POLITECNICO
MILANO 1863

Automation and Control Laboratory 2021/22

Control Techniques of a Ball and Beam System

KalMen Group
Ploner Giovanni - 970997
Sassaroli Guido - 969547
Tallon Filippo - 963471
Vaccari Giulio - 966357

submitted on: 09/06/2022

Contents

1 Introduction

Aim of this report is to show the results obtained in trying to control a Ball and Beam system provided by the laboratories of Politecnico di Milano. This ball and beam system (*Figure 1*) is composed by an electric motor that drives a shaft with the goal to move, up and down, a hinged beam. On this beam a steel ball is moving linearly and the final goal is to control its trajectory.

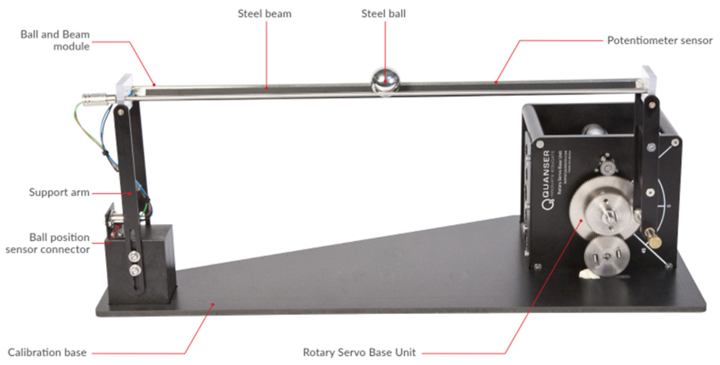


Figure 1: Ball and Beam system.

In this report, we are going to describe all the main phases of the project, starting from the model of the mathematical system (Chapter 2) in both white and gray box approaches and the model of the sensors (Chapter 3). In the same section of sensors it is also possible to find the description of the observers used to better estimate the noisy quantities that we can measure through the encoder and the ball position displacement sensor. Once we have acquired a good knowledge of the system and its components, we focus on control: in Chapter 4 it is possible to find the self-given specifications for the closed loop system that are fundamental to designing all our controllers (Chapter 5). The control techniques used vary from the classical ones (PID regulators, lead compensators) to the modern ones (Pole Placement and LQ control) and even an attempt to apply the Model Predictive Control as advanced method (Chapter 6). Finally, we draw our conclusions by comparing the results obtained (Chapter 7).
For completeness, in the last part of this introductory chapter, are reported all the (nominal) parameters from the available datasheet.
<u>REMARK</u>: notations are the same used in mathematical modelling of the system.

Symbol	Meaning	Value
R_m	Motor's armature resistance	$2.6\ [\Omega]$
L_m	Motor's armature inductance	$0.18\ [mH]$
K_t	Motor's torque/current constant	$7.66 * 10^{-3}\ \left[\frac{N*m}{A}\right]$
K_m	Motor's back-emf constant	$7.66 * 10^{-3}\ [V * s]$
τ_{high}	Motor's high-gear ratio	$70\ [-]$
J_{high}	High-gear equivalent moment of inertia	$2.087x10^{-3}\ \left[kg * m^2\right]$
D_{high}	High-gear equivalent viscous damping	$0.015\ [N * m * s]$
g	Gravitational acceleration constant	$9.8\ \left[\frac{m}{s^2}\right]$
m_{ball}	Ball's mass	$0.64\ [kg]$
J_{ball}	Ball's moment of inertia	$4.193 * 10^{-5}\ \left[kg * m^2\right]$
r_{ball}	Ball's radius	$0.0127\ [m]$
L_{beam}	Beam's lenght	$0.4255\ [m]$
m_{beam}	Beam's mass	$0.65\ [kg]$
J_{beam}	Beam's moment of inertia	$0.0392\ \left[kg * m^2\right]$
r_{arm}	Radius of Motor-Beam arm	$0.0254\ [m]$

Table 1: System's data table.

The moments of inertia were not provided and, since we dealt with a solid sphere and a rigid bar, they have been computed as: $J_{ball} = \frac{2}{5} m_{ball} r_{ball}^2$ and $J_{beam} = \frac{m_{beam} L_{beam}^2}{3}$.

2 System's Mathematical Model and Parameters Identification

As explained in the introduction, we are dealing with a system composed by two main dynamics: the motor and the mechanical connection.

2.1 Motor's Electrical Model

To have a good understanding of our system we started modeling the only dynamics of the motor by unplugging the shaft from the motor.
We followed both a white box approach, in which we derived mathematical equations for all the components of the system using parameters available in the datasheet, and a gray box one, in which we update all the uncertain parameters based on data retrieved from the real system.

2.1.1 White box

The motor used in our experiments is a DC one. From our control-oriented prospective, the DC motor is a SISO system that received as input the a voltage (V_{DC}) and as output the angular position of the motor (θ). The goal is to find the mathematical relationship between these two quantities: the starting point are the following constitutive equations:

$$\begin{cases} T_m = K_t I_{DC}; \\ E = K_m \Omega; \\ L_m \dot{I}_{DC} + R_m I_{DC} + E = V_{DC}; \end{cases}$$

Nevertheless, it is possible to simplify L_m due to its very low value and for its faster dynamics compared to the one of the complete system. The gear ratio of the transmission subsystem is taken into account and set to the high-gear of the datasheet. For sake of clarity, we define now the angular speed of the DC motor as $\dot{\theta}$ and we set $K_t = K_m = K$. Hence, adding the classical torque-load equation dynamics:

$$\begin{cases} T_m = \tau_{high} K I_{DC}; \\ E = K\dot{\theta}; \\ R_m I_{DC} + E = V_{DC}; \\ T_m = J_{high}\ddot{\theta} + D_{high}\dot{\theta}; \end{cases}$$

Then, by applying Laplace transform and with some simple computations, is it possible to write the transfer function between the supply voltage V_{DC} and the motor's angular position θ:

$$G_{V_{DC}\theta}(s) = \frac{\tau_{high}K}{R_m J_{high} s^2 + R_m D_{high} s + \tau_{high} K^2}$$

However, due to some uncertainties in parameters, the results obtained using a purely white box approach are not satisfactory (as can be seen in the next section). For this reason, we moved to a gray box approach, in which we keep the same structure of the transfer function, in terms of number of poles and zeros, but we identified the correct values of their frequencies through some experiments on the real system.

2.1.2 Gray box

In this section is explained how we derived the transfer function from V_{DC} to θ using a gray box approach.
We started by collecting some experiments on the motor (in open loop), in particular by applying a chirp voltage to the motor and measuring the value of θ from the encoder. Then, we used the tfest function of Matlab to estimate a continuous-time transfer function system. It receives as input a time-domain data and the number of poles of the transfer function that we want to estimate, giving as output the computed transfer function. As explained before, the number of zeros and poles is selected using the knowledge of the system that we got from the design of the white box model. In this case, 2 poles and no zeros.
As it is possible to see from the following plots, we were able to estimate with a good degree of accuracy our transfer function.
The estimated transfer function takes this form:

$$G_{V_{DC}\theta}(s) = \frac{50}{s^2 + 34.68s}$$

The overall dynamics of the system, given by its poles, is very well defined and modeled while the gain of the system changes slightly based on the type of response that we ask to the system. We found values ranging from 50 to 60, but we decided to keep 50 as it works generally better on continuous inputs. It is noteworthy that with values closer to 60 the model predicts better the steps' behaviour, this is also the reason why on certain models (notably the MPC) we used the value of 59. The different gains are probably caused by variations in the friction, or even the values of K_τ, the torque motor constant (that depends on the motor torque).
This function reaches up to 99% test accuracy and consistently more than 90% on test data for the reasons previously discussed.

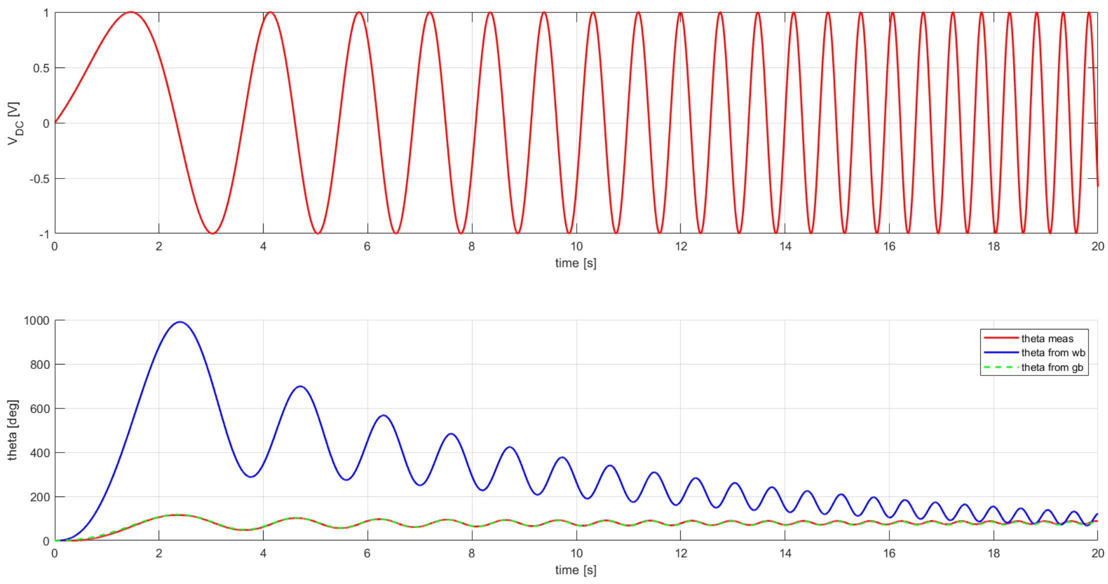


Figure 2: Training.

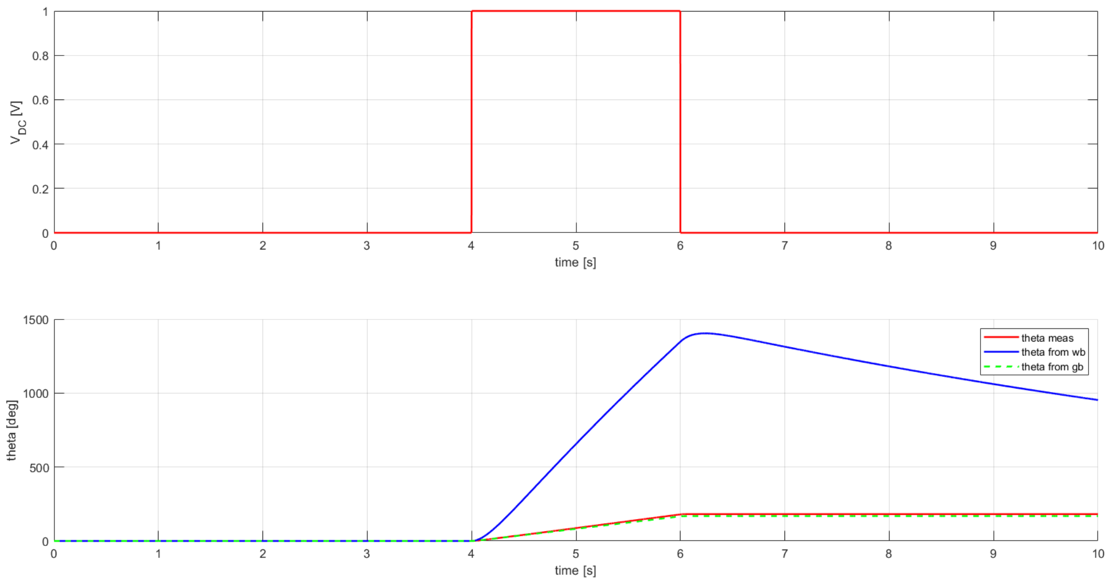


Figure 3: Testing.

In *Figure 2* we present our training process. In color red are the real data, green the estimation and blue the white box with parameters taken from the data table. We used a chirp to explore enough frequencies to build a model robust enough for the different possible scenarios. *Figure 3* represents a validation on the real system given two steps in the voltage, one positive quickly followed by one negative that bring the value back to 0, stopping the movement of the motor. Our model predicts perfectly the motor behaviour.

2.1.3 State Space Model of the Motor

Once obtained the final transfer function of the motor model we can directly compute the state space form of the system. This is useful for the implementation of state space observers that will be explained in the next sections.
The state space model can be written in this way:

$$\dot{\mathbf{x}} = A_m\mathbf{x} + B_m u$$
$$y = C_m\mathbf{x} + D_m u$$

Where:

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix}\theta & \dot{\theta}\end{bmatrix}^T; \\ u &= V_{DC}; \\ A_m &= \begin{bmatrix}0 & 1 \\ 0 & -34.64\end{bmatrix}; \\ B_m &= \begin{bmatrix}0 \\ 50\end{bmatrix}; \\ C_m &= \begin{bmatrix}1 & 0\end{bmatrix}; \\ D_m &= 0; \end{aligned}$$

The motor system's corresponding Bode plot is shown in *Figure 4*

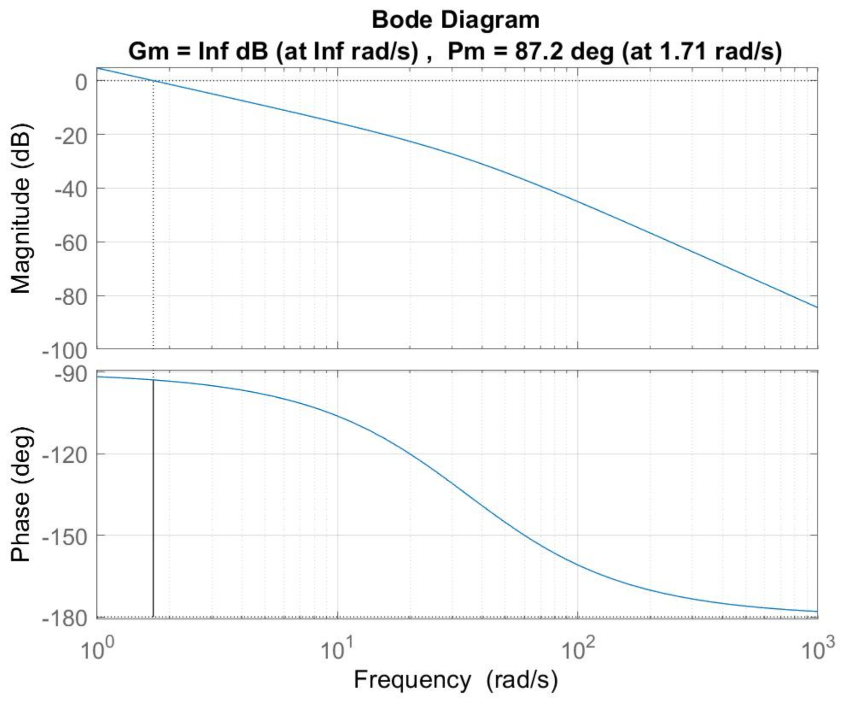


Figure 4: Bode plot of $G_{V_{DC}\theta}(s)$

Being useful for some control techniques used in this report, the following lines of code shows that the system in question is fully reachable and observable, owing to they ranks are equal to the system's order:

```
>> rank(ctrb(A_m,B_m))

```

```
ans =
      2

>> rank(obsv(A_m,C_m))

ans =
      2
```

Listing 1: Motor Controllability and Observability test (Matlab)

2.2 Ball and Beam Mechanical Model

We then proceed analyzing the model of the complete system, again using both white and gray box approaches. In this case the relation we want to discover is from θ to x (displacement of the ball).
In *Figure 5*, it is possible to see the representation of the system with its variables and parameters used to developed the mathematical model. Notice in particular the direction of the x axis, pointing towards the hinged point of the beam.

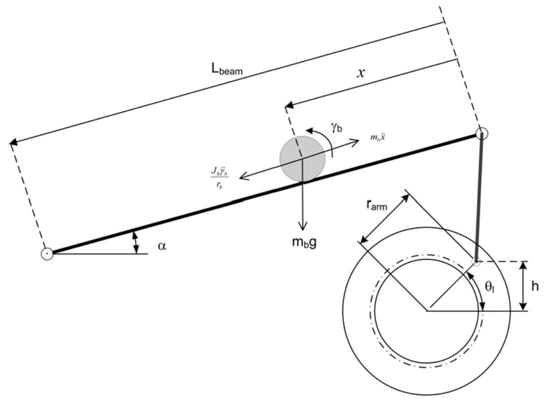


Figure 5: Ball and Beam system schematics.

2.2.1 Non linear Model

In order to study the Ball and Beam mechanical model, we make use of the Lagrange equations method based on the energy balance of the system. Recalling the Lagrange equations:

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

Where T is the kinetic energy of the system,D the dissipation function (taking into account all the dissipative forces of the system), V is the potential energy of the system, Q is the Lagrangian component of all the remaining forces and $\mathbf{q}$ is the vector containing the system's degrees of freedom, which are, in our case, x, i.e. the position of the ball with respect the beam end attached to the motor, and α, i.e. the angular position of the beam with respect to the horizontal plane.

$$\begin{aligned}
&T = \frac{1}{2}J_{beam}\dot{\alpha}^2 + \frac{1}{2}m_{ball}\dot{x}^2 + \frac{1}{2}J_{ball}\left(\frac{\dot{x}}{r_{ball}}\right)^2 + \frac{1}{2}\left(J_{beam} + J_{ball} + m_{ball}\left(L - x\right)\right)\dot{\alpha}^2\\
&D = 0\\
&V = m_{ball}g\left(L_{beam} - x\right)\sin\alpha + m_{beam}g\frac{L_{beam}}{2}\sin\alpha\\
&Q = \left[0, T_m\right]^T ; \mathbf{q} = \left[x, \alpha\right]^T
\end{aligned}$$

Performing all the computation, we obtain the dynamics equations of our non-linear system:

$$\begin{cases}
\left(m_{ball} + \frac{J_{ball}}{r_{ball}^2}\right)\ddot{x} + 2m_{ball}\left(L_{beam} - x\right)\dot{\alpha} - m_{ball}g\sin\alpha = 0;\\
\left(J_{beam} + J_{ball} + m_{ball}\left(L_{beam} - x\right)^2\right)\ddot{\alpha} - 2m_{ball}\left(L_{beam} - x\right)\dot{x}\dot{\alpha} +\\
+\left(m_{ball}\left(L_{beam} - x\right) + m_{beam}\frac{L_{beam}}{2}\right)g\cos\alpha = T_m;
\end{cases}$$

2.2.2 Linearized model

In order to apply classical control techniques, we need to deal with a linear system and to do so we apply a linearization process to the previously described one. Choosing as state variables of the original system $\mathbf{x} = [x, \alpha, \dot{x}, \dot{\alpha}]^T$, we can apply a linearization around the equilibrium point $\mathbf{x}_{eq} = [\bar{x}, 0, 0, 0]^T$ and $u_{eq} = T_{m,eq} = \left(m_{ball}\left(L_{beam} - \bar{x}\right) + m_{beam}\frac{L_{beam}}{2}\right) g$ which leads us to the following system:

$$\dot{\delta\mathbf{x}} = A_{LIN}\delta\mathbf{x} + B_{LIN}\delta u$$
$$y = C_{LIN}\delta\mathbf{x} + D_{LIN}\delta u$$

Where:

$$\begin{aligned}
&\delta\mathbf{x} = \mathbf{x}_{eq} - \mathbf{x};\\
&\delta u = T_{m,eq} - T_m;\\
&A_{LIN} = \begin{bmatrix} 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{m_{ball}g}{m_{ball}+\frac{J_{ball}}{r_{ball}^2}} & 0 & 0\\ \frac{m_{ball}g}{J_{beam}+J_{ball}+m_{ball}(L_{beam}-x)^2} & 0 & 0 & 0 \end{bmatrix};\\
&B_{LIN} = \begin{bmatrix} 0\\ 0\\ 0\\ \frac{1}{J_{beam}+J_{ball}+m_{ball}(L_{beam}-x)^2} \end{bmatrix};\\
&C_{LIN} = \begin{bmatrix} 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1 \end{bmatrix};\\
&D_{LIN} = 0;
\end{aligned}$$

Since our aim is to control x through θ is essential to express the first with respect the latter in our equations. This can be done by exploiting the correlation between α and θ, i.e. $\alpha = \arcsin\left(\frac{r_{arm}}{L_{beam}}\theta\right)$, which, linearizing around the equilibrium, becomes:

$$\alpha = \frac{r_{arm}}{L_{beam}}\theta$$

Thus, taking into account the second entry in the third row of A_{LIN} matrix we obtain:

$$\ddot{x}(t) = \frac{m_{ball}g}{m_{ball} + \frac{J_{ball}}{r_{ball}^2}}\frac{r_{arm}}{L_{beam}}\theta(t)$$

Or, in the frequency dominion:

$$s^2X(s) = \frac{m_{ball}g}{m_{ball} + \frac{J_{ball}}{r_{ball}^2}}\frac{r_{arm}}{L_{beam}}\Theta(s)$$

Which leads us directly to the transfer function between the motor angular position θ and the ball position x:

$$G_{\theta x} = \frac{r_{arm}m_{ball}g}{L_{beam}\left(m_{ball} + \frac{J_{ball}}{r_{ball}^2}\right)}\frac{1}{s^2}$$

Notice that $G_{\theta x}$ has the following structure:

$$G_{\theta x} = \frac{\mu}{s^s}$$

So the next step will be to correctly identify the parameter μ, that, linearizing around $\theta = 0$, depends from mechanical parameters of the ball and beam system which are unknown or uncertain. However, there is no a correct solution to this problem, μ is related on the value of θ we are using to perform the linearization.
Like discuss in previous subsection, we use data to find the right parameter of this transfer function.

2.2.3 Gray box

Having a good understanding of the mechanical model of the system and knowing its number of poles and zeros we were able to estimate the parameter μ to use in our experiments, given the simplified linearized model:

$$G_{\theta x} = \frac{\mu}{s^2}$$

Having gathered enough data it is only a matter of performing a linear regression (linear unconstrained least square) to find the unknown parameter. This can also be done online as we will see in the Adaptive MPC.
The following plots (*Figure 6*) show the result we obtain with this identification: the first graph represents the input: a step on the reference of θ_{ref}; note that in this identification the internal control loop has been already closed and neglected for its faster dynamics. In the second plot we can see the good fitting between the measured x (in red) and the predicted one (dotted blue).
With respect to the white box this approach has less sensitivity to the errors of the single parameters, and can take into account the approximations derived by the linearization, allowing us to have an higher accuracy but keeping the advantages of the linearity.
As seen in *Figure 6* the error is very close to 0% also in test data, keeping a good degree of generality.
To perform the computation of μ we used the MATLAB function $fminsearch$.
Since a good sample size was available, we were able to determine the value of μ to be around 0.26 during steps, with a variance of around 0.003.
We decided to focus on steps and not to test a wider range of frequencies because sweep sines are difficult to do on the Ball and Beam system and produced uncertain results, mainly due to the slow dynamic of the system. Furthermore, most of our controllers will rely on steps or on step related control actions.
At this point, we have identified two different transfer functions: $G_{V_{DC}\theta}$ for the electric motor and $G_{\theta x}$ for the ball and beam system and we are ready to implement our control algorithms.
However, before closing the chapter on mathematical model of this system, we need also to write the state space representation, useful to build our observers.

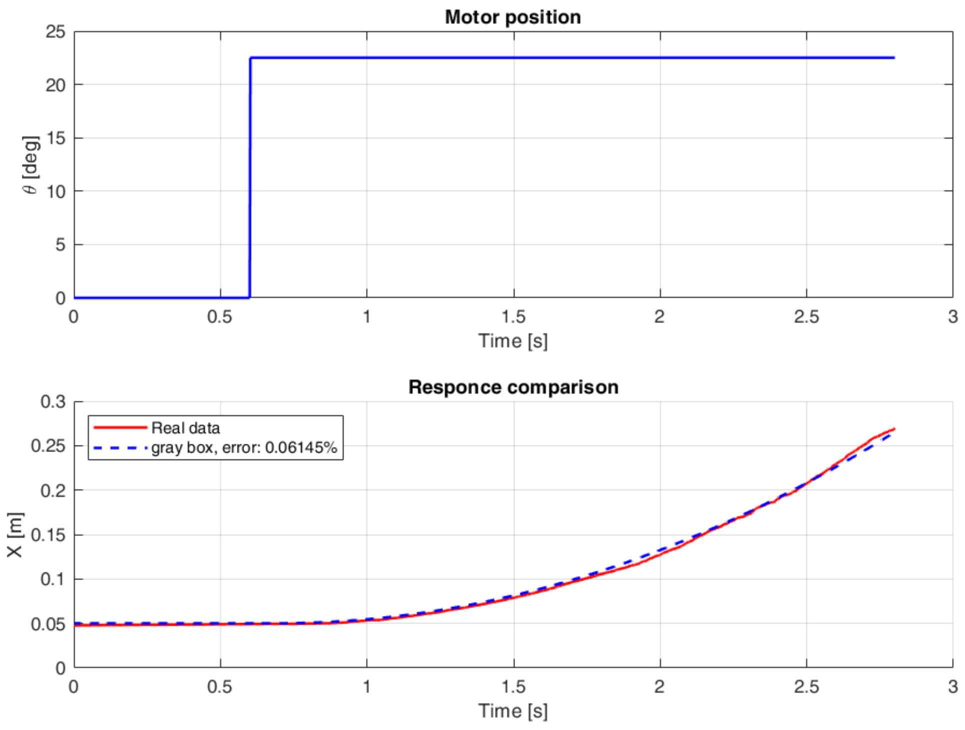


Figure 6: An example of the real and the predicted step response using the gray box model on test data.

2.2.4 State Space System

Once obtained the final transfer function of the ball and beam model, we can directly compute the state space form of the system:

$$\dot{\mathbf{x}}_{bb} = A_{bb}\mathbf{x}_{bb} + B_{bb}u_{bb}$$
$$y = C_{bb}\mathbf{x}_{bb} + D_{bb}u_{bb}$$

Where:

$$\begin{aligned}
\mathbf{x}_{bb} &= \begin{bmatrix} x & \dot{x} \end{bmatrix}^T; \\
u_{bb} &= \theta; \\
A_{bb} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}; \\
B_{bb} &= \begin{bmatrix} 0 \\ \mu \end{bmatrix}; \\
C_{bb} &= \begin{bmatrix} 1 & 0 \end{bmatrix}; \\
D_{bb} &= 0;
\end{aligned}$$

The system described by these equations fulfils the conditions guaranteeing its full observability and reachability:

```
>> rank(ctrb(A_bb,B_bb))

ans =
     2

>> rank(obsv(A_bb,C_bb))

ans =
     2
```

Listing 2: Ball and Beam Controllability and Observability test (Matlab)

2.3 Final Complete Model

In conclusion, we ended up with two different models, one for the motor, the other one for the ball and beam system, both in transfer function and state space. Note that the two models are strongly related: the output of the motor (θ) is the input of the linearized mechanical system.
Now, before treating control techniques and regulation matters, we still need to understand the behaviour of the sensors.

3 Sensors and Observers

3.1 Angular position sensor

The electrical motor is provided of an optical encoder that measures the angular position of the load shaft. The encoder used is a US Digital E2 single-ended optical shaft encoder that offers a high resolution of 4096 counts per revolution in quadrature mode (1024 lines per revolution). The complete specification sheet of the E2 optical shaft encoder is given in [1].
The incremental encoders measure the relative angle of the shaft, this is fundamental in order to have a good controller. The position signal generated by the encoder can be directly connected to the data-acquisition device using a standard 5-pin DIN cable. We applied this transformation to the signal in order to transform it from counts per revolution to radians.

$$\theta_{rad} = \frac{360}{4096} \cdot \frac{\pi}{180} = \frac{2\pi}{4096}$$

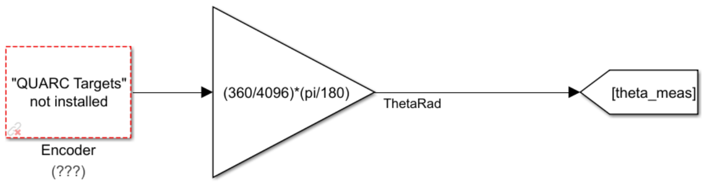


Figure 7: Simulink scheme of the angular position sensor.

The measurement of the angular position is relative, this fact must be taken into consideration during the kick off of the system. However, according to our reference system, $\theta = 0$ means beam horizontal and parallel to the plane (as in *Figure 7*).

3.2 Ball position sensor

The tracking of the linear transducer module, on which the metal ball is free to roll, consists of a steel rod in parallel with a nickel-chromium wire-wound resistor forming the track. The resistive wire is the black strip that is stuck on the plastic which is fastened onto the metal frame. The position of the ball is obtained by measuring the voltage at the steel rod. When the ball rolls along the track, it acts as a wiper similar to a potentiometer resulting in the position of the ball. In

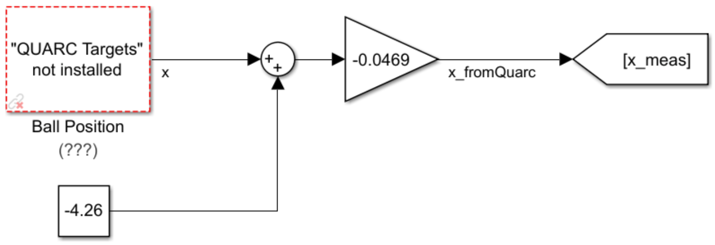


Figure 8: Simulink scheme of the ball position sensor.

the Ball And Beam User Manual it is written that the ball position sensor range is from -5 V to +5 V. We have made different tests and we noticed that the real range is from -4.26 V to +4.26 V. We shifted the origin of the reference system from the center to the end of the beam, on the extreme just above the DC motor. Also the ball position sensor sensitivity is different between the datasheet and the real system: in the first one it results to be -4.25 cm/V but we have seen that -4.69 cm/V is more accurate. Moreover, the gain of -0.0469 is used in order to transform the voltage measurement into meters.
During the sensor tests we noticed several anomalies. The first one is a really strange behaviour when the ball hit the end-stops of the rail. For this reason, we put two felt pads at the ends of the rod.

[1] [1] https://www.usdigital.com/products/encoders/incremental/kit/e2/

Other issues affect the x sensors: it has a non-linear behaviour near edges and the measurement becomes loudly noisy in the "lower" half of the beam (i.e. for x values between 0 and 0.2 meters), as can be seen in *Figure 9.*
In general, the measurement of the position of x is very noisy and it was impossible to have a good control using directly the measurement of the sensor.
Furthermore, to get the velocity we should derive the position, and with such a noisy position the speed is even more inaccurate. For this reason we implemented an observer, in order to have a better reference of our states, θ, $\dot{\theta}$ and x, $\dot{x}$.

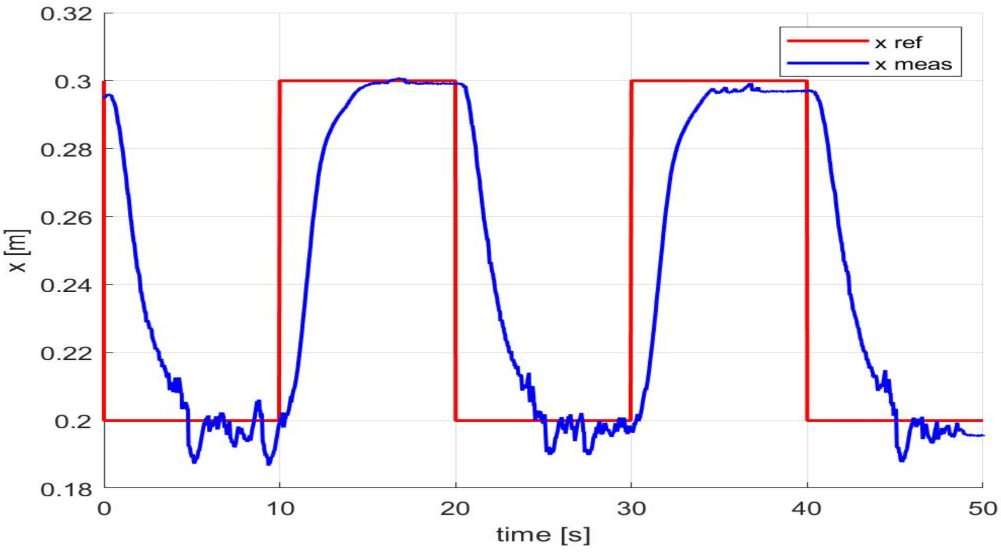


Figure 9: Example of bad sensor acquisition in the "lower" side of the beam.

3.3 Theta observer

In order to have a more accurate measurement of angular position and of the velocity of the motor we decided to estimate it through an observer, in particular we implemented a classical Kalman Filter in continuous time.
Without entering in theoretical details, Kalman Filter is designed using the LTI state space system of the motor already showed in the section 2.1.3 and, through matrices Q, R, N, it computes static gains L that allows to correct the values of the states using the available measurements:

```
Algorithm 1: Kalman Filter ($\theta$)
  Data: $V_{DC}(t)$,$\theta(t)$
  Result: $\hat{\theta}(t)$,$\dot{\hat{\theta}}(t)$
  at each iteration, compute:
    $\hat{\theta}(t) = A_{m_{11}}\hat{\theta}(t) + A_{m_{12}}\dot{\hat{\theta}}(t) + B_{m_1}V_{DC} + L_1[\hat{\theta}(t) - \theta(t)]$
    $\dot{\hat{\theta}}(t) = A_{m_{21}}\hat{\theta}(t) + A_{m_{22}}\dot{\hat{\theta}}(t) + B_{m_2}V_{DC} + L_2[\hat{\theta}(t) - \theta(t)]$
    where all parameters are pre-computed offline.
```

In our setup, the process noise covariance matrix, Q, is set equal to $\begin{bmatrix} 10000 & 0 \\ 0 & 10000 \end{bmatrix}$, so both θ and $\dot{\theta}$ are weighed in the same way; instead the measurement noise covariance matrix, R, is set equal to 1 and the process and measurement noise cross-covariance matrix, N, is set equal to zero. All these values have been determined by tuning the observer in the real system. In particular, we can state that with these parameters we are trusting the encoder due to its good resolution and almost not noisy measurement.
For the implementation, we decided to use the Simulink block "Kalman Filter"[2] that takes as

[2][1] https://it.mathworks.com/help/control/ref/kalmanfilter.html

inputs the state space model described in section 2.1.3 and matrices Q, R, N and computes the estimation of the states ($\hat{\theta}$ and $\hat{\dot{\theta}}$).

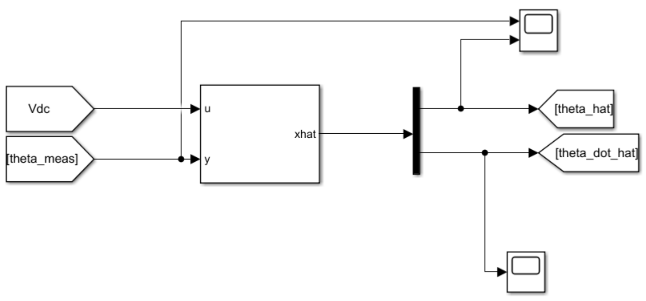


Figure 10: Simulink scheme of motor Kalman Filter.

In *Figure 11, 12,13 and 14* it is possible to see the results obtained using this algorithm. In these figures, it is represented the behaviour of the closed loop system when trying to track a step on the reference: in *Figure 11* we can see in the red line the measured θ, output of the encoder and in dotted blue the estimation, $\hat{\theta}$ of our filter. In this case the function of our observer is to smooth the stepped one of the encoder. Instead in *Figure 14*, is plotted the estimation of the angular velocity of the motor, $\hat{\dot{\theta}}$.

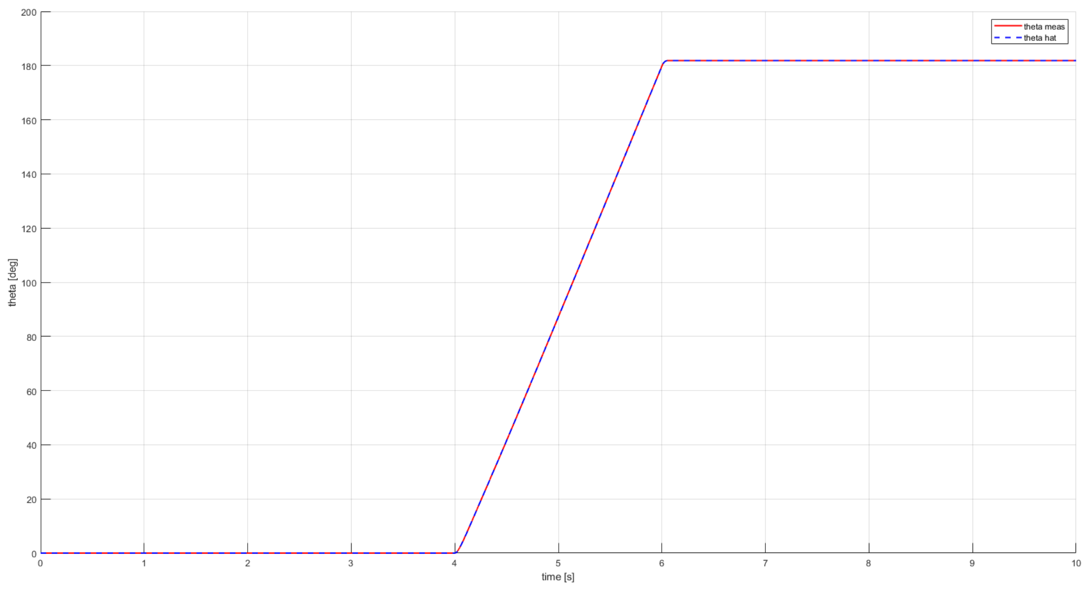
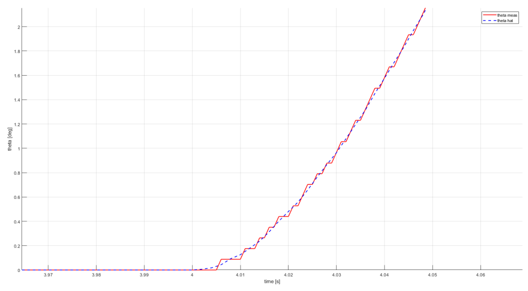


Figure 11: θ estimation using Kalman Filter.



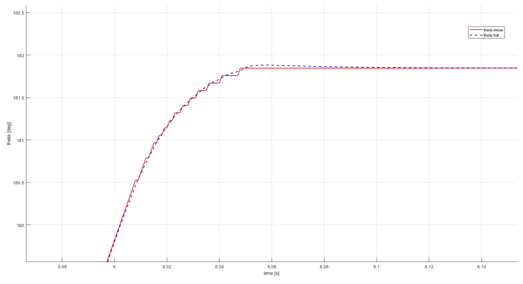


Figure 12: Figure 11, detail.

Figure 13: Figure 11, detail.

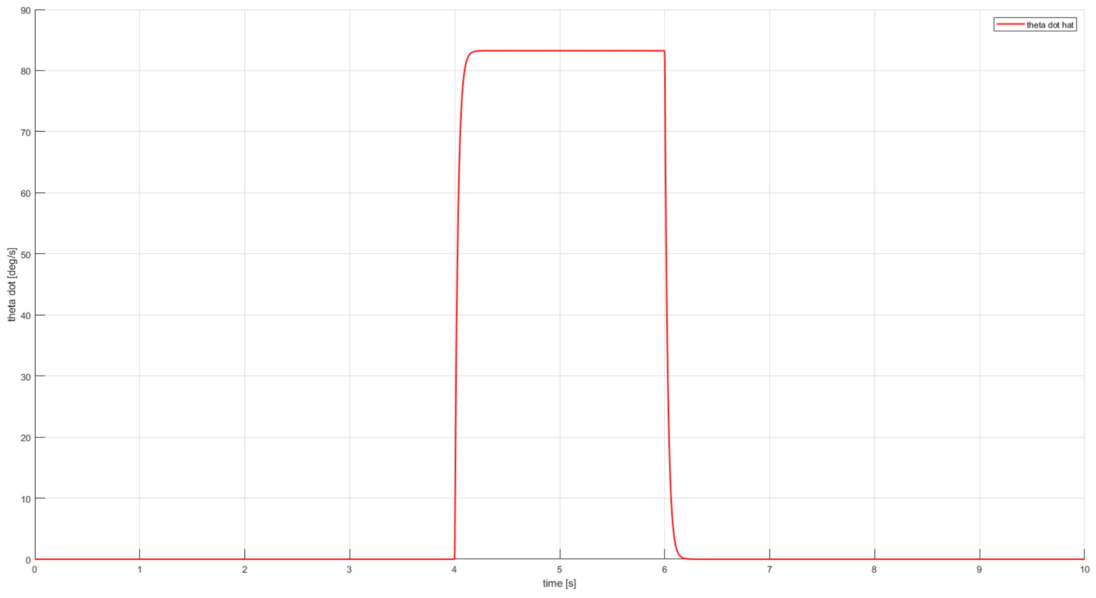


Figure 14: $\dot{\theta}$ estimation using Kalman Filter.

3.4 Position observer

In order to have a more accurate measurement of the position of the ball, we decided to estimate it through an observer, and also in this case we implemented a Kalman Filter. The Kalman Filter is designed using the LTI state space system of the ball and beam already discussed in section 2.2.4.

```
Algorithm 2: Kalman Filter ($x$)
  Data: $V_{DC}(t)$,$\theta(t)$
  Result: $\hat{x}(t)$,$\hat{\dot{x}}(t)$
  at each iteration, compute:
    $\hat{x}(t) = A_{bb_{11}}\hat{x}(t) + A_{bb_{12}}\ \hat{\dot{x}}(t) + B_{bb_1}\theta + L_1[\hat{x}(t) - x(t)]$
    $\hat{\dot{x}}(t) = A_{bb_{21}}\hat{x}(t) + A_{bb_{22}}\hat{\dot{x}}(t) + B_{bb_2}\theta + L_2[\hat{x}(t) - x(t)]$
    where all parameters are pre-computed offline.
```

The process noise covariance matrix, Q, is set equal to $\begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$, R is set equal to 1 and N = 0. Also in this case the parameters were found by tuning this observer on the real system. As with θ we keep a high Q/R ratio to have a faster observer trusting more observation with respect to rely on our mathematical (linearized) system.

REMARK: this tuning is made on the worst case scenario, so when the operating region is the lower part of the beam.

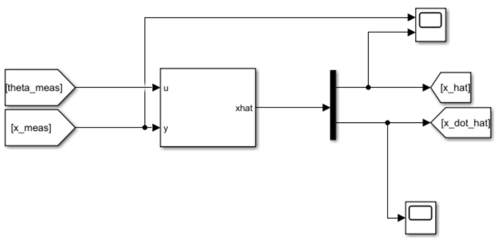


Figure 15: Simulink scheme of the ball position Kalman Filter.

In *Figure 16, 17,18 and 19* it is possible to see the results obtained using this algorithm. In the first figure (16) it is possible to see in red the ball position sensor output x and in dotted blue its estimation $\hat{x}$, output of the Kalman filter. As it is easy to see from this figure, the mismatch between our filter and the measure increases in the region around $0.2m$ but our filter is capable of smoothing also big measure oscillations. Finally in *Figure 19*, is visualized $\hat{\dot{x}}$ that as expected, is zero when the ball is at steady state and more oscillatory when the ball is in the lower part of the beam.

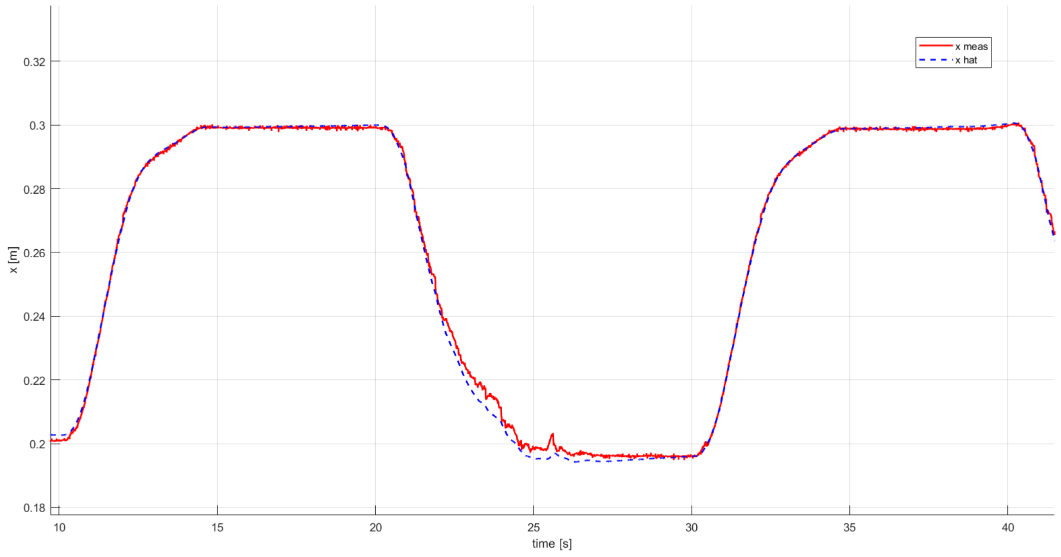


Figure 16: x estimation using Kalman Filter.

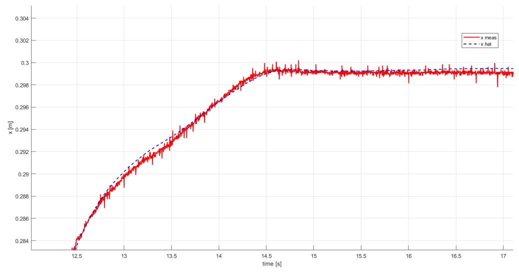


Figure 17: Figure 16, detail.

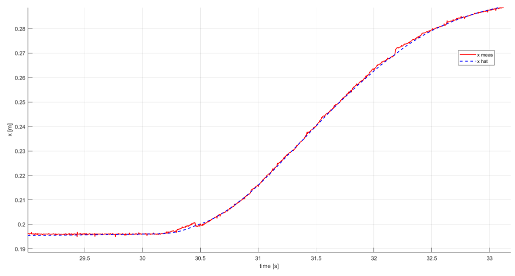


Figure 18: Figure 16, detail.

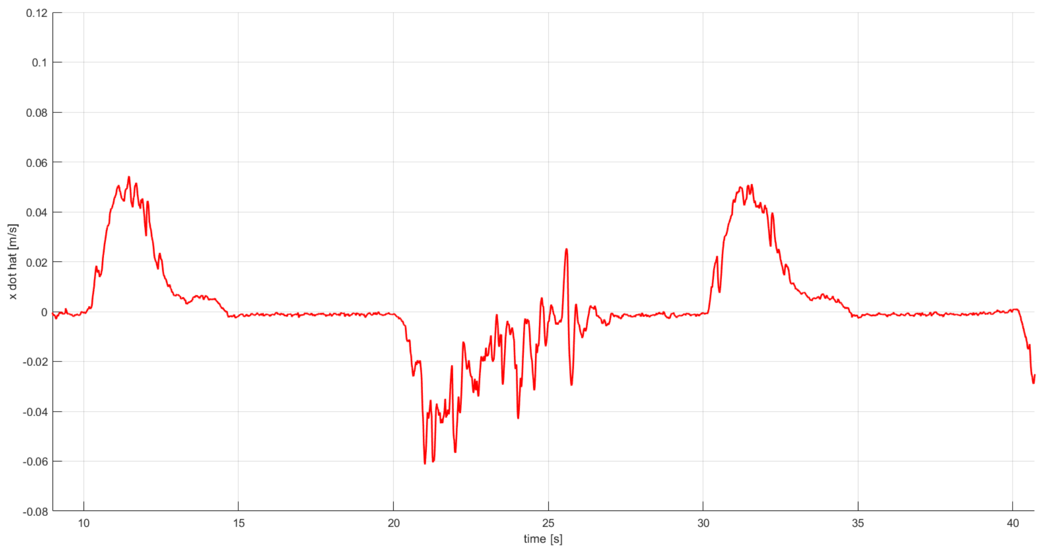


Figure 19: $\dot{x}$ estimation using Kalman Filter.

4 Self-given Specifications

After modelling motor dynamics, we started to design our controllers.
As it is clear from the modelling chapter, our system it is composed mainly by two block: electric motor and ball and beam mechanics. The idea is to have two nested control loops, the internal one, related to motor dynamics, that receives in input the reference on θ (θ_{ref}) and provides as output the actual θ.
The external loop, instead, tries to follow a reference on ball position (x_{ref}), and it is designed based on the mechanical system (and the transfer function of the internal loop, $T_{\theta ref 2\theta}$).

4.1 Motor's Angular Position Required Specifications

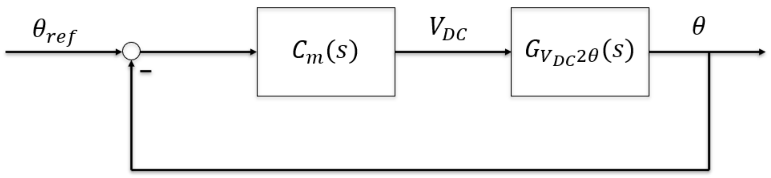


Figure 20: High level view of the θ control scheme.

For the motor's angular position control loop, we have fixed the following specifications:

- settling time lower or equal to 0.5 seconds;
- zero steady state error;
- no overshoots neither oscillations;

These specifications are really important and strict: if our controller is able to satisfy these constraints, we can approximate the theta control loop as a unitary block. In this way it is possible to design the ball position control law without taking into account the delay and dynamics of this loop.

4.2 Ball Position Required Specifications

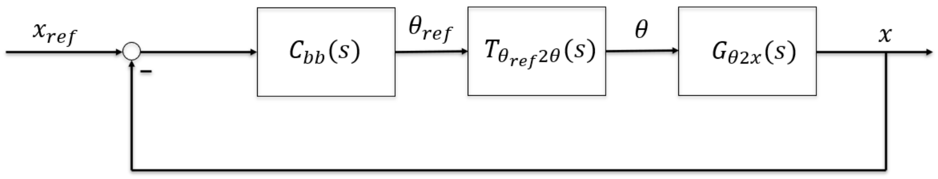


Figure 21: High level view of the x control scheme.

For the motor's angular position control loop, we have fixed the following specifications:

- settling time lower or equal to 10 seconds (as fast a possible, compatible with slow ball and beam dynamics);
- zero steady state error;
- (possibly) no overshoots neither oscillations;
- the motor's reference position should stay between $-45°$ and $+45°$;

These specifications are less stringent with respect to the theta control loop. In particular, it is not easy to assign a target bandwidth because an excessively fast position control will try to follow the measurement that, as explained before, is not very ”clean”.
In conclusion, we can state that a control that can track constant references without moving too much the beam and with a settling time lower than 10 second can be considered a good result for our problem.

5 Classical and Modern Control Techniques

This chapter can be considered the "core" of our project, in which we will focus on the design of our controllers.
At this point we have modelled all the main parts of our system and understood the behaviour of our sensors and actuators, the only thing missing is the implementation of the algorithm(s) that is capable of stabilizing the ball.
For the design of the controllers, we tried to implement the classical and modern techniques we have learned in our "control" courses and in particular we are going to present:

- PID;
- Pole Placement;
- Linear Quadratic (LQ) control + integrator;

5.1 Motor's Angular Position Control

Like in modelling section, the first step is to close the internal control loop and design the "θ regulator" based on $G_{V_{DC}2\theta}$.

5.1.1 PID Controller

In order to control the motor system we firstly use a classical lead-compensator tuned looking on the transfer function of the model: at first we design a regulator with a zero that could cancel the gray box system's pole, a fast enough pole for feasibility and a tunable gain to obtain the desired closed loop specifications. However, as described in Observers' chapter, the plant has not a real integrator (due to friction), so it is necessary to add an integrator in the controller in order to reach a zero steady state error. This problem is caused by the friction of the gears of the motor.
This causes an integrator windup in the PID feedback control. This happens because we use an integral action with some saturation in the loop, so there is an error accumulation in the integrator that bring to a worse step response. For this reason we also implemented the controller using a anti wind-up scheme.
In *Figure 22* we can see the PID controller in the anti-windup configuration. We reached a good response.
To improve the performance and erase the overshoot effect, it is added a prefilter that acts on the reference signal for θ with a time constant of $T_{pf} = 0.1sec$:

$$P(s) = \frac{1}{1 + sT_{pf}}$$

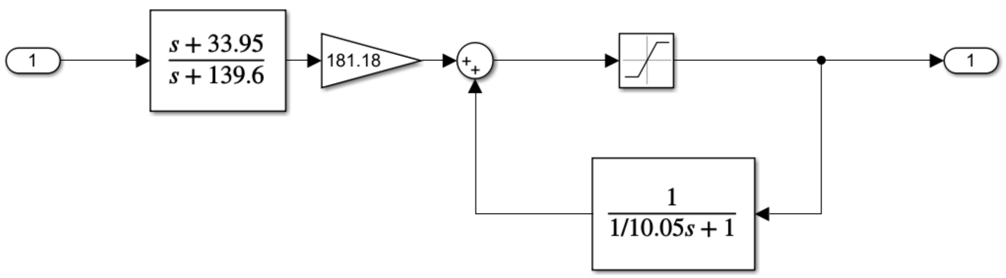


Figure 22: Simulink scheme of the PID controller for the theta's loop.

By doing so we improve the overall tracking of the set point signal and obtain the following responses.
We can see in *Figure 23,24,25,26,27,28* the different responses. In *Figure 24* we can see that the

response time of our controller is less than 500ms, there is no steady state error and there are neither overshoots or oscillations.
Figure 26 represents the response to a chirp signal. The controller follows the reference really well at low frequencies, but the higher the frequency the worse is the reference tracking.

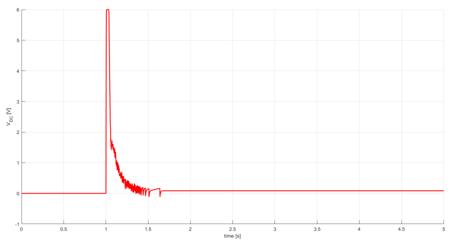


Figure 23: PID Step response: motor voltage

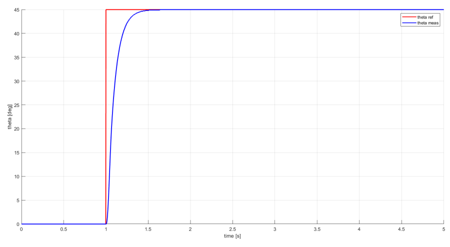


Figure 24: PID step response: angular position

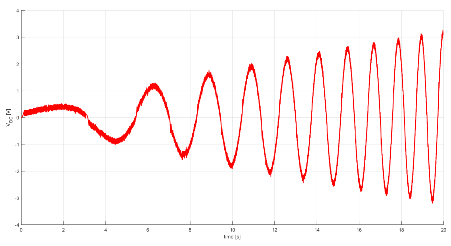


Figure 25: PID chirp (0.01-1 Hz) response: motor voltage

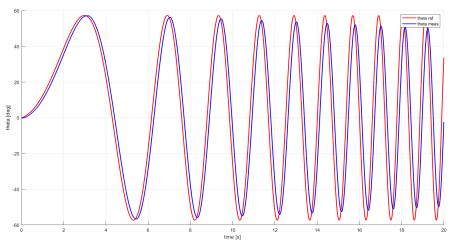


Figure 26: PID chirp (0.01-1 Hz) response: angular position

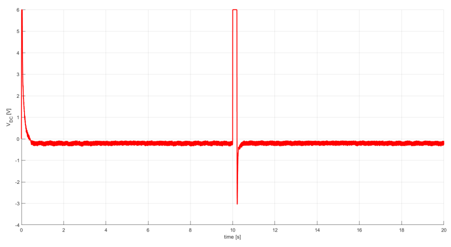


Figure 27: PID sawtooth response: motor voltage

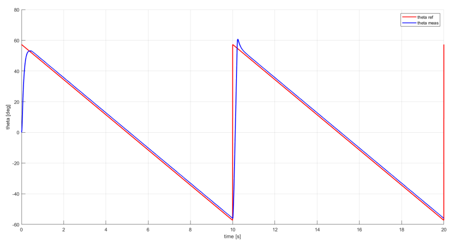


Figure 28: PID sawtooth response: angular position

5.1.2 Pole Placement

Moving on modern control techniques, the first one that we chose to try is Pole Placement (or Full State Feedback).
However before doing so, the state space system must be extended with an integral action, in order to have zero error at steady state. Hence the extended model gets the following form:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} A_m & 0 \\ -C_m & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ v \end{bmatrix} + \begin{bmatrix} B_m \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} y^0;$$

or, alternatively:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{v} \end{bmatrix} = \bar{A}_m \begin{bmatrix} \mathbf{x} \\ v \end{bmatrix} + \bar{B}_m u + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} y^0;$$

If we look at the state space representation of the extended motor system, it can be noticed that it fulfils the requirements to apply the method mentioned above: in fact, $rank\left(\bar{A}_m, \bar{B}_m\right) = 3$ hence the pair $\left(\bar{A}_m, \bar{B}_m\right)$ is reachable, done exactly in the same way as in 2.1.3.
Furthermore, since the system is a single-input one, the condition to make use of the Ackermann's Formula is satisfied.
By applying the control law $u = -\bar{K}_{mot}\begin{bmatrix} x \\ v \end{bmatrix} + \eta$ the Pole Placement technique allows us to select the poles of the closed loop extended system, that is $\left(\bar{A}_m - \bar{B}_m\bar{K}_{mot}\right)$.

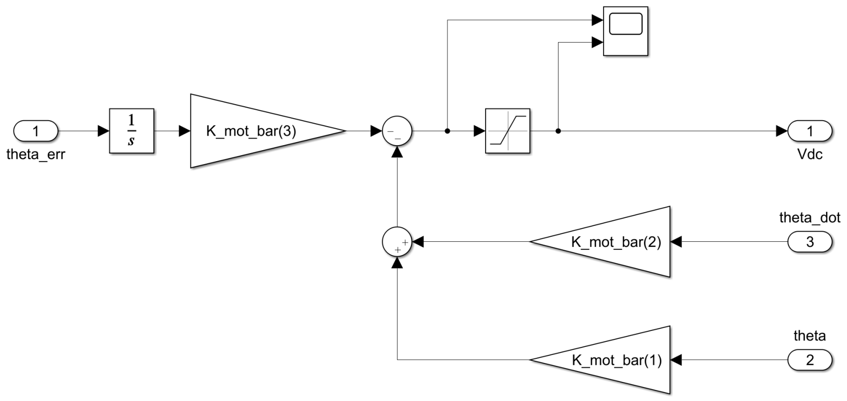


Figure 29: Simulink scheme of the Pole Placement controller for the theta's loop.

After a series of tries, the best result in terms of response of the system is obtained by placing the poles in $(-11, -80, -80.0001)$ (the third one is chosen so since *place* function in MATLAB doesn't allow you to select poles in the exact same location). Being the dominant pole in -11, the settling time is approximately about 0.45 seconds which is still lower than the maximum acceptable.
The chosen set of poles leads to the following $\bar{K}_{mot}$: $\begin{bmatrix} 137.5 & 2.3 & -1187 \end{bmatrix}$; it can be quickly noticed that the most weighted component is the integral one if compared expecially with the one associated with the second state, i.e. $\dot{\theta}$.
The correct design in terms of settling time is provided in *Figure 30*; notice that the response, in blue, is smooth and with zero steady-state error.
For completeness, are reported also the chirp and saw-tooth response, where the tracking of the reference is fine and without abrupt movements in both cases.

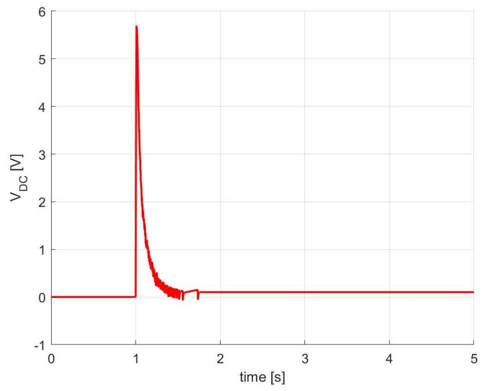


Figure 30: Pole Placement step response: motor voltage

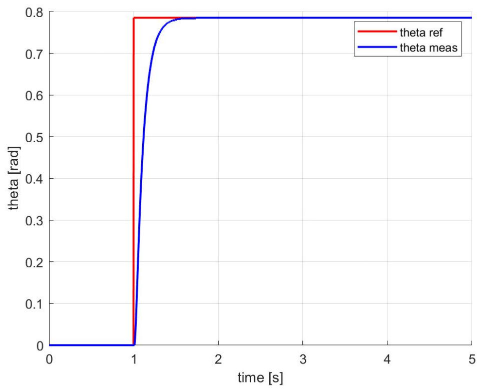


Figure 31: P.P. step response: angular position

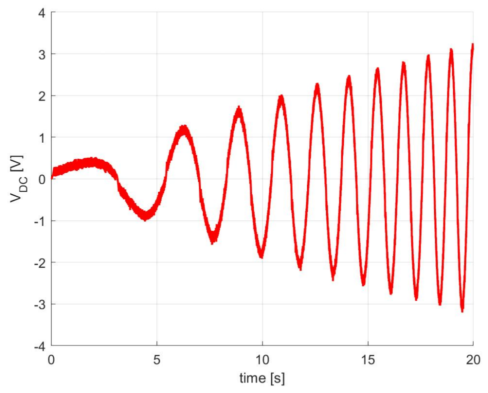


Figure 32: P.P. chirp response (0.01-1 Hz): motor voltage

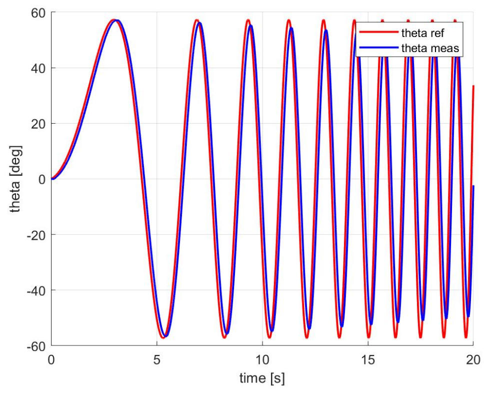


Figure 33: P.P. chirp (0.01-1 Hz) response: angular position

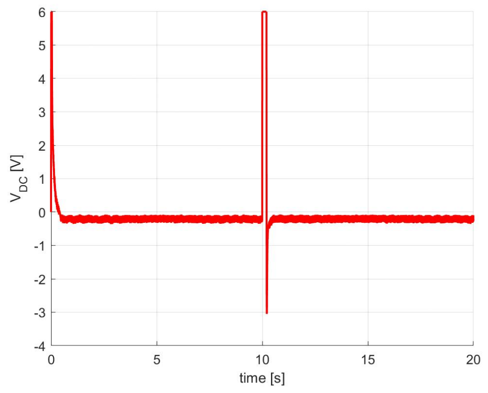


Figure 34: P.P. sawtooth response: motor voltage

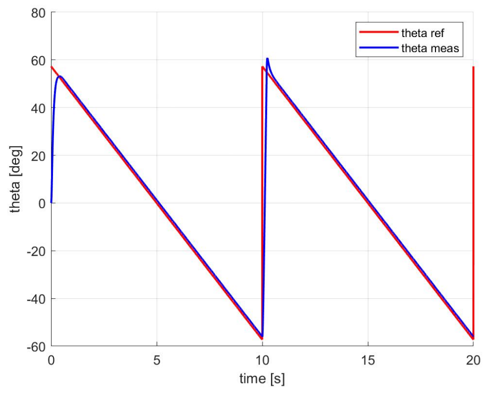


Figure 35: P.P. sawtooth response: angular position

5.1.3 LQI Control

For the control of θ we designed a Linear Quadratic Integral controller. It is based on the state space system of the motor already showed in the section 2.1.3.
This approach is conceptually similar to the Pole Placement one, so the same comments for the augmentation of the state and the controllability check still holds. The only difference is in the computation of the gains: we have used the *lqi* function of MATLAB that computes an optimal state-feedback control law for the given plant, receiving as inputs the weighting matrices Q, R, N. We have used the following matrices:

$$Q = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 50000 \end{bmatrix}; \qquad R = 0.1; \qquad N = 0;$$

So we decided to weight a lot the integral action with almost no feedback on $\dot{\theta}$. The LQI control gave us the best performance. We can see in *Figure 37* that the settling time is more or less 300 ms.
The tension on the motor, V_{DC}, reaches the limit and saturates but not too much. The chirp response shows us how the motor can follow the reference at different frequencies.

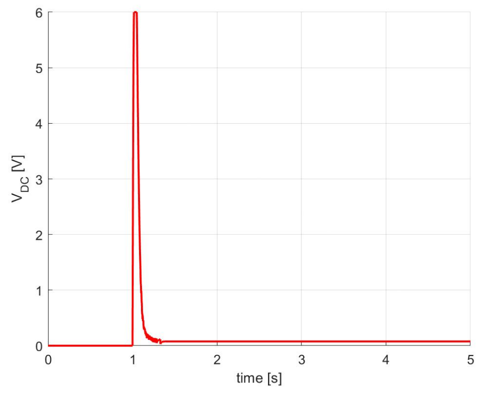


Figure 36: Linear Quadratic Integral Control step response: motor voltage

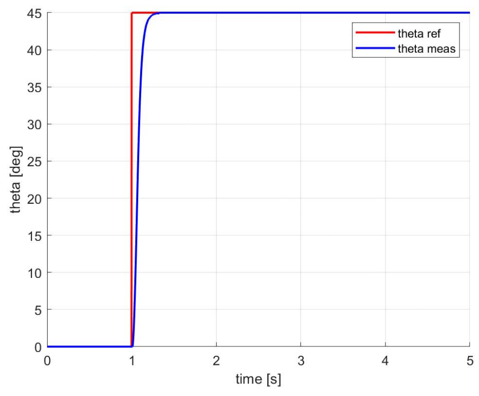


Figure 37: L.Q.I. step response: angular position

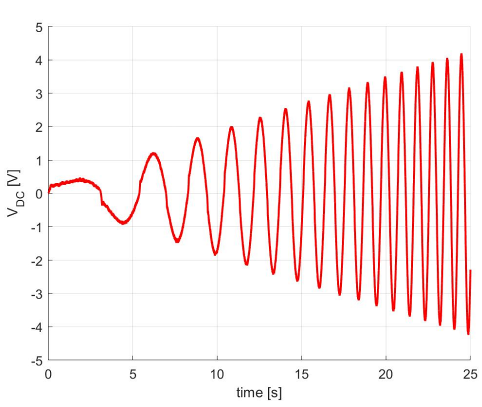


Figure 38: L.Q.I. chirp response: motor voltage

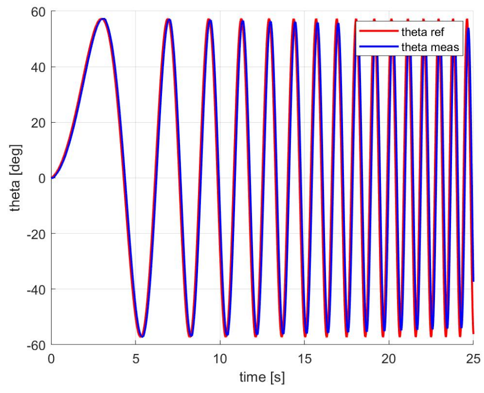


Figure 39: L.Q.I. chirp (0.01-1 Hz) response: angular position

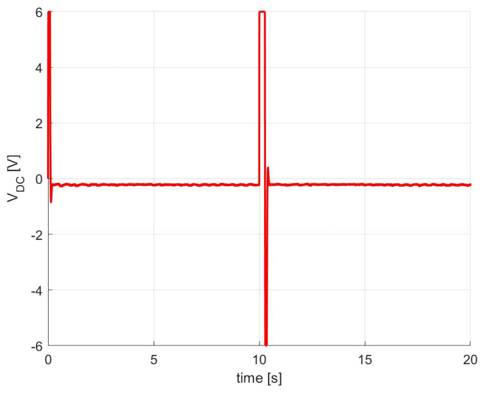


Figure 40: L.Q.I. sawtooth response: motor voltage

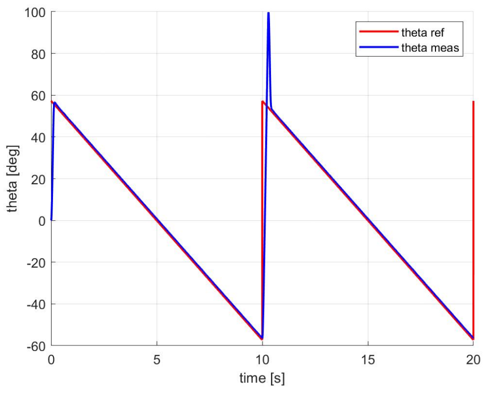


Figure 41: L.Q.I. sawtooth response: angular position

5.1.4 Conclusion

Looking at the test made on each control system type on the *theta* loop scheme is it possible to draw some conclusions on this first part of the task.
Firstly, the PID regulator guarantees a satisfactory response, widely meeting the requirements, although it shows some difficult in following the chirp signal. Furthermore, the good shape of the

response it is obtained through a prefilter action on the set point that makes more demanding the overall control scheme.
While, the Pole Placement technique let us obtain a quite robust control with almost the same settling time of the PID scheme (just slightly slower), and a better following of the chirp signal.
As expected, the best results are obtained through the LQI control scheme, which provides the fastest step-response and the finest tracking of the chirp signal. Albeit the sawtooth response of the LQI is the one that presents the highest overshoot, we still consider it as the best control method for the aims of the experiments on the whole system.
The step responses of the three control techniques used for the *theta* loop are represented synoptically in *Figure 42*.

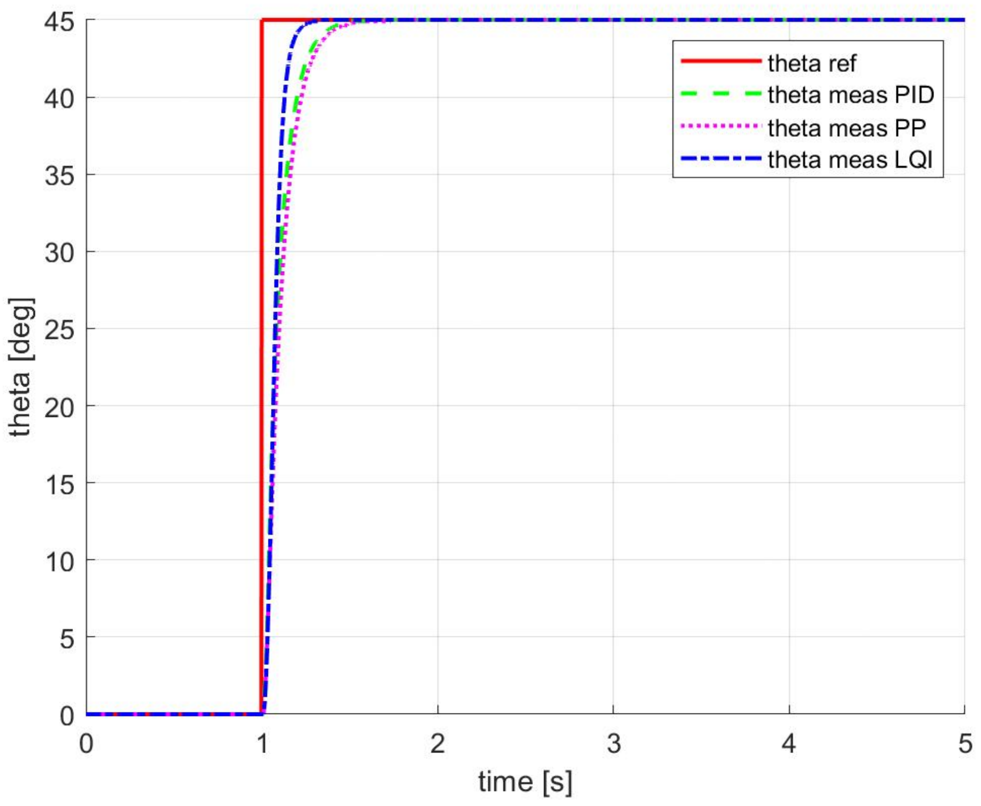


Figure 42: Comparison between the different *theta* control's schemes step responses.

5.2 Ball Position Control

Once we have closed the control loop in theta, we focused on the external position control loop. The design of this regulator is made on $G_{\theta 2x}$.

5.2.1 PID Controller

What makes the double integrating process special is that derivative action is actually necessary for stabilization. This is caused by the fact that the feedback system is unstable with proportional only controller. For this reason it is also impossible to use classical tuning method as the Ziegler and Nichols method.
A first attempt was to stabilize it through a PD controller. We thought that the two integrators of the system were enough to reach a zero steady-state error. Then, testing it on the physical system, we noticed a steady state error. This error is caused by the presence of the rolling friction between the ball and the beam.
So we decided to stabilize the system using the root locus method, finding the right position for the zeros and the poles and tuning the gain. This time we added an integrator to the controller.
The simulation works quite well, but then in the real system the response was different because of the disturbance in the x measurement. The continuous variation of the x measurement was

amplified by the derivative action of the PID and it causes a too aggressive reference for theta. The controller has zero steady state error but the theta reference is too nervous.
In particular there is an high frequency oscillation at steady state. We tried to solve it writing the PID in the parallel form, defining separately the Proportional, Integral and Derivative part. In this way we tried to tune each single action of the PID in order to reduce the derivative contribution. We found a quite good compromise, but reducing too much the contribution of the derivative part brings firstly to big oscillations and secondly to instability.
So we tried to apply the derivative action only to the controlled variable, in order to limit it, but this solution doesn't work well.

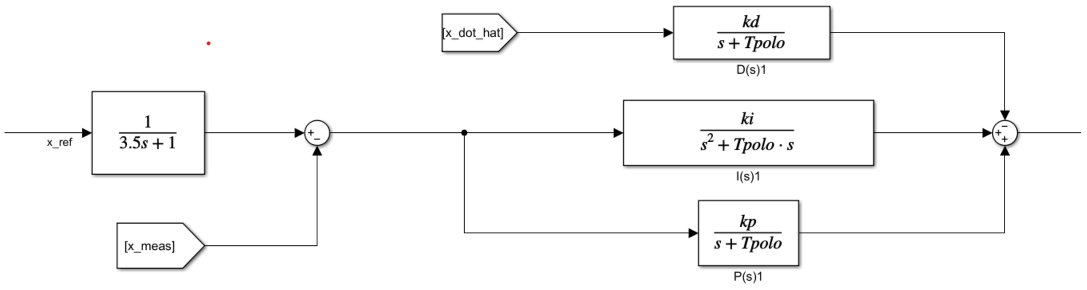


Figure 43: Simulink scheme: PI and Kalman Filter with "derivative" action

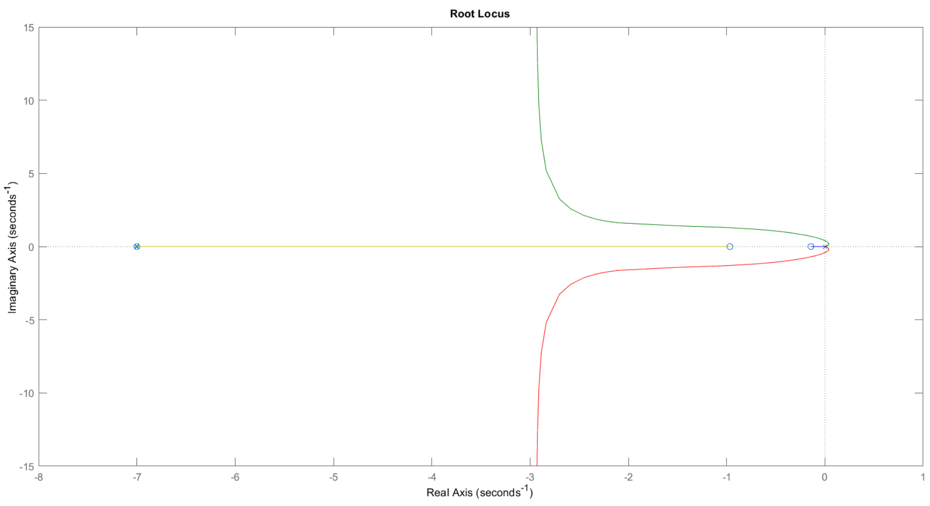


Figure 44: Root Locus of the system.

The main problem was the derivative action on x, so we decided to eliminate the derivative part and apply a proportional control on $\hat{\dot{x}}$, the velocity estimated by the Kalman Filter. We leave the PI control on x, that works really well.
The best value that we have found are $kp = 100, kd = 80, ki = 14, Tpolo = 7$. In this way we have an asintotically stable response with a phase margin larger than 45° and thanks to the prefilter without overshoot.
The result was good both in simulation with noise on x and in the real system.

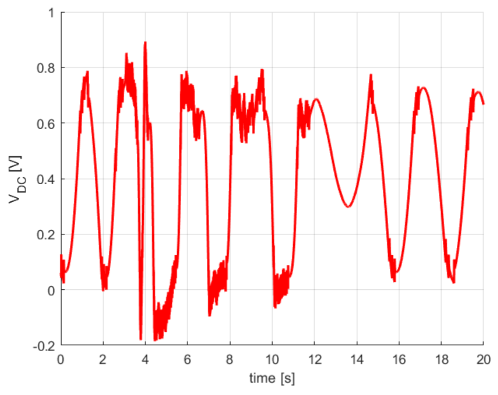


Figure 45: PID Step response: voltage

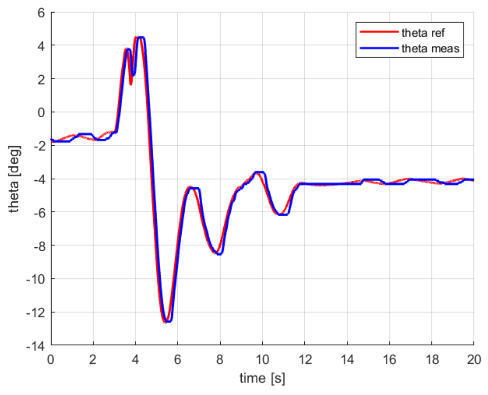


Figure 46: PID Step response: angular position

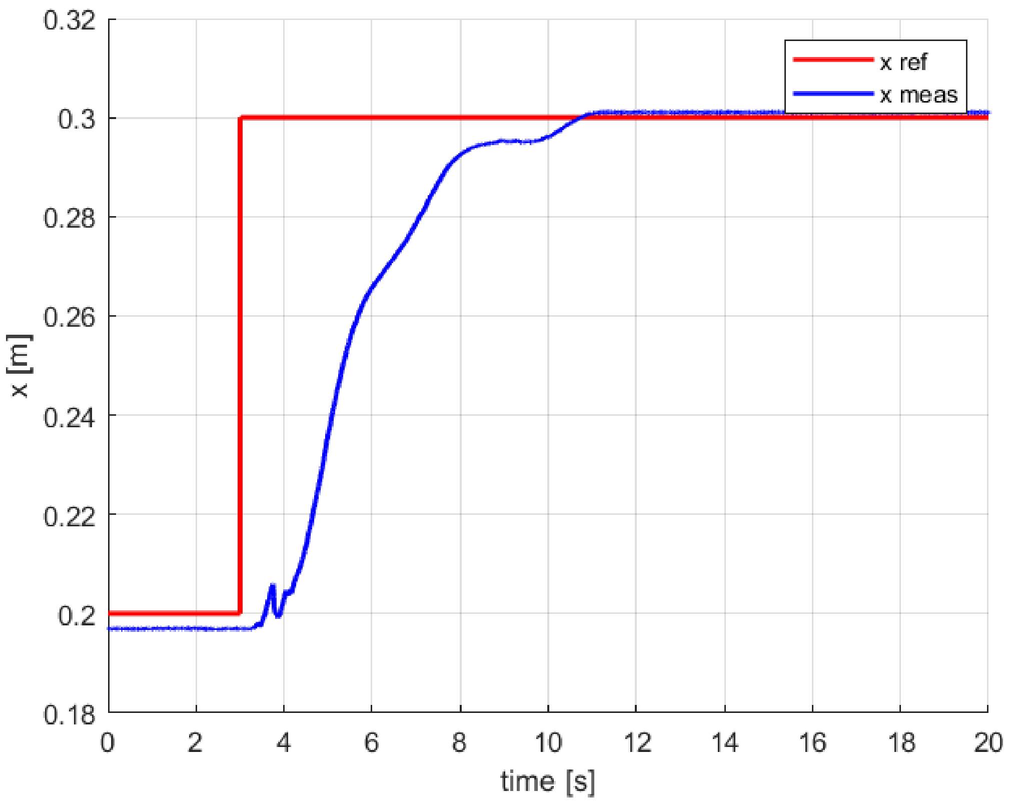


Figure 47: PID. Step response: ball position

In *Figure 48* we can see the controller behaviour when we hit the ball. We can see that the controller brings back the ball to the right position, the controller is quite slow due to a not too high integral action.

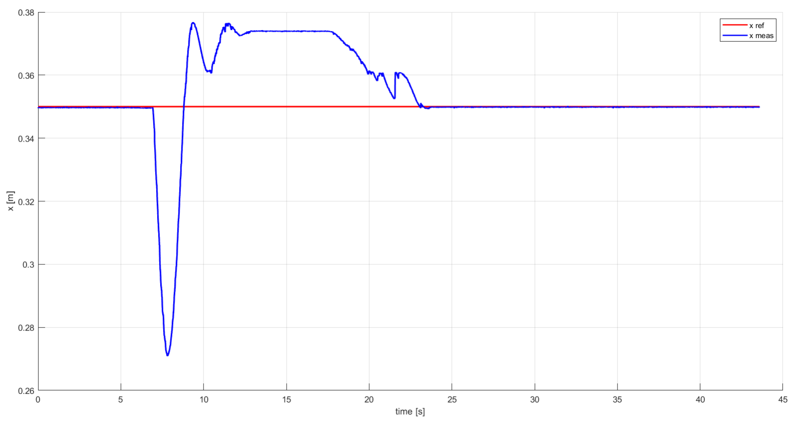


Figure 48: PID. Disturbance by hitting the ball

5.2.2 Pole Placement

Similarly to what was done in section 5.1.2, the ball and beam system is extended, adding an integral action.
Thus, the system treated with Pole Placement is the following:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} A_{bb} & 0 \\ -C_{bb} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ v \end{bmatrix} + \begin{bmatrix} B_{bb} \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} y^0;$$

that is:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{v} \end{bmatrix} = \bar{A}_{bb} \begin{bmatrix} \mathbf{x} \\ v \end{bmatrix} + \bar{B}_{bb} u + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} y^0;$$

Also in this case, the conditions to apply the Pole Placement technique and the Ackerman's Formula are fulfilled, since the system has just one input and the pair $(\bar{A}_{bb}, \bar{B}_{bb})$ of the extended system is reachable. Again the reachability test is performed in the same way as done in 2.2.4 for the original system.
Therefore, it is possible to compute the $\bar{K}_{bb}$ vector such that the poles of the closed-loop extended systems are in $(-0.75, -3.1, -3.10001)$. These numbers have been chosen since they resulted to be the best trade-off in terms of guaranteeing a fast enough response and avoiding an excessive saturation of the control variable theta. The respective $\bar{K}_{bb}$ obtained is $\begin{bmatrix} 54.85 & 26.73 & -27.72 \end{bmatrix}$; however, in this case the stress is put on the first state, that is the ball position x, while the integral action has a lower weight, though still dominant on the second state $(\dot{\theta})$.
In *Figures 50, 51, 52* are represented respectively the voltage on the DC motor, *theta* before (red) and after (blue) the 45° saturation and the system's response on a step from 0.2 m to 0.3 m. Eventually, *Figure 53* shows the recovery of the system after manually hitting the ball to simulate a disturbance effect.

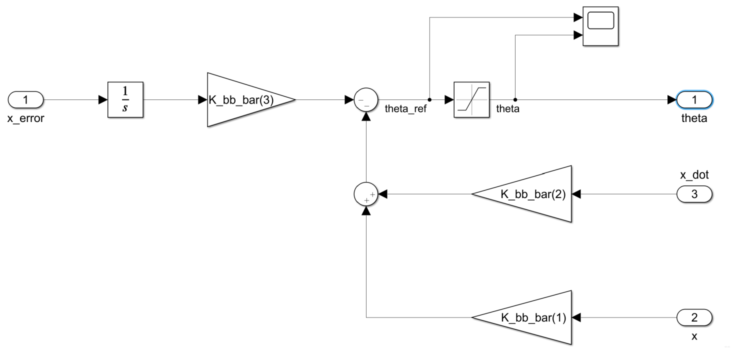


Figure 49: Simulink scheme of the Pole Placement controller for the ball position's loop.

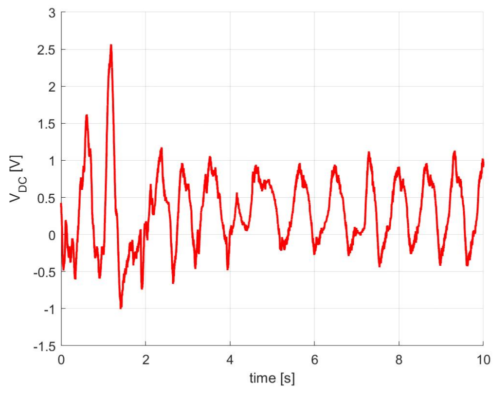


Figure 50: Pole Placement. Step response: motor voltage

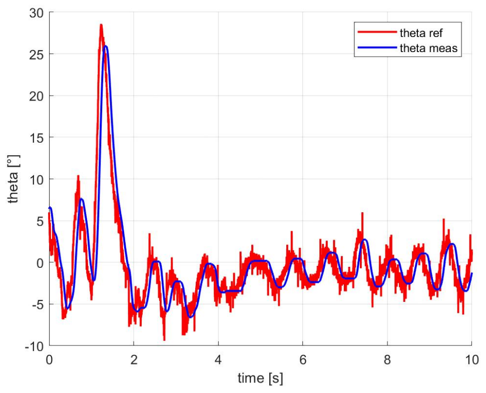


Figure 51: Pole Placement. Step response: angular position measured and estimated

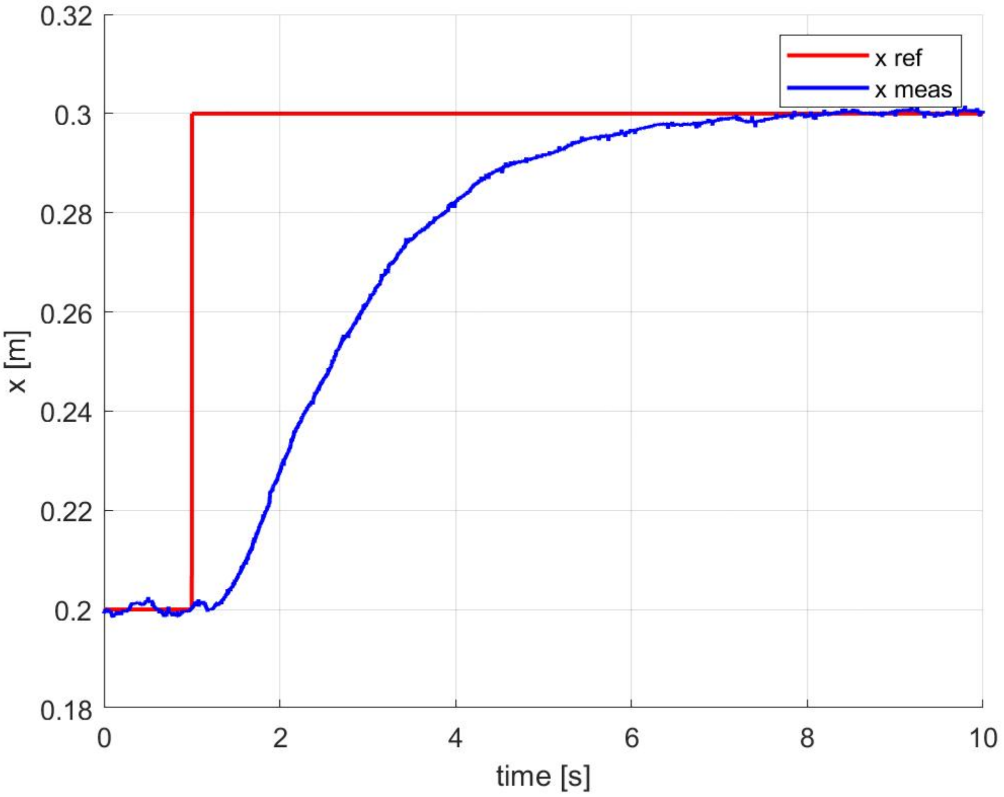


Figure 52: Pole Placement. Step response: ball position

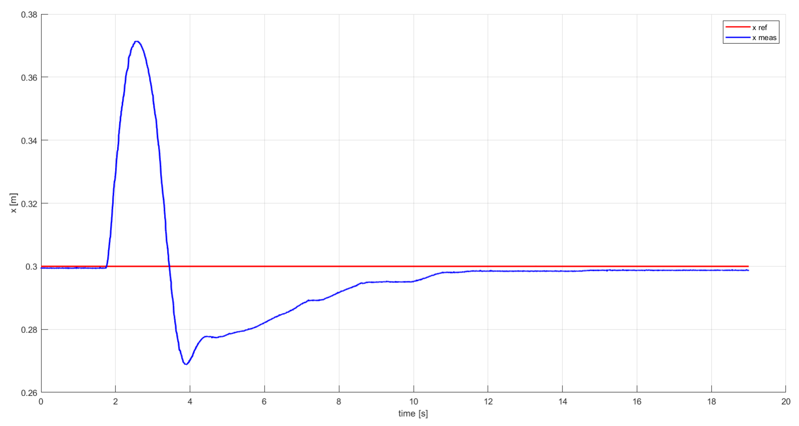


Figure 53: P.P. Disturbance by hitting the ball

5.2.3 LQI Control

Also for the control of the ball position x we designed a Linear Quadratic Integral controller. It is based on the state space system of the ball and beam already showed in the section 2.2.4 and adopt the same control scheme of the Pole Placement controller introduced in the previous section. We have used the *lqi* function of MATLAB that computes an optimal state-feedback control law for the given plant.
We have used the following matrices:

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \qquad R = 0.01; \qquad N = 0;$$

As it is possible to see from this matrices, we decided to weight more x and the error with respect to $\dot{x}$. This tuning was the one that guarantees the best response on the real system.
From the plots analysis, we can see that this controller satisfy all the self-given specifications, obtaining very good result in terms of settling time (lower then 5 s).

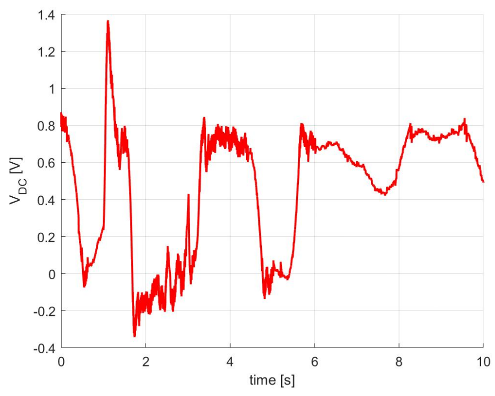


Figure 54: label.

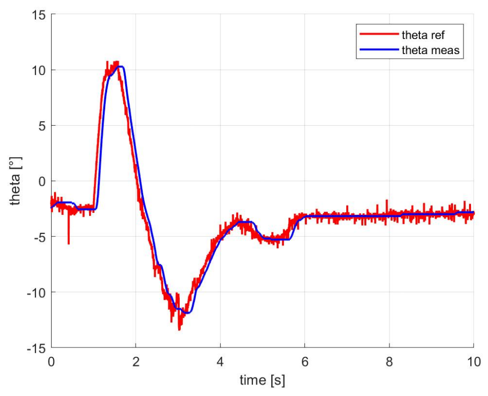


Figure 55: label

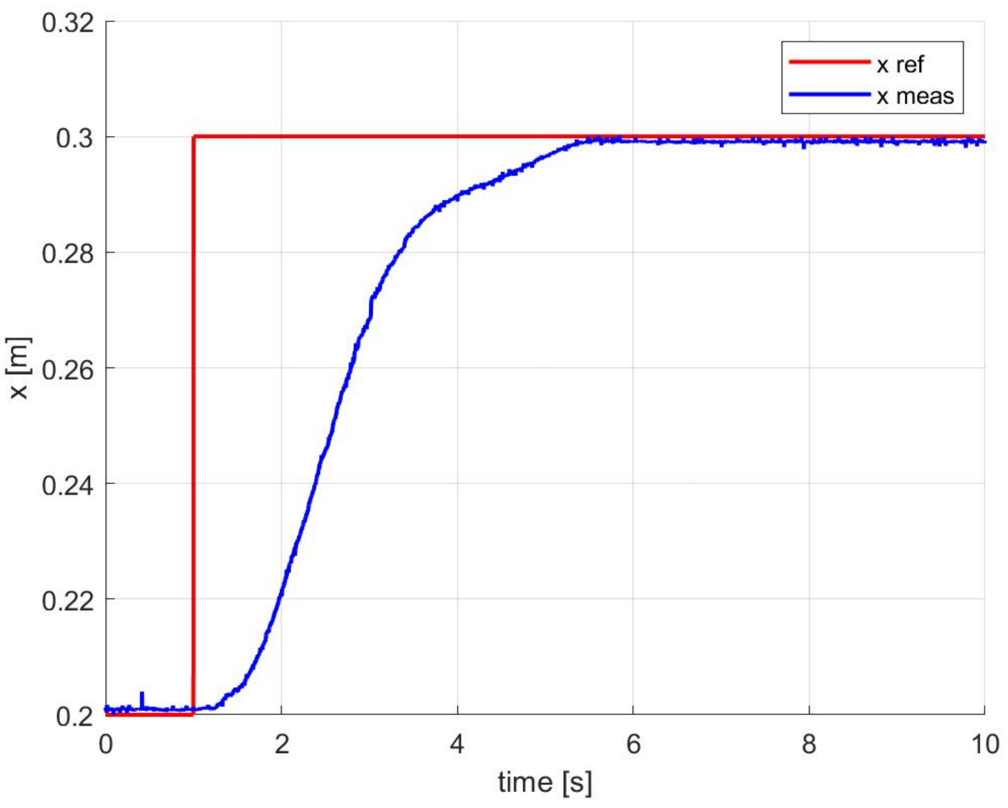


Figure 56: LQI Control Response of x.

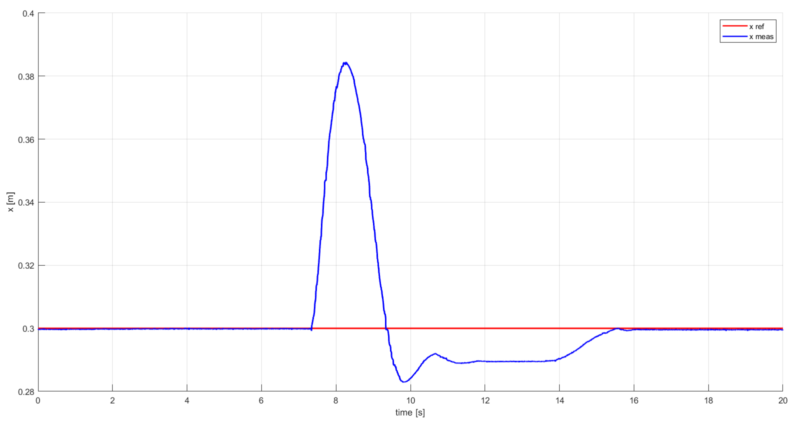


Figure 57: LQI response to disturbance by hitting the ball

5.2.4 Conclusion

Albeit not being an easy task, mainly because to the noisy position sensor, all the three control techniques designed for the ball position control provide an acceptable regulation of the system.
The PID control is the slowest, owing to the trade-off between having a relative short settling time and avoiding overshoots or inverse responses. Since its integral action parameter is quite low, it takes more time with respect to the other methods to reach the zero steady-state error. However, it has no major issues in recover from disturbances.
Moving on, the Pole Placement controller guarantees a faster and smoother response with few oscillation and zero error at steady state. Furthermore, it takes less time in recovering from external disturbances, whilst it struggle a bit in reaching exactly zero error again (probably due to a $\bar{K}_{bb}(3)$ parameter not big enough).

Eventually, again the LQI control clearly shows the best performance among the three, since it shows a desirable recovery robustness from external disturbances and it reaches steady state in a smaller time without error, even though with a slightly less smooth shape with respect to that resulting from the controller based on Pole Placement.
A final comparison of the step responses provided by the three control scheme can be seen in *Figure 58*.

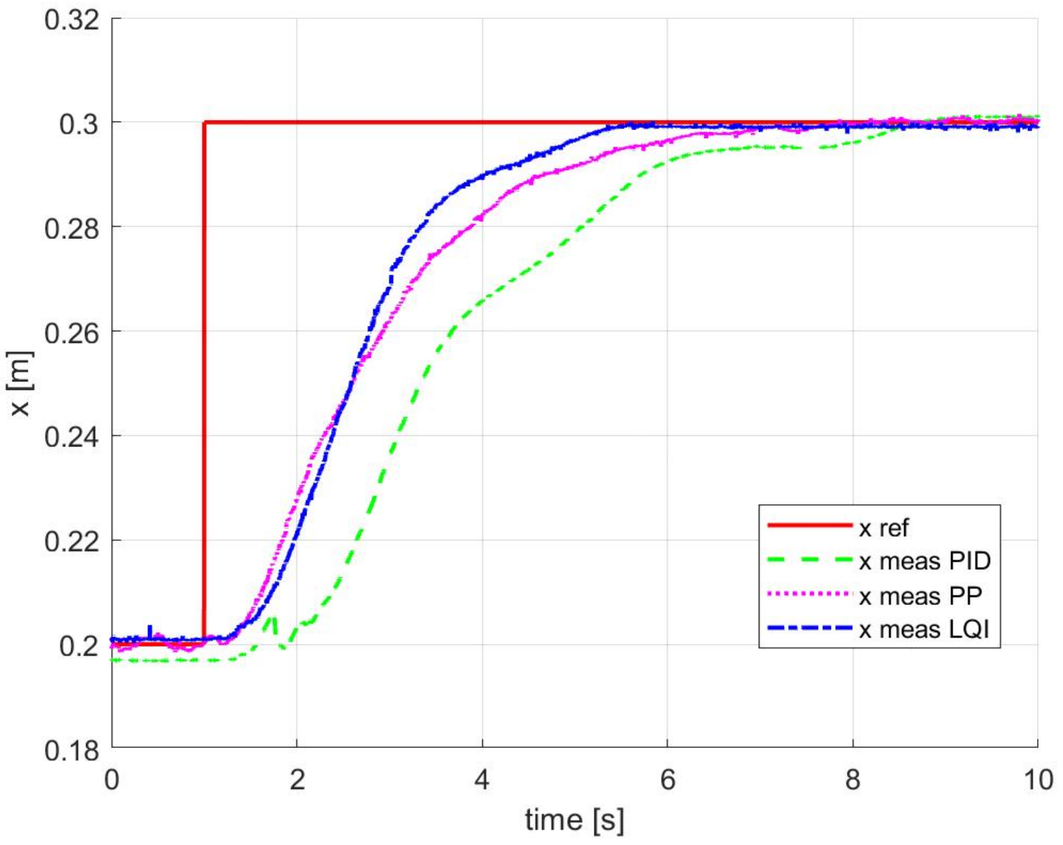


Figure 58: Comparison between the different x control's schemes step responses.

6 Advanced Control Techniques

6.1 Model Predictive Control

One of the most popular and reliable advanced control technique is the Model Predictive Control. The MPC offers optimal control at the cost of a quite considerable computational burden. Given the lack of time and the need to use the physical Ball and Beam system to tune the parameters, and considering also the differences in the computational time of the computers, the design of this controller has been particularly challenging. This is why we decided to tackle the problem from several different angles.

6.1.1 Matlab Toolbox

MATLAB MPC Toolbox offers a set of ready tools to build a MPC. We tested the control of the position of x, using LQI as motor control for the internal loop. Tuning with prediction horizon N=13 and a control horizon T=3 offers already a good response from the system, but it has a considerable steady state error. The MPC allows us to set the constraints of the angles already on the optimization, reducing the need to use saturation blocks or other techniques.
A strong drawback of using the toolbox is the lack of control on the inner work of the block and the difficulties in the choice of loss functions different from those proposed by Matlab.
An advantage of this block is the possibility to give a preview of the input to the MPC, realizing the well known *MPC with preview.* As expected the MPC with preview has a lower T_S and behaves better when the reference changes quickly or when we filter the step, but it still has a noteworthy steady state error.

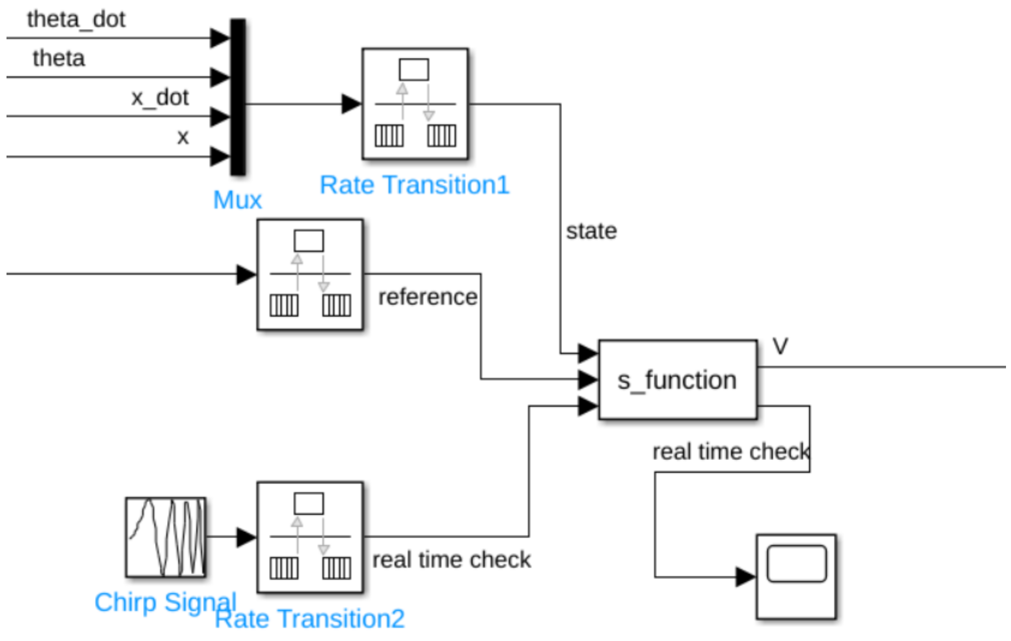


Figure 59: MPC as S-function with rate transitions and the three inputs: state, reference and a third one to check if it's running in real time.

6.1.2 CasADi

To solve the issues related to the lack of customizability we decided to build our own MPC from scratch, using the popular non-linear numerical optimization and algorithmic differentiation tool CasADi[3] on Python. One of the main advantages of using CasADi, other than the good range of supported optimization libraries, is the possibility to generate stand alone code in C that does not require external libraries when executed.

[3][1] https://web.casadi.org/

Given the additional computational complexity required by adding an LQI in the model of the system we decided to build this MPC without using our well tested internal controller for the motor. Instead this MPC controls directly the voltage, thus requiring a simpler model:

$$X = \begin{bmatrix} \dot{\theta} & \theta & \dot{x} & x \end{bmatrix}; A = \begin{bmatrix} 34.68 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0.26 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}; B = \begin{bmatrix} 59.33 & 0 & 0 & 0 \end{bmatrix};$$

$$\dot{X} = AX^T + B^T u;$$

Note that in this system we used 59 as the motor gain as the MPC gives inputs to the system in the form of steps and, in this specific situation, 59 works better for the reasons explained in 2.1.2.
The first step in the building of our MPC is the discretization of the system to get from an ordinary differential equation to a differential algebraic equation. This is done in the python script using the method *Runge-Kutta*. The discretization time T_S is a tunable parameter, it is preferable to stay under 0.5s but higher than 0.2s as more than 0.5s makes the system difficult to control while less than 0.2s may result in difficulties in using Simulink in real time.
From now on we will consider the system as discretized.
To have a good control of the whole system we decided to use a complex loss function that gave us control of all the states:

$$J_{MPC} = \sum_{k=0}^{N+1} Q_1(x_k - x_r)^2 + Q_2(\dot{\theta}_k)^2 + Q_3(\theta_k)^2 + Q_4(\dot{x}_k)^2 + R(u_k)^2;$$

The complete MPC formulation is:

$$\min_{u,X} J_{MPC}$$

subject to:

$$\begin{aligned} X_{k+1} &= AX_k^T + Bu_k; \\ u_k &\leq 6; \\ u_k &\geq -6; \\ \theta &\leq pi/4 \\ \theta &\geq -pi/4 \\ X_0 &= \begin{bmatrix} \dot{\theta}_0 & \theta_0 & \dot{x}_0 & x_0 \end{bmatrix} \end{aligned}$$

Where x_r is the reference position.
The tuning of the hyperparameters T_S, Q_1, Q_2, Q_3 and Q_4 (keeping $R = 1$) proved to be not doable using our simulation as we obtained some very different results from simulation to the real model. In simulation we were able to obtain perfect results considering also some disturbances on x, with a settle time $T = 4s$ and almost 0 stationary error using simple values such as $T_s = 0.3s$, $Q_1 = 1e2$, $Q_2 = Q_3 = Q_4 = 0$ and $R = 1$. The difference between simulation and real model can be caused by some non modeled disturbances (such as the weight of the ball and beam or the frictions) but also from the use of a different computer with different computational times or even by the optimizer as we will discuss in a moment. To test if the issues were generated by the computational burden required we modified the S-Function to give one more input that will be returned as it is only after the end of the optimization cycle (See Figure 59). Giving as input a chirp signal we did not find any discrepancy between the input and the output. This proves that for the tested parameters the simulation was able to be run in real time on the system.
To further explore different possibilities we tested having different Q_1 during the simulation, in particular we set higher values toward the end of the simulation (when k is close to N) to lower the steady state error. This approach did not lead to any improvement with respect to the normal version.
As we observed, thanks to the constraints, the motor position θ immediately is taken inside the safe zone from its initial position of -56°. this means that using the MPC it is not necessary to add any specific script that takes the motor to 0°.
As optimizer we had to use *QRQP*, a quadratic programming solver. *IPOPT* is another generally preferrable option for CasADi, as it is known to be more robust and reliable than *QRQP*, unfortunately it is also much harder to implement as a S-Function for Simulink. This choice could lead

to a less stable solution in some edge cases and in general it makes much more difficult to tune the MPC, as using some parameters the solver is not able to find a solution also in some simple simulated scenarios.
To have the C code run on Simulink we need to build a level-2 the S-Function: to do so we first used CasADi C code generator to build a stand alone script that took into account also *QRQP* (*F.c*), we then had to write another C function to describe the behaviour of the S-Function to Simulink (*S_Function.c*). MATLAB allows to compile the two scripts into a single function that can be then imported in Simulink using the following command:

```
mex S_Function.c F.c
```

Listing 3: Build S Function

More on the designing of the S-Function in the appendix: chapter 8.1. The S-Function in Simulink is called at the same frequency of the rest of the script, in our case every 20ms. This frequency is too high for the MPC that, as already mentioned, requires a time between 0.2s and 0.5s.
For this reason we had to use a rate transition block at the S-Function inputs (See *Figure 59*).
Below are shown some of the results that we got from the simulations. Starting from the result in python (*Figure 60*), the S-Function on Simulink (*Figure 61* and *62*) and finally the response on the real system (*Figure 63*).

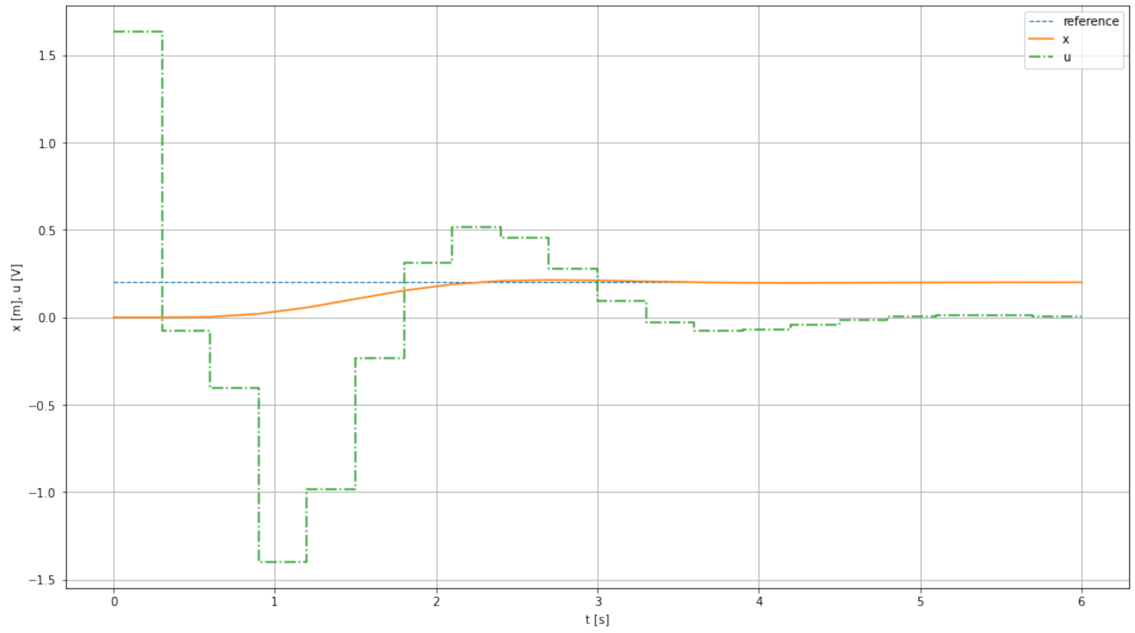
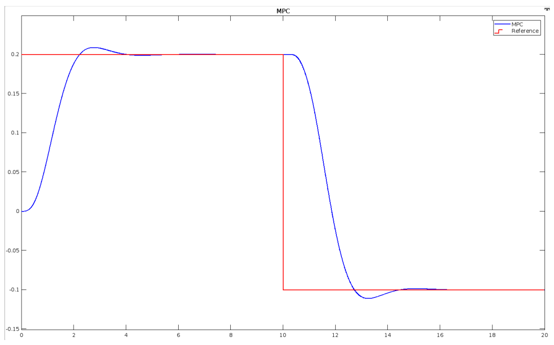


Figure 60: Response in Python. The green line represents the input in voltage given to the motor.



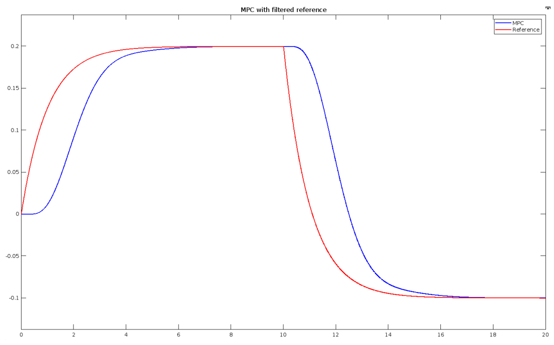


Figure 61: CasADi MPC simulated in Simulink.

Figure 62: With filtered reference in Simulink.

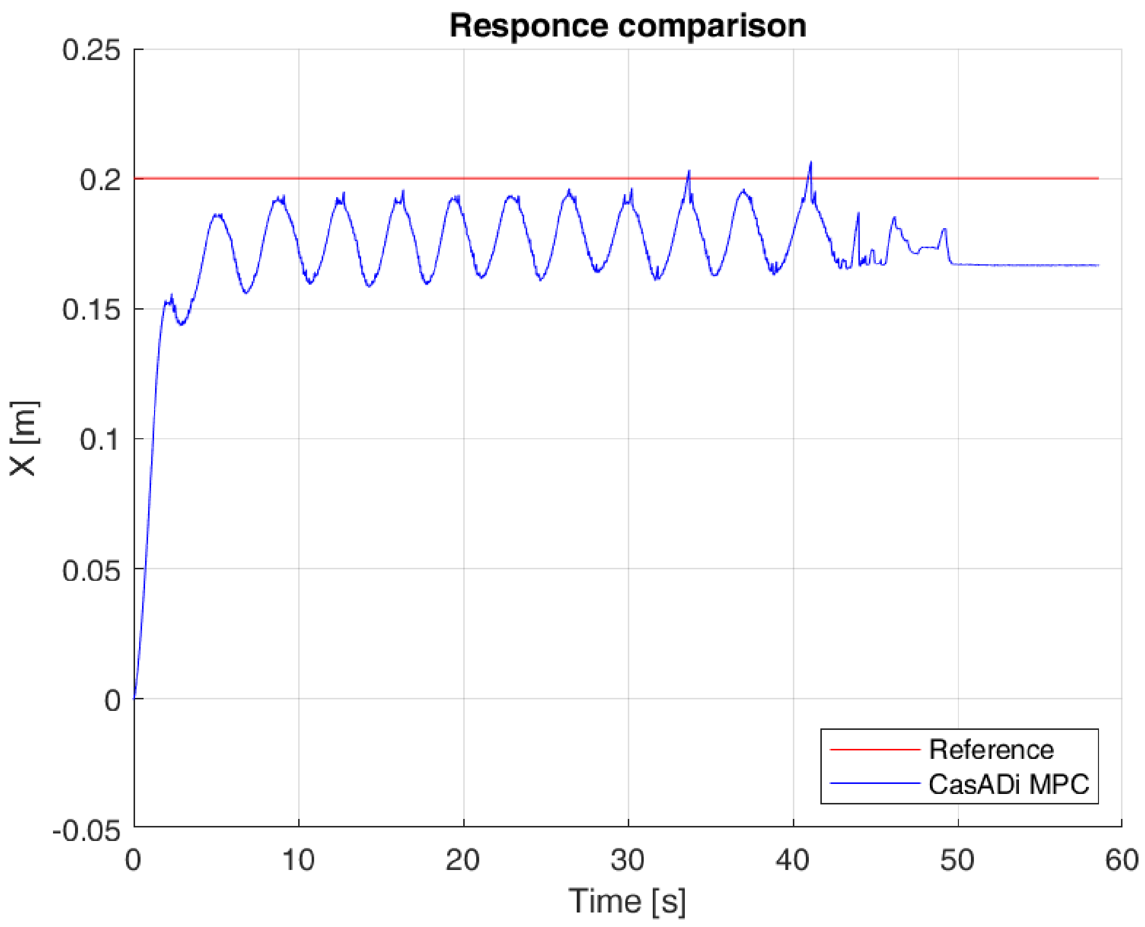


Figure 63: MPC on the real system.

We did not try to build a MPC with preview using CasADi as it is not trivial to have simulink give to the S-Function the future references in advance. To mitigate overshooting (already very low thanks to the parameters correctly tuned) we decided to filter the reference (*Figure 61* and 62). This caused the MPC to be a bit slower but guarantees a more robust response.
To conclude we present a list of possible reasons why the MPC does not work well in the real system. Fixing one of the following issues may already solve the problem entirely.

1. Not enough time to do a proper parameter tuning on the real system.
2. The model does not take into account the motor friction, giving directly a reference for theta and controlling the motor with for example LQI is very likely to solve the problem.
3. $\dot{\theta}$ estimation issues.
4. QRQP robustness issues.
5. Computational burden issues (very unlikely).

6.1.3 Sensitivity

With $Q_2 = Q_3 = Q_4 = 0$ and changing only $Q1$ the MPC is not able to stabilize the real system. The reason is that it gives too strong inputs to the system when $Q_1 = 1e3$ while it does not reach the reference when Q_1 is too small (notably smaller than 10). To solve this issue we tried with different values of Q_4: with $Q_4/Q_1 = 1$ ad $Q_1/R = 1e3$ the system behaves as in *Figure 63*: it stabilizes but it does not reach its reference. With $Q_4/Q_1 = 1e - 1$ the system behaves similarly to $Q_4 = 0$. With $Q_4/Q_1 = 1e1$ the system stabilizes far from the reference.
The tuning of the parameters prove this method to be effective. Unfortunately we did not have time to test enough parameters in the real system. Considering the unstable behaviour probably it would be better to test with $Q_1/R < 1e2$ and $R < Q_4 < Q_1$.

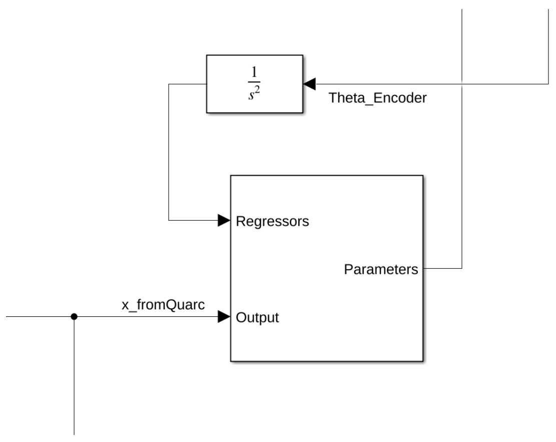


Figure 64: Least squares estimator block.

6.1.4 Adaptive MPC

Using our already built MPC we updated the model to leave μ as a parameter that can be given as input to the model, thus having the same model as before but with a parametric A:

$$A(\mu) = \begin{bmatrix} 34.68 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix};$$

μ is considered time invariant inside the model, and it is still linear.
As a simple approach to estimate μ we used the Least Squares Estimator block provided by Simulink. To mitigate the initial estimation errors, and knowing that the value of μ should be inside a specific range, we also decided to put a saturation to force $0.1 \leq \mu \leq 0.3$, being 0.1 and 0.3 extreme values found from the data we previously gathered. This approach works extremely well during simulation, reaching results close to *Figure 62* even starting from very bad estimations. Unfortunately we did not have time to test this approach on the real model.

6.1.5 Non linear MPC

We built a NPMC using the previously defined equations. We did not test it as the linear approximation works very well and we did not have time to tune the parameters. Furthermore this non linear model, despite being compiled in C, takes a good amount of computational power and requires a control horizon much smaller than the linear MPC, reducing the advantages of using the complete model.

6.2 Reinforcement Learning

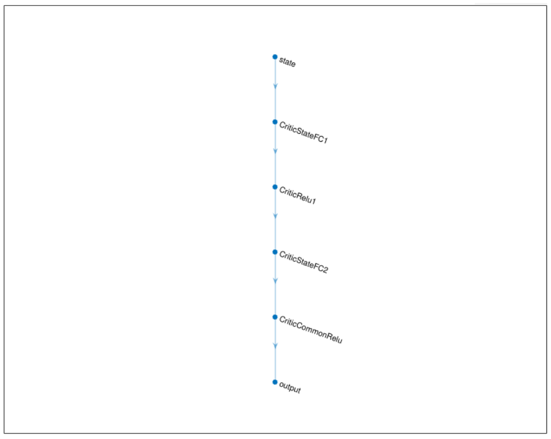


Figure 65: DQN model

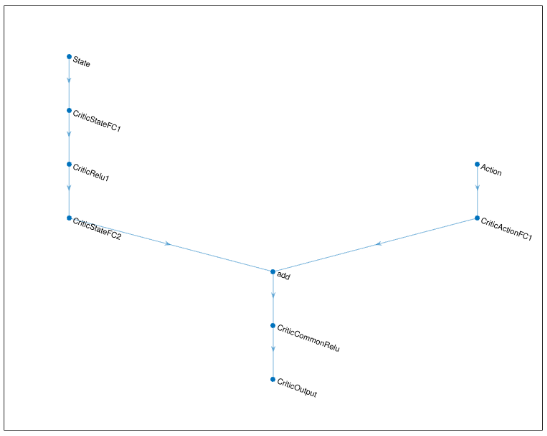


Figure 66: DDPG model

Using MATLAB Reinforcement Learning Toolbox we built a model to train an agent for the Ball and Beam, controlling only the position of theta and delegating to an LQI the control of the motor. We know that Reinforcement Learning is far from being the best control strategy for this model but we still decided to try it in simulation.
We built a continuous reward strategy that depends on the distance of the ball from its reference point:

$$R(x, x_r) = \begin{cases} -K_1|x - x_r|, & \text{if } |x - x_r| \geq 0.05 \\ (-K1 + K_2)|x - x_r|, & \text{otherwise} \end{cases};$$

With $K_1, K_2 > 0$ and $K_2 >> K_1$.
We started by discretizing the input range to be used with a Deep Q Network, we tried with this discrete input values: $-\pi/4, -\pi/6, -\pi/8, 0, \pi/8, \pi/6, \pi/4$.
Our DQN model follows very closely the one provided by MATLAB, with an input layer with two neurons for the two observations (x and x_r) and 2 fully connected layers with 24 and 48 neurons followed by RELU activations.
To overcome the problem of discrete inputs we tested also a Deep Deterministic Policy Gradient (DDPG), as seen in the figure this model is an actor-critic, the critic branch tries to guess what will be the reward given the action proposed by the actor. This model is more complex than the DQN and takes much longer to train (several hours) so we were not able to tune the network appropriately.
To build the environment for the models we used Simulink with the RL agent block set accordingly and giving the already described rewards and observations.
Unfortunately we were not able to obtain a satisfying result using this approach, this was caused by the long training times required and an always crashing MATLAB simulation. Having more time we would have tried to build a better reward function and tested different networks.

7 Conclusion

Throughout this report, despite the challenge represented by noises and modeling complexities, we were able to design and implement different working controllers using various different techniques, some of them for the first time on a real system. Despite the apparent simplicity of this system, the ball and beam is a challenging system from a control point of view because it can be modelled as a double integrator, one of the most fundamental systems in control theory with several practical applications: double integrated processes are really difficult to stabilize due to the presence of two poles in the origin and a starting phase equal to -180°. This makes it really difficult to design asymptotically-stable controllers with good performances.
However, starting from a good understanding of the theory and improving our expertise step by step, we not only respected the self given specification but we even improved our initial expectation, realizing controllers much faster than the original forecasts. Notably the LQI settling time is as low as 0.3s in the motor (against the 0.5s specified) and around 6s in the ball control (against 10s).
Our working method was to first test the controllers in simulation in the ideal case (no noise on measurement), then with noise on x only for the positon controllers and finally on the real system. This approach has allowed us to tune our controllers step by step to make them compatible with the real system.
We already highlighted some of our ideas to improve our results. The most noteworthy improvement that could further refine our result is the motor model. We did not include the gear friction in our model, neither we designed a weight compensation scheme, instead we delegated to our controllers and their integrators the correction of the little discrepancies that necessarily derive from our approximations. Other improvements come from our peculiar work on the MPC: removing the motor model approximations and with a better tune of the parameters we would be able to use on the real system the extremely fast and robust controller that we have seen on simulation, without having to worry about the computational time required. Moreover, as seen in chapter 6.1.4, with an adaptive MPC further discrepancies on the beam model can be resolved online, reducing the settle time and increasing the robustness. We are also convinced that, with the right parameters, our very fast algorithm based on CasADi could handle real time predictions using a complete non linear model, opening new and unexplored possibilities that come from the use of a non linear MPC. With Reinforcement Learning we explored an original way to tackle the problem. We don't believe that this controller would provide real improvements with respect to the other methods, but it could provide further insights on the system under analysis.
Between our proposed methods the LQI both in the θ and x is by far the fastest and the most robust and reliable controller, this is why it is our final go to choice. Being its optimal version it improves the already very good Pole Placement strategy from every point of view.

We would like to thank the professors for the help during this laboratory, since none of the group members ever had an experience on a real system before. Limited by time constraints we were not able to test all the ideas that we had and that we sketched but we nevertheless obtained great results using the tools provided and we are happy with our results, even though a deeper analysis could have been done to improve the results.

8 Appendix

8.1 On the design of a level-2 S-Function from CasADi in Python

Building an S-Function is not trivial task, for future reference we will here present the most important steps. We based our work on an article posted in the CasADi blog[4]. Our idea was to start from the MPC in python, generate a CasADi function and have it compiled in a format readable by Simulink. To do so we need to build a level 2 S-Function. Differently from level 1 functions, that only call MATLAB code, level 2 functions are designed to call some external code written specifically in a Simulink compilable format. Other than the correctly formatted C code to have an S-Function running we also need to have a second file which specifies the headers and a third that describes the expected behaviour of the level 2 S-Function. For more on the design of *S_Function.c* see the official MATLAB documentation[5].
Notable in the design of the S-Function is the callback that regulates the sample time: *mdlInitializeSampleTimes()*. During the lab we did not set a different sampling time, instead we let the block inherit the time from its driving block and we used a rate transition. If you plan to improve our design we would start by setting a different fixed sample time already in the S-Function to match that used in the sample time of the MPC, thus removing the need to use rate transitions.
Giving f as a CasADi function the following lines generate *f.c* and its headers in *f.h*:

```python
opts = dict(mex=True)
cg_options = dict()
cg_options["casadi_real"] = 'real_T'
cg_options["casadi_int"] = 'int_T'
cg_options["with_header"] = True
cg = CodeGenerator('f',cg_options)
cg.add_include('simstruc.h');
cg.add(f); # f is our CasADi function
cg.generate();
```

Listing 4: Generate f.c from Python.

In the generated script the value of the variable *casadi_real_min* is not specified and if not fixed this will cause an error at compilation time. To fix this issue just change the following code from NOT SPECIFIED to a small enough value, such as 0.0000001.

```c
#ifndef casadi_real_min
  #define casadi_real_min <NOT SPECIFIED>
#endif
```

Listing 5: f.c with NOT SPECIFIED still not edited.

As already specified in Chapter 6.1.2 the optimizer is already in *f.c* and there is no need for further external dependencies. This comes at the cost of being restricted to use only the optimizer *QRQP*. Building an S-Function with *IPOPT* is feasible, but it requires to have the CasADi runtime installed on the working machine[6].
The use of the generated code in Simulink is now rather straightforward. We first compile the code in a Simulink compilable format using the following command:

```
mex S_Function.c F.c
```

Listing 6: Build S Function

This will generate a file *.mexa* or *.mex64* depending on the operating system. This function can now be called from the S-Function block of Simulink, that will behave just as the function f designed on python. Note that on Windows it is also necessary to give explicitly the headers to the S-Function block writing f (with no extension) in the box *parameters* inside the block properties. Having done everything correctly, our S-Function will automatically adjust to display in a typical Simulink fashion the number of available input and output signals as in figure 59.

We published our complete notebook in an open source Github repository. It's possible to find the repo url in the footnotes[7].

[4] [1] https://web.casadi.org/blog/s-function/
[5] [2] https://it.mathworks.com/help/simulink/sfg/writing-level-2-matlab-s-functions.html
[6] [3] https://web.casadi.org/blog/mpc-simulink2/
[7] [4] https://github.com/giuliovv/ball_beam/blob/master/mpc_from_V.ipynb