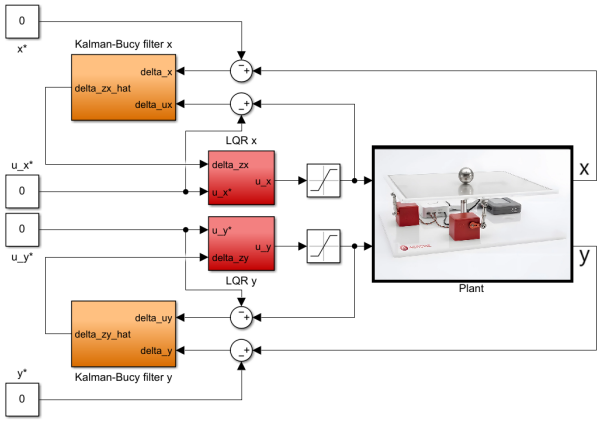

Figure 31: Schematic Simulink model of the LQR control. Kalman filter is expanded in figure 32, while the controller in figure 36 or 37

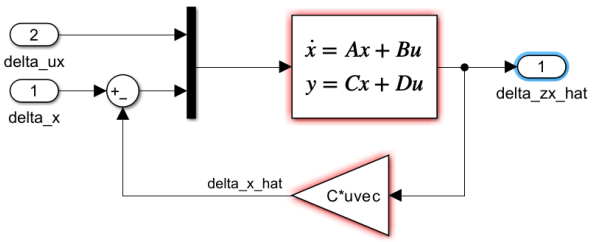

Figure 32: Kalman filter block. In the state-space block, B is actually $[\mathbf{B} \quad \mathbf{K_{obs}}]$ *and* $C = \mathbf{I}, \mathbf{D} = \mathbf{0}$

In *figure 33*, we can observe Kalman filter in action in a real test, effectively estimating the measured position and reducing noise. Unfortunately, due to the limited number of outputs allowed by the PoliScope interface, we didn't save any data of the unfiltered measurements when using the low-pass filter, so an experimental comparison between the two filters is not so easy. However, we performed two simulations introducing noises and disturbances in the system, and the results are illustrated in *figure 34*.
Both filters effectively reduce noise in the measurements, as evident in the plots. Nevertheless, KF exhibits superior performance by accurately estimating the position despite disturbances, due to its

integration of the system model into the estimation process. In contrast, the LPF relies solely on noise suppression, leading to less precise results and greater overshoot in the position response (top plot). This advantage is also reflected in the control input (bottom plot), where the KF provides smoother and more stable control signal with fewer oscillations. This demonstrates that the KF not only enhances estimation accuracy but also improves the overall control quality, ensuring a more reliable and efficient system performance. Note that we have used an additive disturbance on the input, which simulates uncertainties on motor's angles.

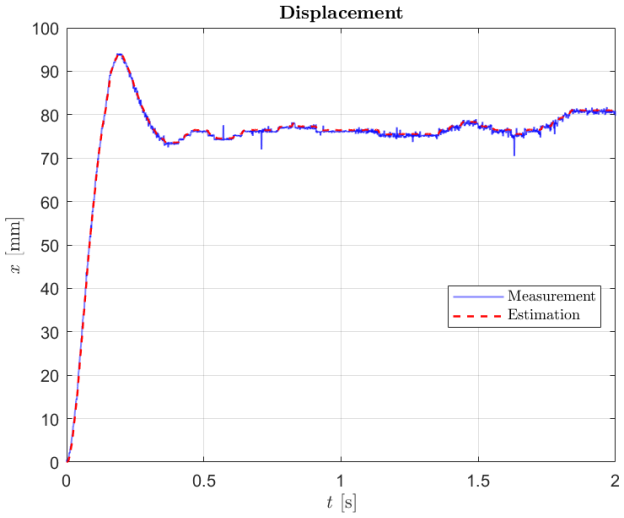

Figure 33: KF estimation vs measured position in a real test

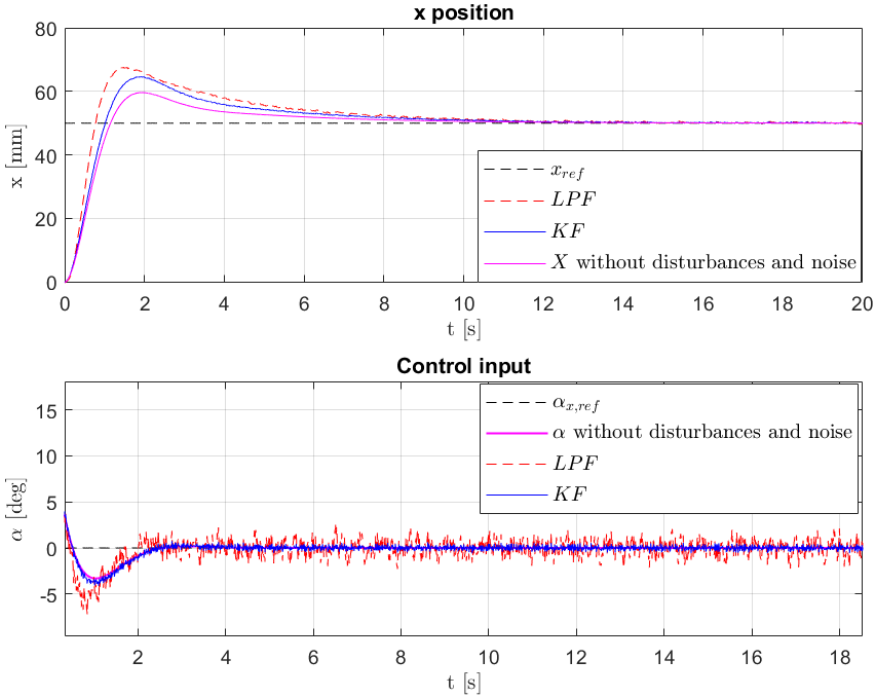

Figure 34: Simulated KF vs LPF

4.3.2 Infinite time control

We designed an infinite time horizon LQR, since we are interested in stabilizing the ball at a given position and null final control input and velocity. It is an optimal state feedback controller, where the gain matrix $\mathbf{K}$ is such that control action $\mathbf{u}$ minimizes the following cost functional:

$$J = \int_0^\infty \frac{1}{2} \left[\mathbf{z}^T(t)\, \mathbf{Q}\, \mathbf{z}(t) + \mathbf{u}^T(t)\, \mathbf{R}\, \mathbf{u}(t) \right] dt$$

subject to:

$$\dot{\mathbf{z}}(t) = \mathbf{A}\, \mathbf{z}(t) + \mathbf{B}\, \mathbf{u}(t)$$

where the (normalized) weighting matrices $\mathbf{Q}$ and $\mathbf{R}$ are defined as:

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad \mathbf{R} = \begin{bmatrix} 0.05 \end{bmatrix}$$

This choice was made to give more importance to the position rather than to the velocity, while also trying to maintain a limited control effort with $\mathbf{R}$. This balances the need to achieve stabilization efficiently without requiring excessively aggressive or energy-intensive control inputs. The resulting controller, implemented as in *figure 36*, is:

$$\mathbf{u}(t) = -\mathbf{K}\, \mathbf{z}(t), \qquad \mathbf{K} = \mathbf{R^{-1}B^{T}P}$$

where $\mathbf{P}$ solves the Algebraic Riccati Equation (which is handled offline):

$$-\mathbf{Q} - \mathbf{A^{T}P} - \mathbf{PA} + \mathbf{PBR^{-1}B^{T}P} = \mathbf{0}$$

Remember that, since we are trying to stabilize the ball at x_{ref} with $\dot{x} = u = 0$, the controller is actually defined in a neighborhood of the references, with $\mathbf{\Delta z}$, Δu, etc (it's just a shift of the origin).

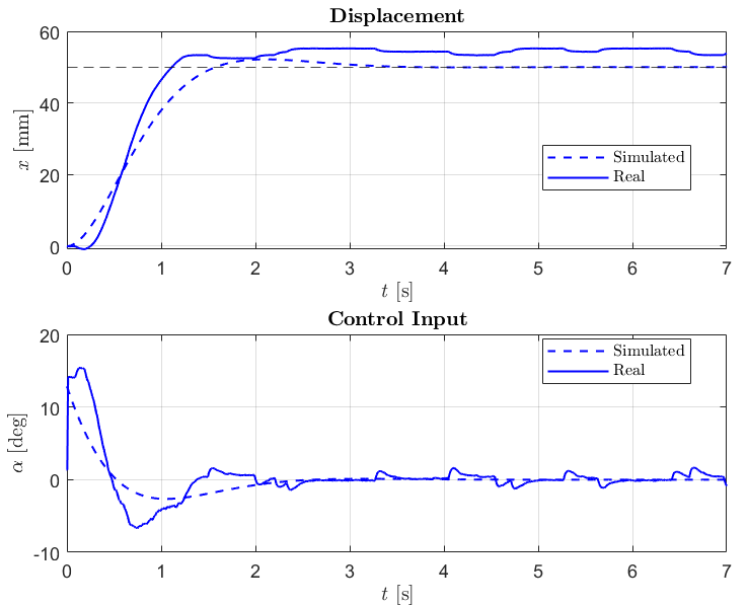

Figure 35: LQR with $Q_x = 1$, $Q_{\dot{x}} = 0$, $R = 0.05$

The experimental result is shown in *figure 35*, and compared with the simulation. It is evident that the control effort required by the LQR controller is significantly lower compared to previous control strategies. The displacement plot shows that the system achieves stabilization with minimal overshoot. However the real system exhibits a steady-state error as for the PD: this issue can be addressed by introducing an integral action.

Integral action (LQRI) The integral action works by accumulating the tracking error of the position over time and feeding it back into the control input. This ensures that the steady-state error converges to zero, also improving robustness to disturbances. However it can introduce slower response or oscillations.
This is implemented by augmenting the state introducing the integral of the tracking error:

$$w = \int (x - x_{ref})\, dt$$

so that:

$$u = -\mathbf{Kz} - k_i \cdot w = -\mathbf{K_{ext}} \cdot \mathbf{z_{ext}}$$

Of course, now matrix $\mathbf{Q}$ has an additional weighting term associated to the integral state:

$$\mathbf{Q_{ext}} = \begin{bmatrix} \mathbf{Q} & \mathbf{0} \\ \mathbf{0} & Q_w \end{bmatrix}$$

Finally, by solving the Algebraic Riccati Equation (ARE) for the augmented matrices, we obtain the optimal control gain $\mathbf{K_{ext}}$. The implementation of this controller is shown in *figure 37*.

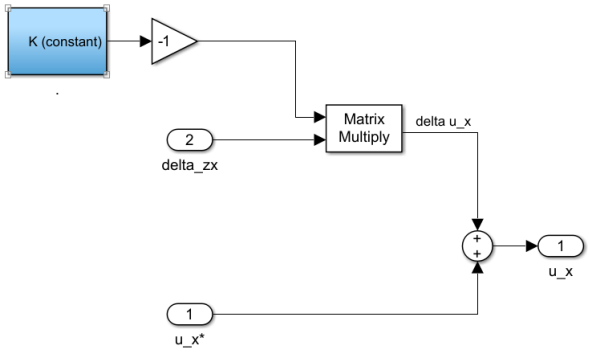

Figure 36: LQR block

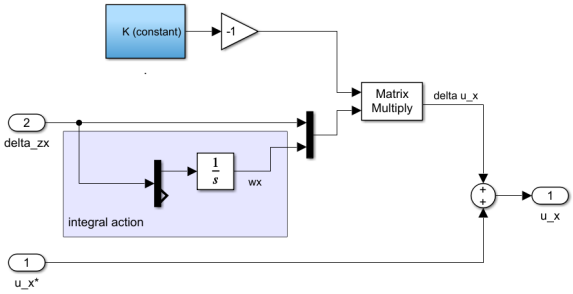

Figure 37: LQRI block

Different test were performed by changing values of the weighting parameters. As shown in both the reported tests, the integral action successfully eliminates the steady-state error, ensuring accurate tracking of the desired position. However, this comes at the cost of increased overshoot and higher oscillations in the control input compared to the standard LQR (see *figure 35*).
In the right test (*figure 39*), the control gain associated, in particular, to the integral action is higher, as expected, due to the increased weight on the integral term $Q_w = 1$, which prioritizes minimizing the cumulative error. While this results in faster convergence to the reference position, the control

input becomes more oscillatory, potentially undermining the system's overall stability and performance. The excessive oscillations observed in the control input may indicate that the overly aggressive weighting on the integral term has had a counterproductive effect, as the system continues to oscillate even after achieving convergence (no anti-windup was used in this case).

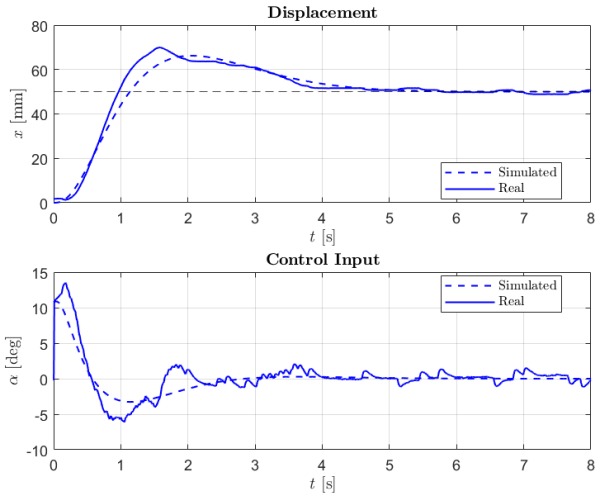

Figure 38: Comparison between the simulated and real system with LQRI controller: $Q_x = 1$, $Q_{\dot{x}} = 0$, $Q_z = 0.1$, $R = 0.11$, $K = [3.64 \;\; 2.70 \;\; 0.91]$

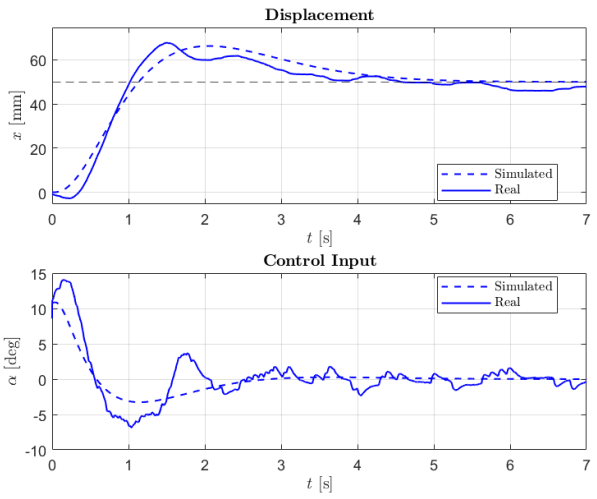

Figure 39: Comparison between the simulated and real system with LQRI controller: $Q_x = 1$, $Q_{\dot{x}} = 0$, $Q_z = 1$, $R = 0.275$,$K = [5.19 \;\; 3.22 \;\; 2.89]$

Conversely, in the left test (*figure 38*), the lower integral weight results in less aggressive corrections, leading to smoother control input and fewer oscillations. Although the error convergence is slightly slower, the system appears more stable after reaching the reference position.

LQR trajectory tracking LQR was also used to track a circular trajectory, following a similar strategy to the one adopted for the PID regulator. Also in this case, the trajectory is provided as a series of discrete reference positions, with a fixed time interval between them. As with the previous scenario, tracking accuracy improved for "slower" trajectories. Furthermore, the control effort was noticed to be lower compared to the PID approach, and there was more room for faster control. In *figure 40*, a circular path with radius $50\ mm$ and period $T = 5\ s$ was chosen along with the following weights: $Q_x = 1$, $Q_{\dot{x}} = 0$, $R = 0.05$.
The accuracy on the radius of the trajectory is slightly worse compared to the one obtained with PID, but this is because we focused on trying to increase the speed of the ball. However, when considering the overall context, the results are actually better, as the system is able to handle higher speeds more effectively without losing control of the ball or experiencing excessive overshooting, a problem we encountered when using faster trajectories with the PID controller. Moreover, as we already mentioned, the control action in *figure* 40 is less demanding than the one observed in *figure 23*.

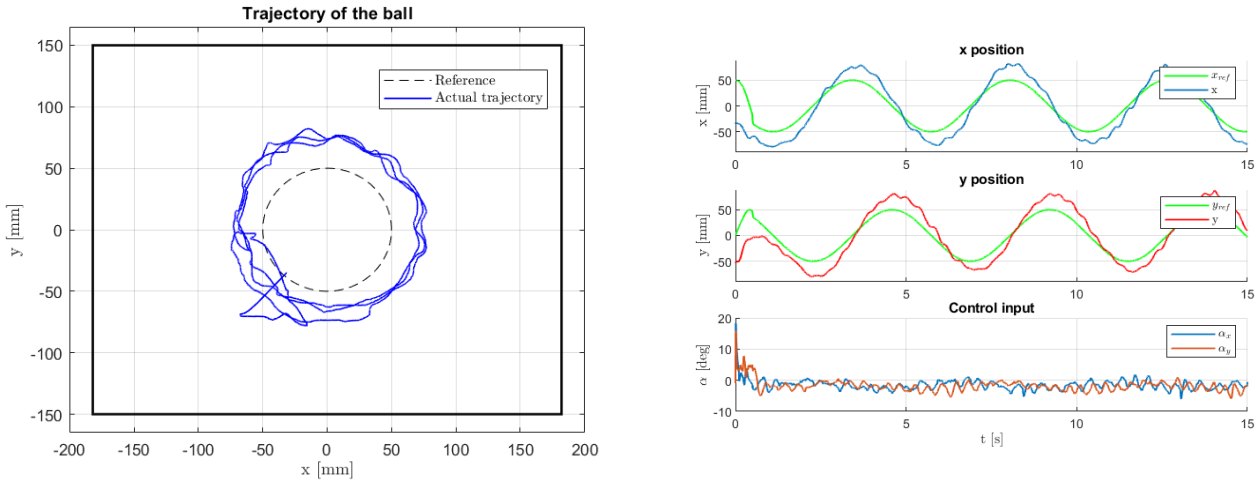

Figure 40: LQR with $Q_x = 1$, $Q_{\dot{x}} = 0$, $R = 0.05$

5 Conclusions, limitations and possible improvements

In this report, we analyzed the design, implementation, and testing of different control strategies for a 2D ball balancing table. Two different setups were used: the original bench, based on an Acrome ball balancing system with a touchpad sensor, and a new bench developed by us, which utilizes a camera-based tracking system and a Texas Instruments Launchpad microcontroller.

Summary results All the tested control strategies were able to successfully stabilize the ball at a desired position, also being consistent with the simulations. In accordance with the theory, an integral action has been used to assure zero steady-state error, which may arise due to disturbances. The addition of the anti-windup architecture was observed not to have a particularly relevant effect on the system, since the control input was rarely saturated. Filters were also added to reduce the effect of measurement noise: Kalman filter showed better results compared to the low pass filter since it is optimal, and also allowed us to obtain an estimate of the full states, key feature needed to implement LQR and pole placement.
Trajectory tracking was also achieved but remained imprecise, particularly with PID, which is not inherently designed for such tasks.
In general, the LQR controls showed slightly better performances, specifically in terms of control effort required for the same task.
Concerning the new bench setup, the experiments showed similar outcomes, but with a slightly better stability in presence of external disturbances and for the transmission of signals. We think this difference is mainly related to the defects found in the PoliArd setup.

Hardware Most of the limitations that we faced during the course of the laboratory, were related to the hardware and software setup of the "old bench".
One of the two servomotor's rod was observed to slip if too much torque was applied on it, which happened when the ball moved close to the edge. This caused the calibration to be compromised during many tests, requiring a manual adjustment of the rod itself. Moreover, the touchpad panel, as already mentioned, was not always able to sense the ball close to the borders. These two issues were faced by trying to maintain the ball as close to the center as possible.
Additionally, some delays were noticed in the signals transmission: we think this could be due to the communication chain from the plant to the microcontroller or due to the PoliArd not always being able to work at the specified frequency.These problems were not present in the new bench, that instead had other limitations.
First of all, the camera mounted over the platform required a new calibration each time a series of tests was performed, since the static mask needed to be adapted to the borders of the table and the HSV filtering parameters had to be regulated basing on the illumination conditions. The first issue could be solved by fixing the camera to the bench, while the second one could probably be improved by painting the table in black and using a white ball, reducing disturbances due to reflections and shadows. On the other hand, the new bench also allowed for controlling the ball while it was bouncing, or when a non-metallic ball was used. Finally, as already mentioned, the rough wooden surface of the table caused the ball to stop in same cases: this could be improved by letting the ball spin or by using an heavier ball.

An important observation has to be done: the "new bench" was developed in the very last part of the laboratory and it was just a prototype. For sure, a bigger number of tests would have allowed to improve, for example, data transmission architecture, further increasing the computational speed (so that higher resolution images could be used, for instance, for a more accurate feedback).

Mathematical model The mathematical model of the system that we defined is not accounting for the centrifugal force that tends to "push" the ball towards the edges when the platform is tilting, and a more complex discussion could be developed. However, this effect is negligible, since the platform angles θ are very small.

Control strategies Stabilization around a reference position was achieved using the presented controllers with sufficiently satisfactory performances, ensuring good disturbance rejection and quick responses. On the other hand, path tracking experiments showed a remarkable margin for improvement. For sure hardware limitations played a certain role in this, but we think the main problem was the way we provided the reference trajectory.
Indeed, an idea could be to implement a finite time LQR control, which optimizes the feedback accounting for the whole time history of the control input and the reference trajectory. We actually developed a successful simulation in Simulink but, due to lack of time, we didn't have the chance to test it on the real system. Nevertheless, since it is a more computationally demanding control strategy, we suppose that it may not work properly on the PoliArd setup, as its board has a limited computational capability.
We could have also implemented parameter estimation using the Kalman filter to assess the disturbances affecting the system. This would enable more accurate compensation or facilitate the development of refined simulations for better tuning the system's gains.
Finally, it is worth to notice that our bench was not including encoders to measure the input angle, which would have allowed to close an additional control loop on the servomotors, helping also with the problem of the slipping rod.

References

[1] Acrome Robotics. Ball Balancing Table. Visited on January 27, 2025. 2024. URL: https://acrome.net/product/ball-balancing-table.

[2] Texas Instruments. LAUNCHXL-F28069M overview. Visited on January 27, 2025. 2019. URL: https://www.ti.com/lit/ug/sprui11b/sprui11b.pdf?ts=1738004492427.

[3] N. Toscani M. Rossi and M. Mauri. "Introduction to Microcontroller Programming for Power Electronics Control Applications: Coding with MATLAB® and Simulink®". Taylor & Francis Ltd, 2021. DOI: 10.1201/9781003196938.