POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Elettrica

Tesi di Laurea Magistrale

Model Predictive Control for Electromagnetic Actuators

Relatore
prof. Iustin Radu Bojoi

.....

Candidato
Fabio Mandrile

.....

A.A.2016/2017

Lehrstuhl für
Elektrische Antriebssysteme & Leistungselektronik
Technische Universität München
Professor Dr.-Ing. Ralph Kennel

Fabio Mandrile

Model Predictive Control for Electromagnetic Actuators

Supervisors

Prof.Dr. Ing. Ralph Kennel and M.sc. Ali El Hafni

Acknowledgement

I would like to thank all the people who supported me in München during my thesis work, in particular my supervisor Ali El Hafni, who guided me with patience during the last months. I would also like to thank Herr Schuster and all the team at the Lehrstuhl für Elektrische Antriebssysteme & Leistungselektronik, whose work leads to an efficient workplace. A special thanks also to Prof. Kennel, who gave me some interesting suggestions.

Thanks to Prof. Bojoi, my supervisor at the Politecnico di Torino, whose lectures during my university years were among the most precious.

Thanks to all my friends (Italian and the international, I met during my stay in Munich) for all the happy moments shared together.

Finally, a special thanks to my family and in particular my sister Giorgia, who greatly supported me during my all life.

Fabio Mandrile

Declaration

The work in this thesis is based on research carried out at the Institute for Electrical Drive Systems and Power Electronics, Technische Universität München (TUM) supervised by M.Sc. Ali El Hafni. It is all my own work unless referenced to the contrary in the text.

Contents

List of Figures

List of Tables

Thesis Outline

The aim of this project is to apply Model Predictive Control (MPC) techniques for the position control of a DC permanent magnet linear actuator. It could be used in valve control, especially in internal combustion engines. The position response should follow the reference trajectory, and has good disturbance rejection capabilities. The control scheme is implemented in MATLAB Simulink to create FPGA code, using the HDL coder package. In the MPC control technique, at every time step, an optimization problem based on a cost function is solved. Since solving optimization problems takes a lot of processing times, Finite Set MPC (FS-MPC) can solve this issue. In FS-MPC, the possible voltage vectors are evaluated and the one that leads to minimal cost is chosen to be applied in the next time step. This modern approach allows the customisation of the cost function and the minimum criteria, on the other hand the hardware required for the calculations performed must be much faster in respect to the traditional microcontrollers for PI, or similar, controls. Another challenge in MPC is the necessity of an accurate model of the system to be controlled to obtain good response and results. The system will use an observer (Extended Kalman Filter, EKF) to estimate both speed of the actuator and load applied on it, measuring position and current.

In the first chapter a brief review of the theory behind Model Predictive Control is carried out.

In the second chapter the control scheme is explained in its every part.

The third chapter presents the simulated control results.

In the fourth chapter the implementation procedure is analysed.

Finally in the fifth chapter, a summary in presented.

Chapter 1

Theoretical Background of MPC

In this first chapter, the theory behind the Model predictive control is presented. Moreover, a comparison with PID controllers is carried out, to highlight the most significant differences between these two control strategies. A good description of this topic has been made by Arne Linder [1, 2] and his approach is also followed in this thesis.

1.1 Classical PID Control

Before going deeper into the details of the MPC, it may be useful to make a brief summary of the widespread PID controller, which is probably the most used classical control technique, underlining its pros and cons. Its story dates back to the 1920s and since then it found wide usage in many different applications. This solution presents many advantages, starting from the simplicity of implementation, in fact it can be implemented in both analog and digital way without much effort. In analog solutions the reduced number of components and the not needed knowledge of the model of the system to be controlled during the operation represented a key factor in the early implementations. When digital controllers were introduced, the reduced computational effort allowed them to keep their privileged role in the control engineering.

On the other hand, PIDs have mainly two limitations. First of all, they are SISO controllers, so they only manage to control one physical quantity (e.g. current) at

the same time. This leads to cascaded structures, such as in Figure 1.1.

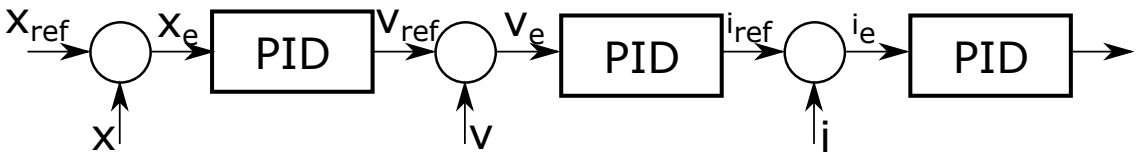

Figure 1.1: Cascaded PID structure example.

The second disadvantage are the poor performances in controlling non linear systems or systems with variable parameters. For these reasons the basic PID structure has been modified and upgraded in time, adding for example feed forward control.

1.2 Predictive Control

Moving into more recent times, after the introduction of digital devices and control techniques, the idea of using the plant model to pre calculate the inputs to the system came out. [2] Starting from the work of Emeljanov (1969), and then especially in the 1980s the first predictive control techniques were introduced, such as Direct Torque Control. Finally in the 1990s the first applications on electric drives were developed. The base principle of the predictive control is shown in Figure 1.2 and it is common to every predictive control technique.

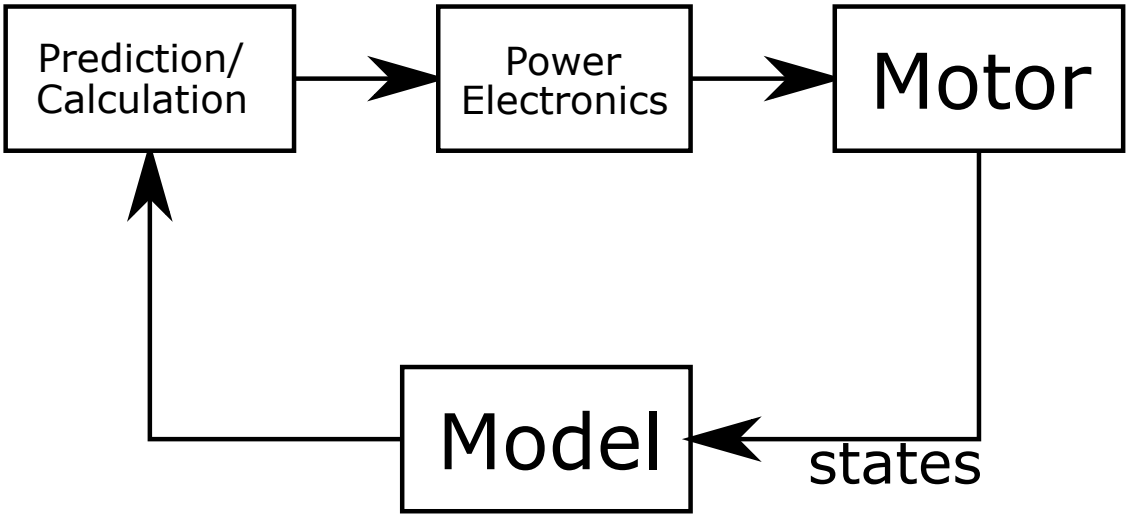

Figure 1.2: Predictive control base structure.

The machine states are measured (or observed) and then fed to the machine model block, which calculates the present state of the device and sends this information to the core component of the control: the prediction and calculation block. This part takes care of choosing the optimal actuating variable to send to the system (e.g. reference voltage for the power converter) according to the reference value and the desired optimization criteria.
Subsequently a further classification can be done based on three categories: [2]

- basic functional principle
- prediction horizon
- inverter control

The first classification is done on the basic functional principle:

- Hysteresis-based predictive control
- Trajectory-based predictive control
- Model-based predictive control

In this thesis only the latter is analysed and implemented.[1]

1.2.1 Model-based Predictive Control (MPC)

MPC are a family of controllers, all based on the same principle of optimizing the actuating variable, using a mathematical model of the plant and a cost function. The first applications date back to the 1970s, but only in the recent years applications to electric drives came out.

In Figure 1.3 the basic working principle is shown; the predicted behaviour of the system is calculated using the model. The response can be divided into two terms, free and forced. The free response is the predicted behaviour of the system, if the actuating variable are supposed zero. The forced response is an additional

[1] For a wider description this book by Linder [2] can be read.

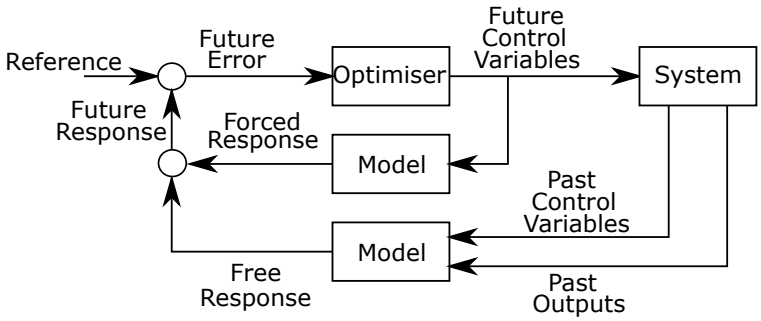

Figure 1.3: MPC working principle. Adapted from [1].

component calculated using the future actuating variables. As it is known, the global response for linear systems can be calculated via superposition of these two terms. The prediction of the future response is worked out up to the prediction horizon N, which represents the final prediction time of the control. For each of those predictions, a predicted error is calculated, knowing the future reference (set by the user). Then the future actuating variables are chosen by the optimiser block, knowing the cost function formulation (which means the optimisation criteria) and respecting the constraint applied to the control (i.e. maximum current). As it can be seen in Figure 1.3, the past values of actuating variables and outputs are used in the calculation. This way, a feedback from the system is obtained.

Since the prediction and optimization phases require high calculation resources, the control horizon N_u can be introduced. The idea behind it is that the controller output remains constant when steady state is reached, and this equilibrium is supposed to be gained after N_u steps. Figure 1.4 illustrates this concept.

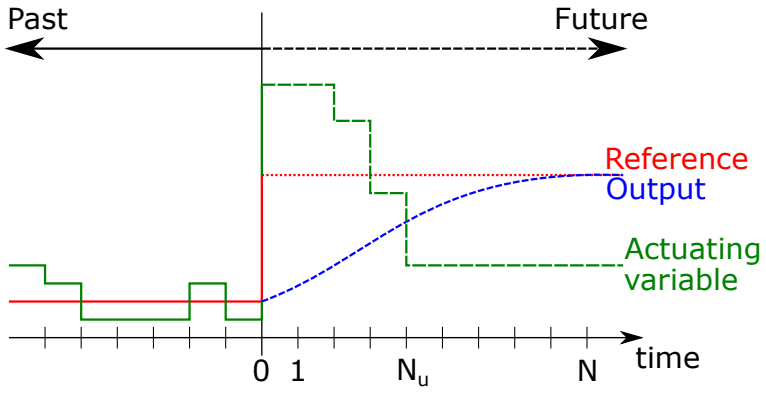

Figure 1.4: Control and prediction horizon. Adapted from [1].

Unfortunately, even with these simplifications the computational requirements are still demanding. For this reason the MPC has been widely utilized in the control of chemical or other slow process plants, where the settling times are in the order of minutes or hours and there is plenty of time for every calculation.
In the electrical engineering field, this is still an open topic and a lot of research is carried on, not only in electrical drives, but also in power electronics. [3–5]

Chapter 2

Control Scheme Design

In this chapter the control scheme design is explained and every element is described and analysed in detail.

The scheme proposed is presented in Figure 2.1 is divided if four parts: first on the left the FS-MPC, the brain of the system. Its role is to send the control signal to the switches of the H-Bridge. The controller receives the reference position from the user, as well as the measured current i and position x values. Moreover speed v and the load applied on the actuator F_L are observed by an Extended Kalman Filter (bottom right) using the voltage U applied to the machine and the measured quantities.

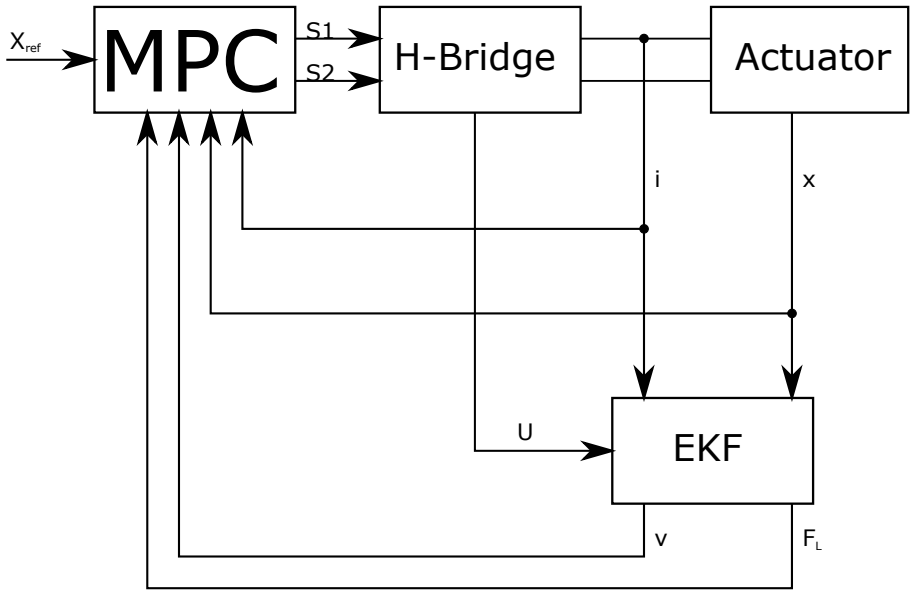

Figure 2.1: Global control scheme.

All the system components will be described in detail in the following sections.

2.1 Actuator Model

The first element to present is the electromagnetic actuator, in order to describe its model, which is then used by both the EKF and the FS-MPC. The actuator used in this project is the model 6033 SP011, produced by Magnetic Innovations. The datasheet characteristics are shown in Table 2.1. It must to be noted that the values of the back e.m.f. and force constant are the same. The centre position is assumed 2.5 mm, in an operating region between 0 mm and 5 mm.

Table 2.1: 6033 SP011 actuator parameters.

Parameter	**Value**
Stroke	8 mm
Force constant K_f	$K_f(x) = -0.12 \cdot x^2 + 8.3\,\frac{\mathrm{N}}{\mathrm{A}}$, x in mm with $x = 0$ at the centre position
Back e.m.f constant K_e	$K_e(x) = -0.12 \cdot x^2 + 8.3\,\frac{\mathrm{Vs}}{\mathrm{m}}$, x in mm with $x = 0$ at the centre position
Reluctance force	3.5 N
Coil resistance R	1.4 Ω
Coil inductance L	1.1 mH
Max. operating voltage	48 V
Mass m	0.13 kg
F continuous (middle position)	38.4 N
F peak (middle position)	259 N (Force applied during 5 s)

2.1.1 Continuous Time Model of the Actuator

An electromagnetic actuator has a structure very similar to a DC motor: a DC winding at stator and a moving iron with a magnet, instead of the rotor.
The well known stator electrical equation is

$$u_s(t) = R \cdot i(t) + \frac{\partial \lambda}{\partial t}, \tag{2.1}$$

where u_s is the voltage applied to the stator winding, R the winding resistance, i the stator current and λ the magnetic flux concatenated by the winding.

It is possible to write the magnetic equation of the flux as

$$\lambda = L(x) \cdot i(t) + \Lambda_{pm}, \tag{2.2}$$

where L is the inductance of the coil, dependant on the position x of the moving iron and Λ_{pm} the permanent magnet flux. Substituting it in (2.1),

$$u_s(t) = R \cdot i(t) + L \cdot \frac{\partial i}{\partial t} + \frac{\partial L}{\partial x} \cdot \frac{\partial x}{\partial t} \cdot i \tag{2.3}$$

is obtained.

The last term of (2.3) represent the speed dependant back e.m.f. With the back e.m.f. expressions described in the datasheet it becomes

$$u_s(t) = R \cdot i(t) + L \cdot \frac{\partial i}{\partial t} + K_e \cdot v, \tag{2.4}$$

where v is the speed of the mover.

Then the force applied by the actuator can be expressed as

$$F(x,t) = K_f(x) \cdot i(t) \tag{2.5}$$

and the mechanical equation of the system is

$$F(x,t) - F_L(t) = m \cdot \frac{d^2x}{dt^2}, \tag{2.6}$$

where F_L represents the force load applied externally to the actuator.

Since $K_e = K_f = -0.12 \cdot x^2 + 8.3$, the system is non linear. This leads to use more complex structures in the observer and in the control.
After having taken some measurements, the expression of $K_e = K_f$ was corrected to

$$K_e = K_f = k_q \cdot x^2 + k_s \cdot x + k, \tag{2.7}$$

with $k_q = -0.3335$, $k_s = -0.3652$ and $k = 8.165$. These coefficients were obtained from the interpolation of the measured characteristic, which can be seen in Figure

2.2.

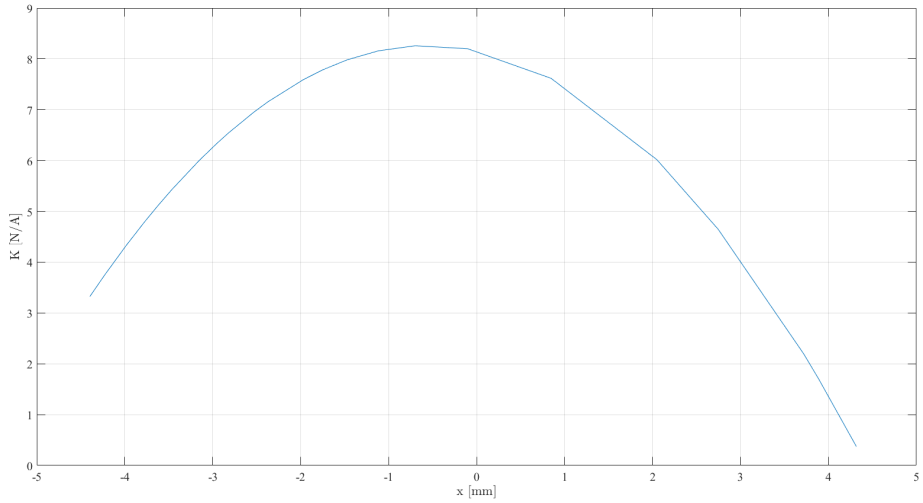

Figure 2.2: Back e.m.f. and force constant characteristic.

The equations can be summarised as follows:

$$\frac{d}{dt}\begin{bmatrix} i \\ v \\ x \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} + \begin{bmatrix} g_1 \\ g_2 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} \cdot i - K_e \cdot v \\ \frac{K_e}{m} \cdot i \\ v \end{bmatrix} + \begin{bmatrix} \frac{U}{L} \\ -\frac{F_L}{m} \\ 0 \end{bmatrix} \tag{2.8}$$

In Figure 2.3 the block diagram of the actuator is presented.

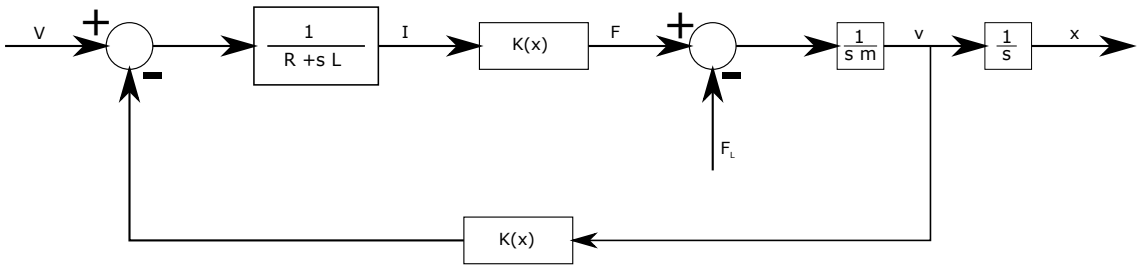

Figure 2.3: Actuator block diagram.

2.1.2 Discrete Time Model of the Actuator

Since the control is digital, and so is the EKF, a discrete time formulation of the actuator model is necessary. To obtain it, the forward Euler discretization method

is used.

The derivatives are substituted with:

$$\frac{dX}{dt} = \frac{X_{k+1} - X_k}{T_s},$$

where k is the time index and T_s is the sample time of the discrete time system. Since the model is non linear, it is described by the equations $f(i, v, x)$ and $g(U, F_L)$ as follows:

$$\begin{bmatrix} i_{k+1} \\ v_{k+1} \\ x_{k+1} \end{bmatrix} = \begin{bmatrix} f_{1_d} \\ f_{2_d} \\ f_{3_d} \end{bmatrix} + \begin{bmatrix} g_{1_d} \\ g_{2_d} \\ 0 \end{bmatrix} = \begin{bmatrix} (1 - \frac{R}{L}) \cdot i_k - K_e \cdot \frac{T_s}{L} \cdot v_k \\ K_f \cdot \frac{T_s}{m} \cdot i_k + v_k \\ x_k + T_s \cdot v_k \end{bmatrix} + \begin{bmatrix} \frac{T_s}{L} \cdot U_k \\ -\frac{T_s}{m} \cdot F_{L_k} \\ 0 \end{bmatrix} \tag{2.9}$$

2.2 H-Bridge Model

The second element described is the H-Bridge. In the actual system the inverter model SWM048 is connected to the FPGA control board optically [6]. The H-Bridge circuit is visible in Figure 2.4. In the simulation it was considered as ideal, and a logic model of the device was constructed.

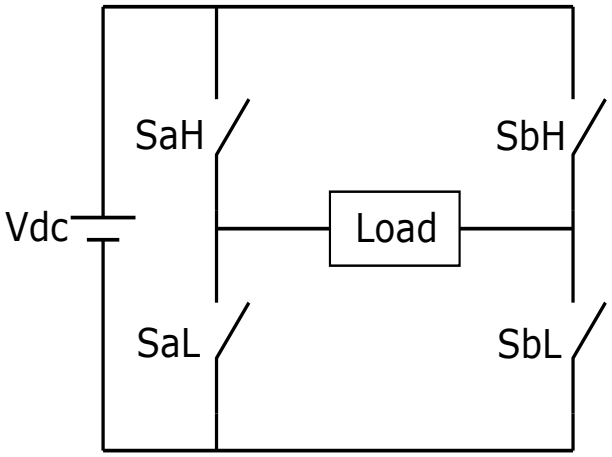

Figure 2.4: H-Bridge structure.

In Table 2.2 is possible to see the switching states of the inverter.

Table 2.2: H-Bridge switching states.

Switch states [S_a,S_b]	V_{Load}
[1,0]	$+V_{dc}$
[0,1]	$-V_{dc}$
[0,0]	0
[1,1]	0

The model used receives the S_a and S_b leg state and produces a voltage according to the table presented. No voltage drops, switching delays, dead times and other real characteristics are considered.

2.3 FS-MPC

Finite Set MPC is a particular variant of Model Predictive Control. In case the system inputs are limited to a finite number, then it is possible to decide which one is the optimal by trying all of them. In this case three voltages can be applied: $+V_{dc}$, 0, $-V_{dc}$. An interesting analogy to explain the logic behind the FS-MPC can be made with the game of chess: at every move the player takes a look at the present situation, having a certain goal in mind, and tries to predict the possible future moves, knowing the rules of the game. Then he decides which move is the best: the one bringing him closer to win the game. Next turn, he repeats the process, considering the new positions of the pieces. In a similar way the controller receives the states of the actuator at a certain time. Knowing the target position and the possible voltage to apply, it predicts the future behaviour of system using the mathematical model and chooses the voltage which leads the machine closer to the reference. In Figure 2.5 it is possible to see a visual example of this process. The three voltages lead to different positions and the controller chooses the sequence leading closer to the set target.

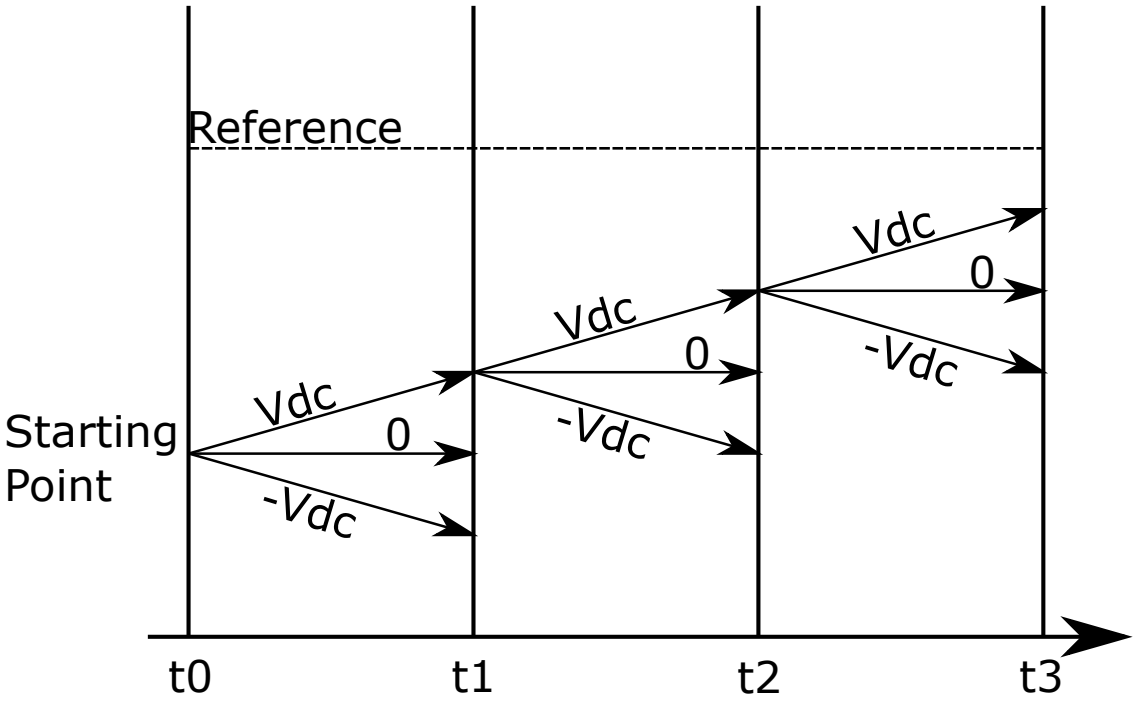

Figure 2.5: FS-MPC working principle.

2.3.1 Voltage List Creation

The list of possible applicable voltages depends on the chosen prediction horizon, the number of steps predicted and analysed by the controller. In this thesis a prediction horizon $N = 3$ was chosen, as a good compromise between computational effort, prediction precision [1] and quality of the control decision[2].

The process of creation of the lists of the possible applicable voltages follows these rules:

- The lists depend on the previously applied voltage.
- The only possible voltage transitions are: V_{dc} to 0 and reverse, $-V_{dc}$ to 0 and reverse. V_{dc} to $-V_{dc}$ is not allowed, in order to decrease the ripple on the current.

In Figure 2.6 it is shown an example of the tree of possible voltages.

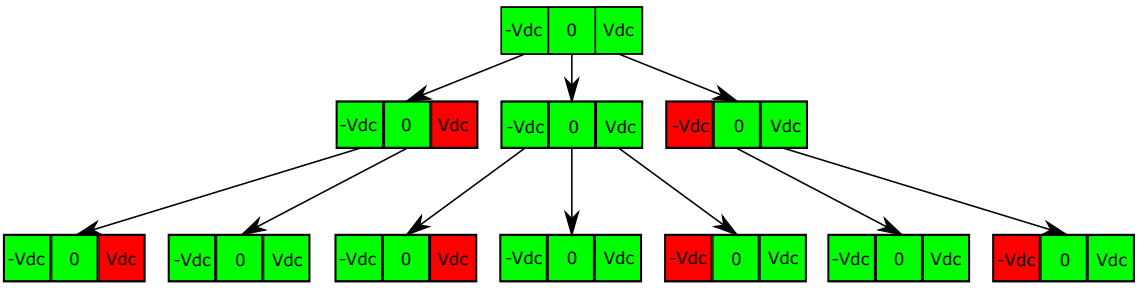

Figure 2.6: Voltage tree example. The previous voltage is supposed 0.

As it is clear from the picture, there are 17 different voltage sequences, when the previous voltage is 0. If the previous voltage is V_{dc} or $-V_{dc}$, then the possible paths are only 12.

2.3.2 FS-MPC to Control the Electromagnetic Actuator

In this subsection the implemented FS-MPC algorithm is discussed.

The block diagram of the algorithm can be seen in Figure 2.7.

[1] Since the model proposed has some uncertainties, the wider the prediction horizon, the bigger the chain effect on uncertainty becomes.

[2] With a narrow horizon the different voltage sequences may not be much different from one another, hence the importance of a longer horizon.

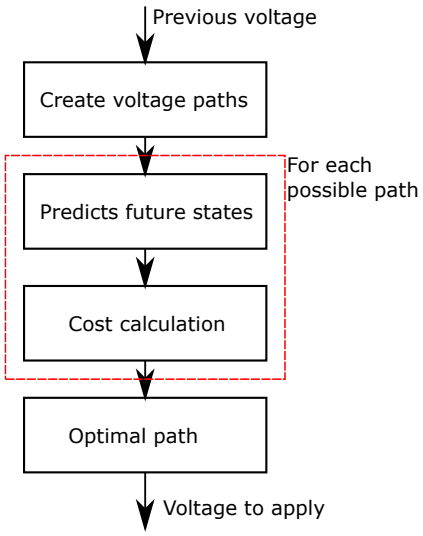

Figure 2.7: FS-MPC algorithm.

At every calculation step the list of possible voltage sequences, according to the last applied voltage, is fed to the prediction section of the algorithm.
The prediction involves the calculation of the current, speed and position in the following time instants, up to the chosen prediction horizon N. This calculation is carried out for all the possible voltages combinations[3]. Then the cost of every single analysed sequence is calculated. The formula to quantify the cost is called cost function and it is as follows:

$$J = \sum_{j=1}^{N} \lambda_x \cdot (x_{ref} - x_j)^2 + \lambda_v \cdot {v_j}^2 + \lambda_i \cdot {i_j}^2 \tag{2.10}$$

The λ coefficients weight the different contributes of the system states. These coefficients are constant in time, their values were chosen heuristically, since not much literature is available on this topic. N is again the prediction horizon, chosen equal to 3. Moreover a restraint on the current is applied:

$$J_j = \begin{cases} \lambda_x \cdot (x_{ref} - x_j)^2 + \lambda_v \cdot {v_j}^2 + \lambda_i \cdot {i_j}^2 & \text{if } -I_{max} \leq i_j \leq I_{max} \\ \infty & \text{if } i_j < -I_{max} \vee i_j > I_{max} \end{cases}$$

[3]The algorithm to calculate the possible voltage sequences was explained in section 2.3.1

The prediction equations are those already obtained in section 2.1.2:

$$i_{k+1} = (1 - \frac{R}{L}) \cdot i_k - K_e \cdot \frac{T_s}{L} \cdot v_k + \frac{T_s}{L} \cdot U_k$$

$$v_{k+1} = K_f \cdot \frac{T_s}{m} \cdot i_k + v_k - \frac{T_s}{m} \cdot F_{L_k}$$

$$x_{k+1} = x_k + T_s \cdot v_k$$

2.3.3 Integral Action in the FS-MPC

Since the goal is to track the position reference without any error at steady state, it is necessary to include an integral action in the system. In the standard control theory it is usually granted by the integral part of the PI, or equivalent, controllers. The literature about this topic in FS-MPC is not rich, but some interesting approaches have been tried [7]. In this thesis three different control structures are proposed:

- FS-MPC without any integral action. The cost function is as described in equation 2.10.

- Speed FS-MPC cascaded with a position PI controller.[4] The MPC cost function in this case is:

$$J = \sum_{j=1}^{N} \lambda_v \cdot (v_{ref} - v_j)^2 + \lambda_i \cdot {i_j}^2$$

- FS-MPC with a modification of the position reference according to:

$$X_{ref_{MPC}} = X_{ref} + K_p \cdot (X_{ref} - X_m) + K_i \cdot \int (X_{ref} - X_m)$$

  The cost function is similar to the one of the first case:

$$J = \sum_{j=1}^{N} \lambda_x \cdot (x_{ref_{MPC}} - x_j)^2 + \lambda_v \cdot {v_j}^2 + \lambda_i \cdot {i_j}^2$$

[4] This solution was proposed and verified in [8].

The choice of the coefficient K_p and K_i was made according to the simulation results.

The behaviour comparison between these structures will be presented in the Chapter 3, along with the simulation results.

2.4 Extended Kalman Filter

To reduce the overall sensor complexity of the system, and to reduce the final cost of the control, an observer is used. The observer allows to calculate the values of the system states. In this case the measured current i and position x are used. From these two measurements and the known value of the voltage U applied to the actuator is possible to observe the speed v of the moving iron in quite an accurate way. Moreover, in order to observe the disturbance load, an augmented model of the system is implemented, including the load force F_L as a state of the system.

The observer chosen is the Extended Kalman Filter (EKF)[5], which is a non linear extension of the Kalman Filter. Since the Kalman Filter can be only applied to linear system, a linearisation using a Taylor expansion around a nominal point has to be done. To linearise the system the Taylor expansion is truncated at the first order. In this case the point, around which the system is linearised, is the estimated operating point $\hat{\mathbf{x}}$. The linearisation for a multivariable system uses the Jacobian of the system's equations. It is possible to use a Kalman Filter like observer, since the process and measurement noise are supposed to be white, Gaussian distributed and with a zero mean value. Considering a non linear system described by these generic equations:

$$\mathbf{x}_{k+1} = \begin{bmatrix} x_{1_k+1} \\ x_{2_k+1} \\ ... \\ x_{n_k+1} \end{bmatrix} = \begin{bmatrix} f_1(x_{1_k}, x_{2_k}...x_{n_k}) \\ f_2(x_{1_k}, x_{2_k}...x_{n_k}) \\ ... \\ f_n(x_{1_k}, x_{2_k}...x_{n_k}) \end{bmatrix} + \begin{bmatrix} g_1(u_{1_k}, u_{2_k}...u_{n_k}) \\ g_2(u_{1_k}, u_{2_k}...u_{n_k}) \\ ... \\ g_n(u_{1_k}, u_{2_k}...u_{n_k}) \end{bmatrix} \tag{2.11}$$

$$\mathbf{Y}_k = \mathbf{H}_k \cdot \mathbf{x}_k \tag{2.12}$$

Where u_j represents the input of the system and $\mathbf{Y}$ the outputs of the system, the algorithm to implement an EKF is:

1. At every time step, compute the following derivative matrices:

[5] A theoretical description of this observer may be found in many books and texts. Practical references are [9] [10] [11]

- $\mathbf{F}_k = \mathbf{J}\begin{bmatrix} f_1(x_{1_k}, x_{2_k}...x_{n_k}) \\ f_2(x_{1_k}, x_{2_k}...x_{n_k}) \\ ... \\ f_n(x_{1_k}, x_{2_k}...x_{n_k}) \end{bmatrix} + \mathbf{J}\begin{bmatrix} g_1(u_{1_k}, u_{2_k}...u_{n_k}) \\ g_2(u_{1_k}, u_{2_k}...u_{n_k}) \\ ... \\ g_n(u_{1_k}, u_{2_k}...u_{n_k}) \end{bmatrix}$
- $\mathbf{C}_k = \mathbf{J}(\mathbf{H}_k)$

Where the derivatives are computed for $X_k = \hat{X}_k$, the observed states of the system.

2. Apply the Kalman filter equations:

   - $\mathbf{K}_k = \mathbf{P}_k \cdot {\mathbf{C}_k}^T \cdot (\mathbf{C}_k \cdot \mathbf{P}_k \cdot {\mathbf{C}_k}^T + \mathbf{R})^{-1}$
   - $\hat{\mathbf{x}}_{k+1} = \mathbf{f}(\hat{\mathbf{x}}_k, \mathbf{u}_k) + \mathbf{K}_k \cdot [\mathbf{Y}_k - \mathbf{H}(\hat{\mathbf{x}}_k)]$
   - $\mathbf{P}_{k+1} = \mathbf{F}_k \cdot (\mathbf{I} - \mathbf{K}_k \cdot \mathbf{C}_k) \cdot \mathbf{P}_k \cdot {\mathbf{F}_k}^T + \mathbf{Q}$

   Where $\mathbf{K}_k$ is the Kalman Gain matrix, $\mathbf{P}_k$ is the covariance matrix, $\mathbf{R}$ and $\mathbf{Q}$ are the measurement and process noise covariance matrices. $\mathbf{I}$ is the identity matrix.

This algorithm has been applied to the electromagnetic actuator model, augmenting by adding the force load as an extra state. In Figure 2.4 an overview of the algorithm is shown.

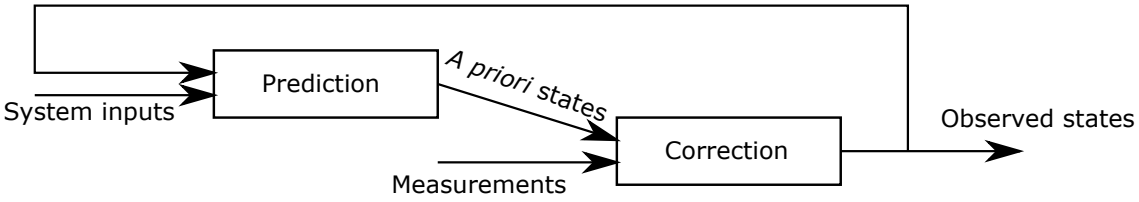

Figure 2.8: EKF algorithm.

2.4.1 EKF Applied to the Electromagnetic Actuator

As mention before, the system model described in equation 2.9 has to be augmented with an extra state. That allows to observe the load applied on the actuator, considered as a disturbance of the system. This method is known as Unknown Input

Observer (UIO) and it has been chosen because its simple to implement and lets both the states and the disturbance be observed at the same time.[6] The applied load value is supposed to be constant over a sample time, assuming its slow variation ($dF_L/dt = 0$). Under these hypothesis, the discrete system becomes:

$$\begin{bmatrix} i_{k+1} \\ v_{k+1} \\ x_{k+1} \\ F_{L_k+1} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} + \begin{bmatrix} g_1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} (1 - \frac{R}{L}) \cdot i_k - K_e \cdot \frac{T_s}{L} \cdot v_k \\ K_f \cdot \frac{T_s}{m} \cdot i_k + v_k - \frac{T_s}{m} \cdot F_{L_k} \\ x_k + T_s \cdot v_k \\ F_{L_k} \end{bmatrix} + \begin{bmatrix} \frac{T_s}{L} \cdot U_k \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.13}$$

Since the measured quantities are current and position, and the gain of the sensors used is 1, the output matrix $\mathbf{Y}$ is:

$$\mathbf{Y}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} i_k \\ v_k \\ x_k \\ F_{L_k} \end{bmatrix}$$

Substituting the equations in the EKF algorithm:

$$\mathbf{F}_k = \begin{bmatrix} \frac{\partial f_1}{\partial i} & \frac{\partial f_1}{\partial v} & \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial F_L} \\ \frac{\partial f_2}{\partial i} & \frac{\partial f_2}{\partial v} & \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial F_L} \\ \frac{\partial f_3}{\partial i} & \frac{\partial f_3}{\partial v} & \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial F_L} \\ \frac{\partial f_4}{\partial i} & \frac{\partial f_4}{\partial v} & \frac{\partial f_4}{\partial x} & \frac{\partial f_4}{\partial F_L} \end{bmatrix} =$$

$$\begin{bmatrix} 1 - \frac{R}{L} \cdot T_{se} & -\frac{Tse}{L} \cdot (k_q \cdot x^2 + k_s \cdot x + k) & \frac{(k_s + 2 \cdot k_q \cdot x) \cdot T_{se}}{L} \cdot v & 0 \\ \frac{Tse}{m} \cdot (k_q \cdot x^2 + k_s \cdot x + k) & 1 & \frac{T_{se} \cdot (k_s + 2 \cdot k_q \cdot x)}{m} \cdot i & -\frac{T_{se}}{m} \\ 0 & T_{se} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

[6] A brief reference about the theoretical aspects can be found in [12], two practical implementations in [13] and [14].

$$\mathbf{C}_k = \mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.15}$$

As it is possible to notice, the $\mathbf{F}_k$ matrix depends on the position of the moving iron. The position x_k has to be expressed in mm, with $x_0 = 0\,\text{mm}$ at the centre position. The $\mathbf{Q}$ and $\mathbf{R}$ matrices are diagonal, and their coefficients are chosen heuristically, since no information is available on the actual system and measurement noise. The starting values at $k = 0$ are $\mathbf{P}_0 = 0$ and $\mathbf{x}_0 = 0$. The coefficient T_{se} is the sample time of the filter, which was chosen $T_{se} = \frac{1}{100\,\text{kHz}}$[7].

2.4.2 Scaling of the System

Foreseeing the digital implementation of the system, it was chosen to use per unit quantities. This way the numbers used in the calculations are always between -1 and $+1$, allowing the choice of the same fixed point data type during the implementation. The details of the data type choice will be discussed in section 4.2, here the scaled system equations are presented.

To scale the system the following transformations have been applied:

$$\mathbf{S}_u = U_{max}; \qquad \mathbf{S}_x = \begin{bmatrix} I_{max} & 0 & 0 & 0 \\ 0 & v_{max} & 0 & 0 \\ 0 & 0 & x_{max} & 0 \\ 0 & 0 & 0 & F_{max} \end{bmatrix}$$

$$\mathbf{x}^s_{k+1} = \mathbf{S}_x^{-1} \cdot \mathbf{A} \cdot \mathbf{S}_x \cdot \mathbf{x}^s_k + \mathbf{S}_x^{-1} \cdot \mathbf{B} \cdot \mathbf{S}_u \cdot \mathbf{u}^s_k$$

$$\mathbf{u}^s_k = \mathbf{S}_u^{-1} \cdot \mathbf{u}_k = \frac{\mathbf{u}_k}{U_{max}}$$

The mark s indicates the scaled values. The following values were chosen as base values, according to the simulation process and datasheet specifications: $I_{max} =$

[7] This time step was decided according to the ADC data acquiring frequency. The ADC produces a filtered data stream at $1\,\frac{\text{MS}}{\text{s}}$, so the Kalman operating frequency has to be necessarily lower. During the tests this frequency was proven effective.

30 A, $v_{max} = 3\,\frac{\mathrm{m}}{\mathrm{s}}$, $x_{max} = 5\,\mathrm{mm}$, $F_{max} = 8.3 \cdot I_{max} \approx 240\,\mathrm{N}$, $U_{max} = V_{dc} = 48\,\mathrm{V}$.

$$\mathbf{A}^s = \begin{bmatrix} 1 - \frac{R}{L} \cdot T_{se} & -\frac{Tse}{L} \cdot \frac{v_{max}}{I_{max}} \cdot (k_q^s \cdot x^2 + k_s^s \cdot x + k) & 0 & 0 \\ (k_q^s \cdot x^2 + k_s^s \cdot x + k) \cdot \frac{Tse}{m} \cdot \frac{I_{max}}{v_{max}} & 1 & 0 & -\frac{Tse}{m} \cdot \frac{F_{max}}{v_{max}} \\ 0 & Tse \cdot \frac{v_{max}}{x_{max}} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

$$\mathbf{B}^s = \begin{bmatrix} \frac{Tse}{L} \cdot \frac{Umax}{I_{max}} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.17}$$

The value k_q^s is the coefficient of the back e.m.f. constant, scaled: $k_q^s = -0.3335 \cdot 10^6 \cdot x_{max}^2$, $k_s^s = -0.3652 \cdot 10^3 \cdot x_{max}$ and k does not change.

And therefore also the Jacobian is modified accordingly. $K_{e_{pu}} = k_q^s \cdot x_{pu}^2 + k_s^s \cdot x_{pu} + k$ for simplicity.

$$\mathbf{F}^s = \begin{bmatrix} 1 - \frac{R}{L} \cdot T_{se} & -\frac{Tse}{L} \cdot \frac{v_{max}}{I_{max}} \cdot K_{e_{pu}} & -2\frac{v_{max}}{I_{max}} \cdot \frac{Tse}{L} \cdot v_{pu} \cdot (kq_s \cdot x_{pu}) & 0 \\ \frac{I_{max}}{v_{max}} \cdot \frac{Tse}{m} \cdot K_{e_{pu}} & 1 & 2\frac{I_{max}}{v_{max}} \cdot \frac{T_{se}}{m} \cdot i_{pu} \cdot (kq_s \cdot x_{pu}) & -\frac{Tse}{m} \cdot \frac{F_{max}}{v_{max}} \\ 0 & Tse \cdot \frac{v_{max}}{x_{max}} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.18}$$

Chapter 3

Simulation Results

In this chapter the implemented control structure is presented and a comparison between different possible solutions is made. Both the FS-MPC and the EKF are implemented in Simulink. The EKF can be easily implemented using Simulink blocks, but in some cases, such as the Jacobian calculation, it is easier to implement some calculations using a MATLAB function. The diagrams of the EKF, which may be of interest, are available in appendix B.
Before presenting the results, some hypothesis have to be presented. In the following simulations a prediction horizon $N = 3$, as explained before, was chosen. White noise is assumed affecting the measurement and its variance is set to: 5 mA for the current and 9 µm for the position. These values were obtained analysing the output of the two sensors when measuring a constant value.

The three variants already described in paragraph 2.3.3 were simulated and compared:

1. FS-MPC without any PI controller.

2. Speed FS-MPC with a cascaded position PI controller.

3. FS-MPC providing integral action with a correction of the position reference.

The EKF parameters $\mathbf{Q}$ and $\mathbf{R}$ were tuned according to these criteria:

- **R** matrix, representing the measurement noise covariance has been chosen according to the added white noise. The two measurements are assumed independent, then the **R** matrix is diagonal.

$$\mathbf{R} = \begin{bmatrix} 5 \cdot 10^{-3} & 0 \\ 0 & 9 \cdot 10^{-6} \end{bmatrix}$$

- The **Q** matrix has been chosen as diagonal, which is a common choice in Kalman filter implementation. The values have been chosen heuristically, knowing that too large values lead not only to a faster convergence in the observation, but also to more oscillations and noise on the results.

$$\mathbf{Q} = \begin{bmatrix} 0.25 & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 \\ 0 & 0 & 10^{-8} & 0 \\ 0 & 0 & 0 & 2.5 \cdot 10^{-7} \end{bmatrix}$$

In the following subsections these solutions are presented and analysed.

3.1 FS-MPC Without PI

FS-MPC without PI is the simplest solution possible, but the main disadvantage of this structure is the absence of integral action. This leads to an error in the position reference tracking at steady state if any load is applied to the system.
The cost function weighting factors are shown in Table 3.1.

Table 3.1: FS-MPC with no integral action structure: weighting factors.

Factor	Value
λ_x	$60 \cdot 10^6$
λ_v	0.7
λ_i	$1 \cdot 10^{-6}$

As it is possible to see in the following Figures the system behaviour is satisfactory under no load conditions (Figure 3.1), but it worsen if a load is applied.

The EKF prove itself effective, providing quite accurate speed and load observation. The load observation has a much bigger than the speed observation error (Figure 3.2) due to the simplified model of the disturbance observer, but is on average correct.

In a subsequent simulation, the system was simulated with a constant load $F = 60\,\mathrm{N}$ applied on the actuator. As it can be seen in Figure 3.3 the position is not reached without error fast enough. The EKF proves effective and the observation errors, after a first settling phase, rapidly go to the same values as the case with no load applied.

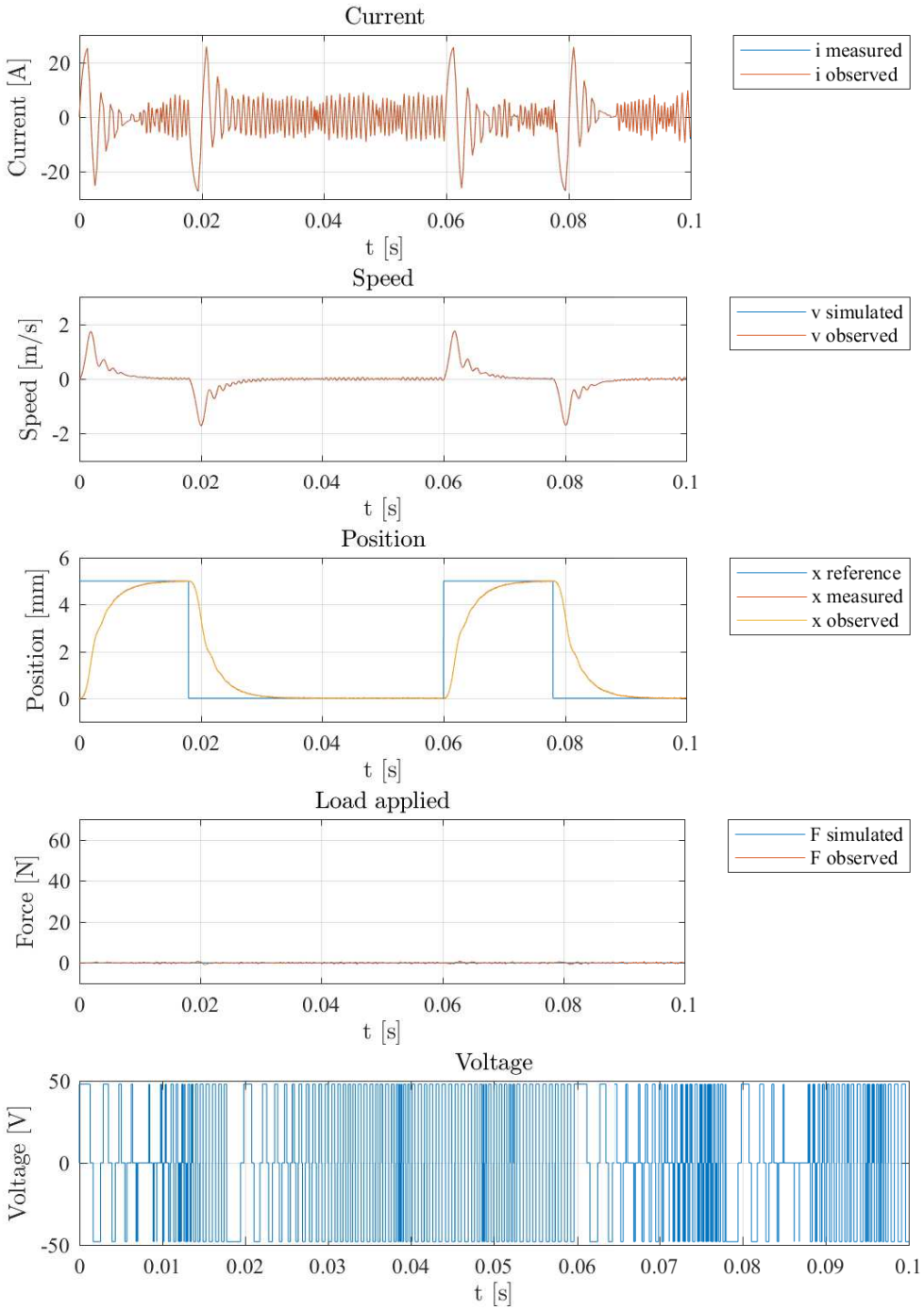

Figure 3.1: FS-MPC simulation results, without PI. No load is applied.

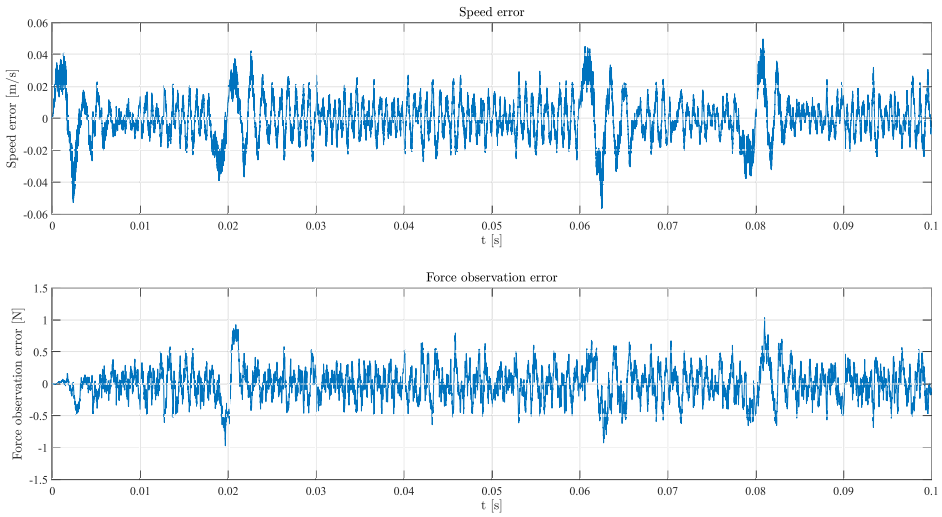

Figure 3.2: EKF errors of speed and load observation. No load is applied.

Finally a spring load ($K_{spring} = 1.4310\,\mathrm{N/mm}$) was simulated. The small load generated by the spring does not affect significantly the position tracking. More interesting is the load observation, which shows its limits, linked to the assumption of a constant load (over a time step): as the load moves to a more variable profile, the observation quality decreases. The speed observation though maintain a good quality.

It is interesting to compare the three load situation:

- the position response worsen depending on the magnitude of the load applied.
- the speed observation does not change significantly, apart from the first instants, where the EKF has to adapt to the sudden change of the system. At steady state the error is comparable in all three situations.
- The force observation is strongly dependent on the load profile. If the load varies in time, the observer is not fast enough to adapt to it. The behaviour may be different and improved with different disturbance observer equations (for example inserting a model of the load, if known).

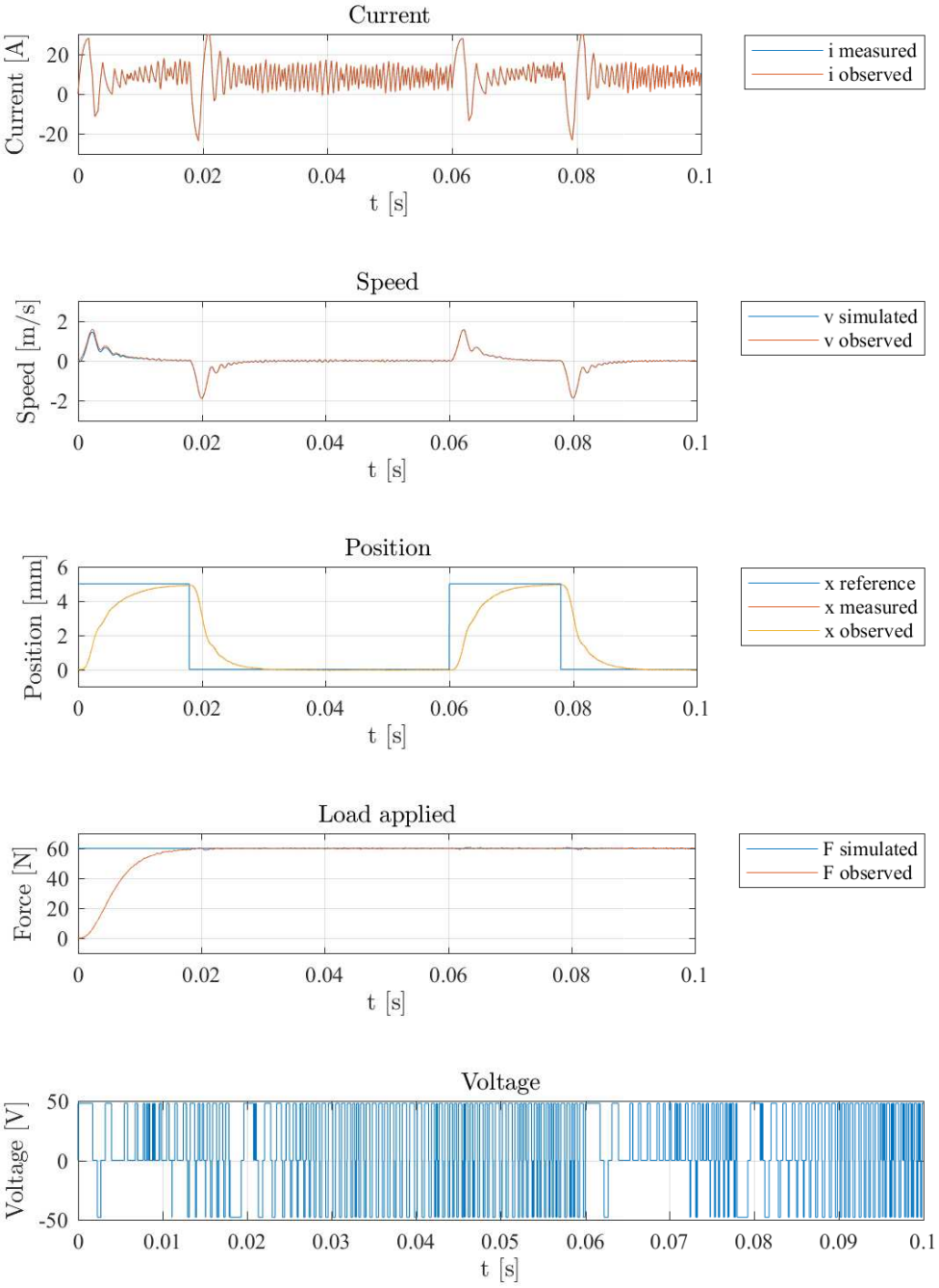

Figure 3.3: FS-MPC simulation results, without PI. Constant load $F = 60\,\mathrm{N}$ is applied.

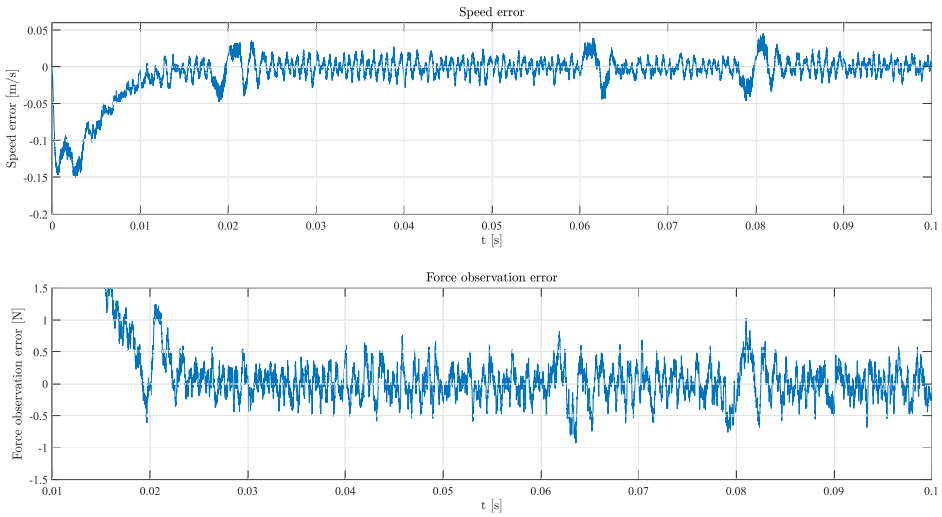

Figure 3.4: EKF errors of speed and load observation. Constant load $F = 60\,\mathrm{N}$ is applied.

In this section the effectiveness of the EKF was shown. The specific results for the speed and load observation errors are omitted when discussing the other control strategies, since they would be identical.

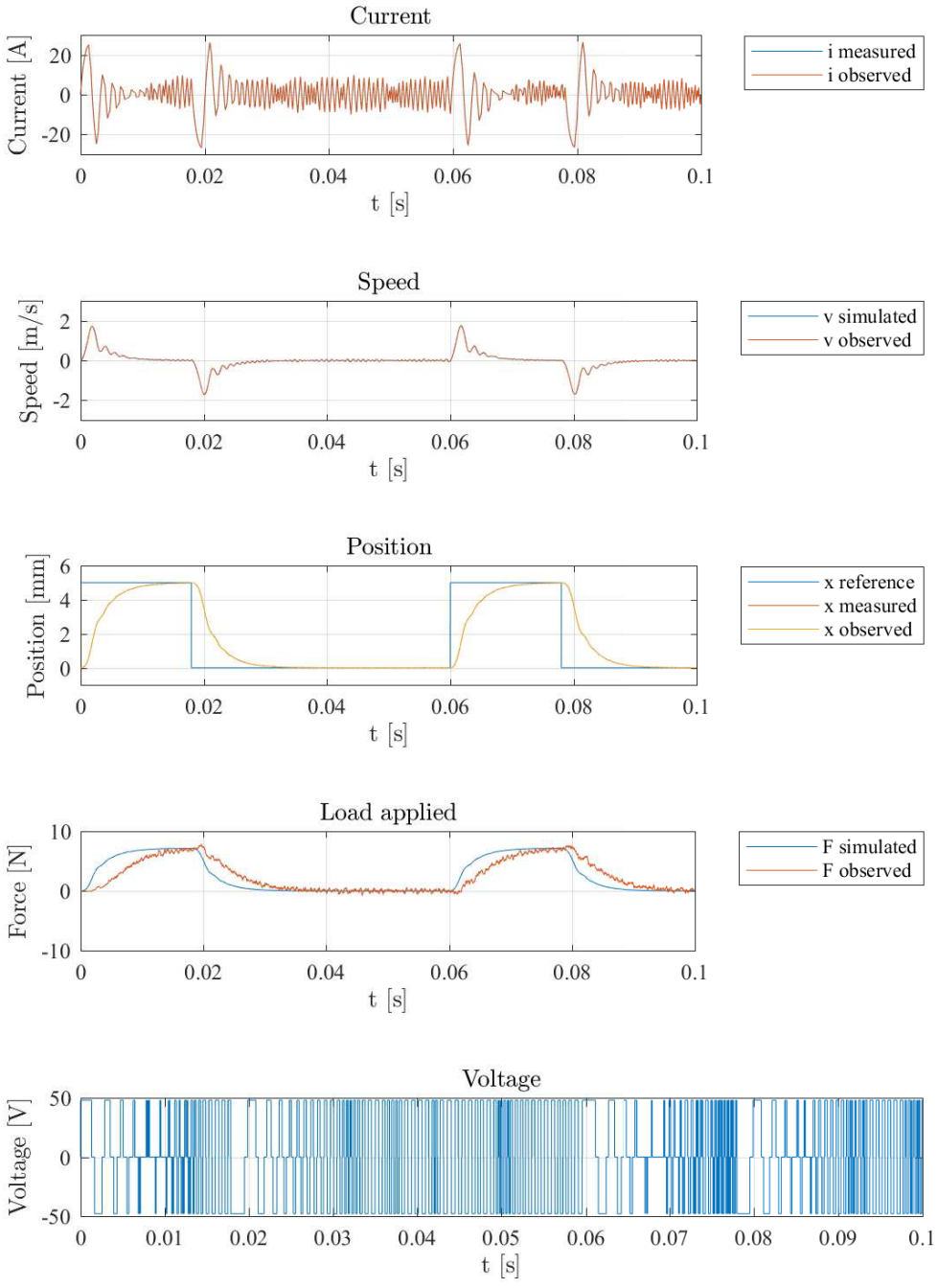

Figure 3.5: FS-MPC simulation results, without PI. Spring load $K_{spring} = 1.4310\,\mathrm{N/mm}$ is applied.

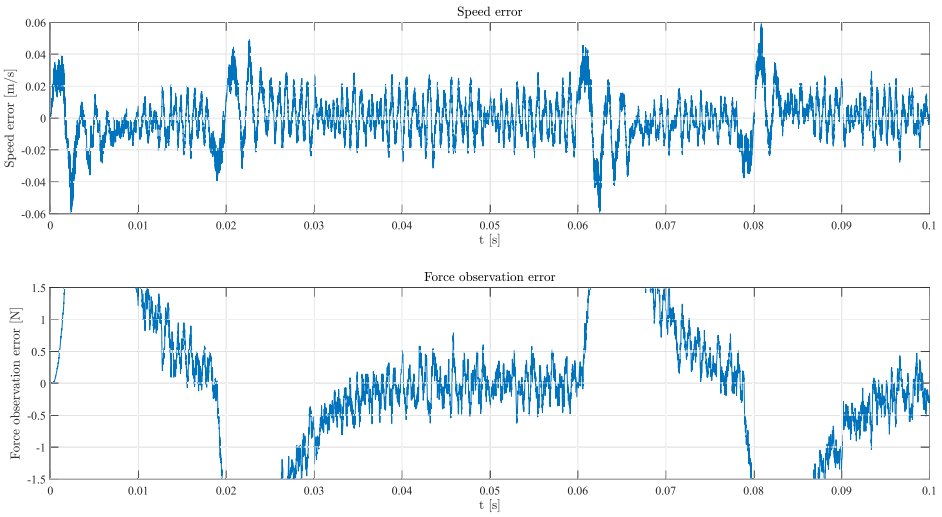

Figure 3.6: EKF errors of speed and load observation. Spring load $K_{spring} = 1.4310\,\mathrm{N/mm}$ is applied.

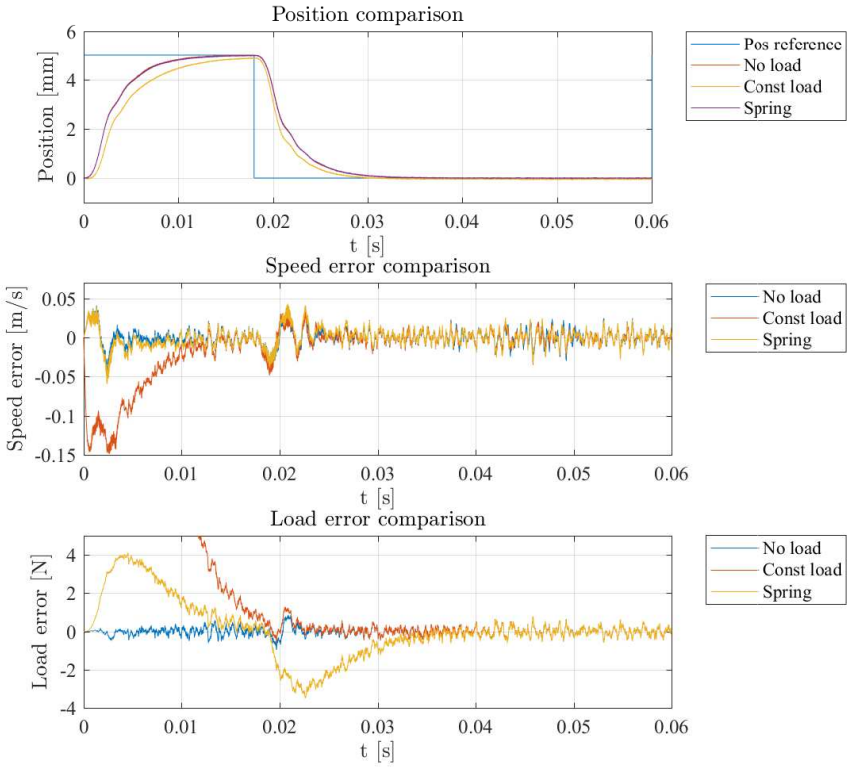

Figure 3.7: FS-MPC without integral action. Comparison.

3.2 FS-MPC with Position PI

The speed and current FS-MPC with a cascaded position PI tries to mix the advantages of the PI integral action with the MPC structure.
The first advantage is the behaviour under load, in fact the error at steady state is erased, thanks to the integral action provided by the position PI controller. A second advantage is the simplified MPC algorithm, and therefore the smaller computational burden. In Table 3.2 the used factors and coefficients are presented. The

Table 3.2: FS-MPC with position PI: factors values.

Factor	**Value**
K_p	500
K_i	2000
λ_v	500
λ_i	$1 \cdot 10^{-3}$

simulations were run in the same load configuration as the past structure.

As it can be seen in the comparison Figure 3.11, the under load behaviour greatly improved, reducing the steady state offset to almost zero. The speed of the controller is almost unchanged, maintaining the same settling time.

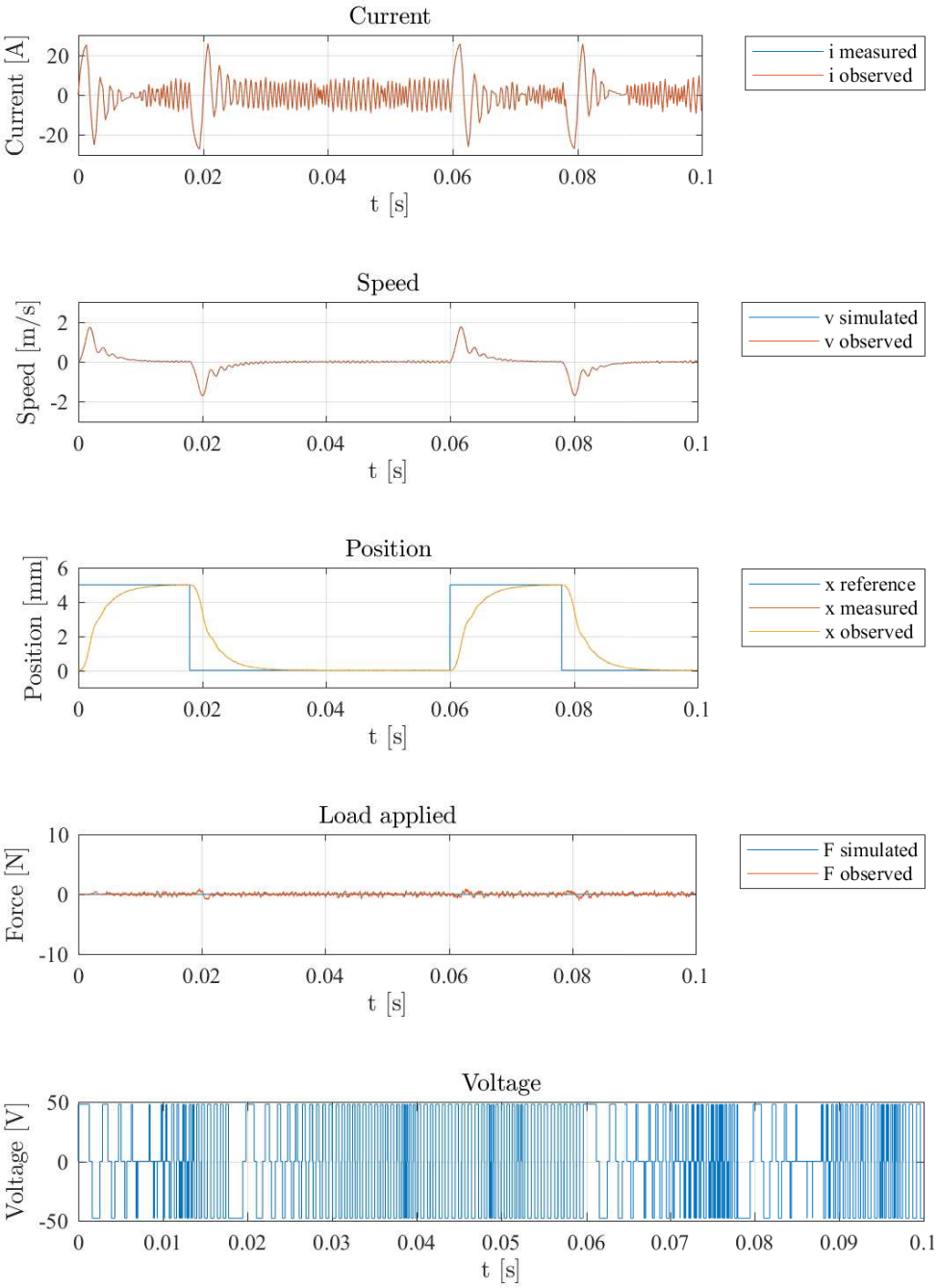

Figure 3.8: FS-MPC simulation results, with position PI. No load is applied.

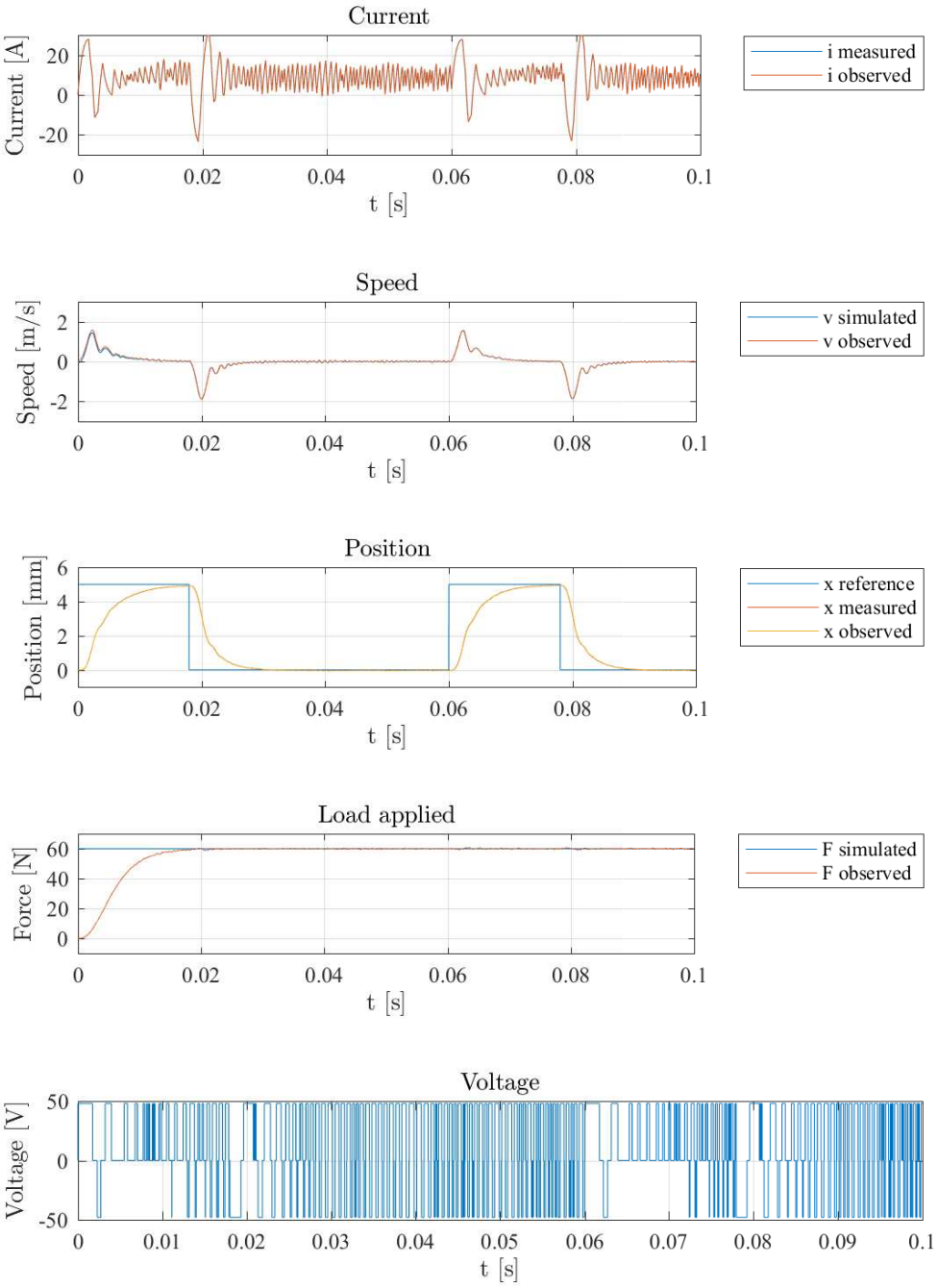

Figure 3.9: FS-MPC simulation results, with position PI. Constant load $F = 60\,\mathrm{N}$ is applied.

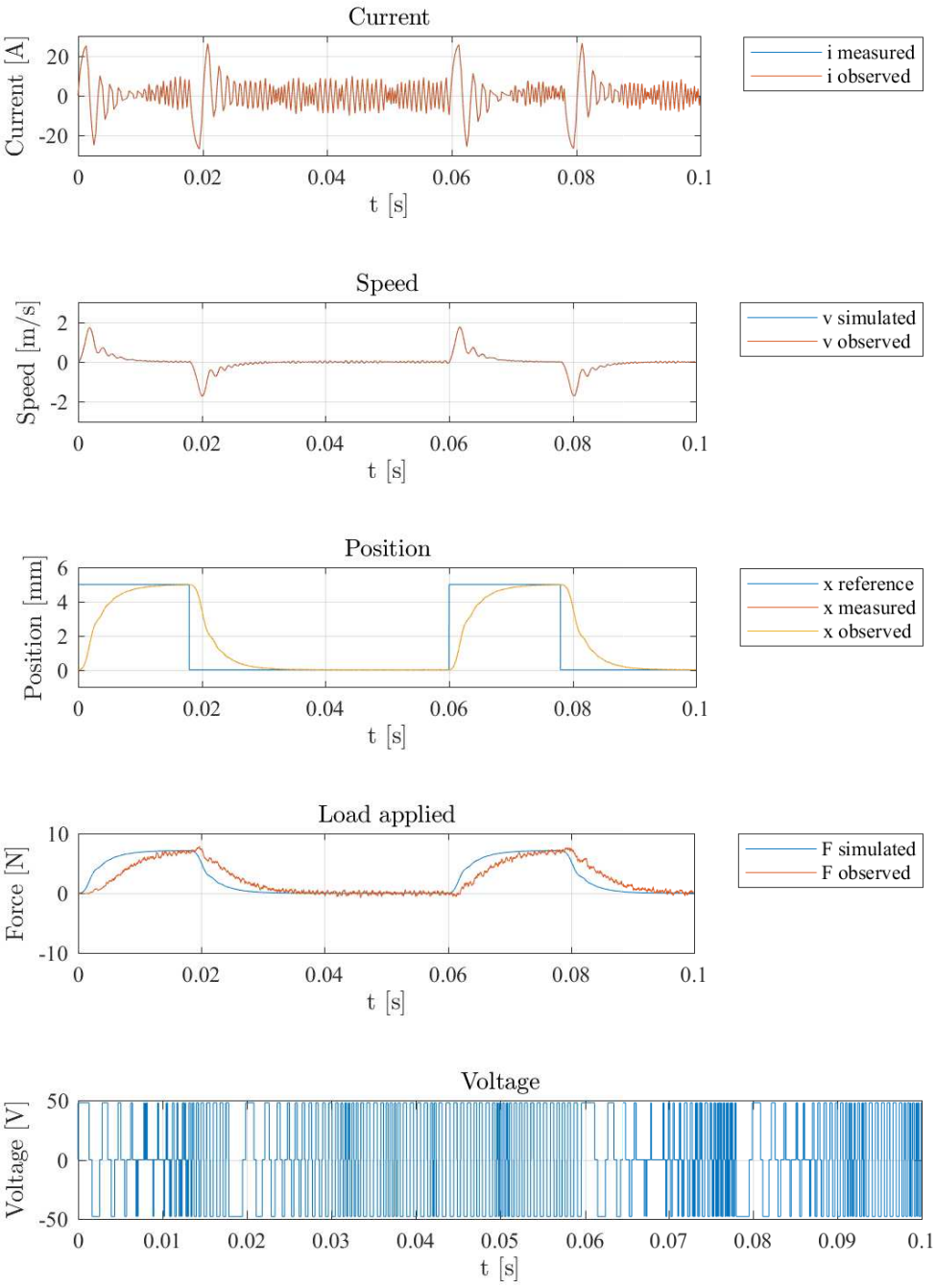

Figure 3.10: FS-MPC simulation results, with position PI. Spring load $K_{spring} = 1.4310\,\mathrm{N/mm}$ is applied.

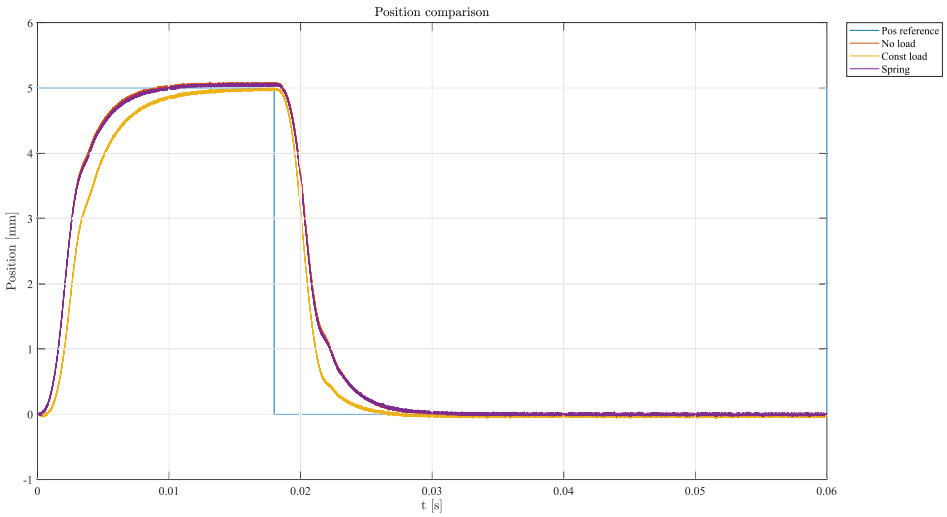

Figure 3.11: FS-MPC without position PI: Position response comparison.

3.3 FS-MPC with Position Reference Modification

This last proposed structure was already described in section 2.3.3. Its functioning principle is rather simple (works in a similar way as a PI controller), and has good position tracking performance and it is robust in terms of quick response and disturbance rejection. The correction parameters were chosen to guarantee no overshoot and the fastest settling time of the system. The settling time is acceptable and shorter than the previously presented structures, around 10 ms, and similar to other control structures described in literature [15] [16] [17].

Table 3.3: FS-MPC with position reference modification: Factors values.

Factor	Value
K_p	0.7
K_i	5
λ_x	$60 \cdot 10^6$
λ_v	0.8
λ_i	$1 \cdot 10^{-6}$

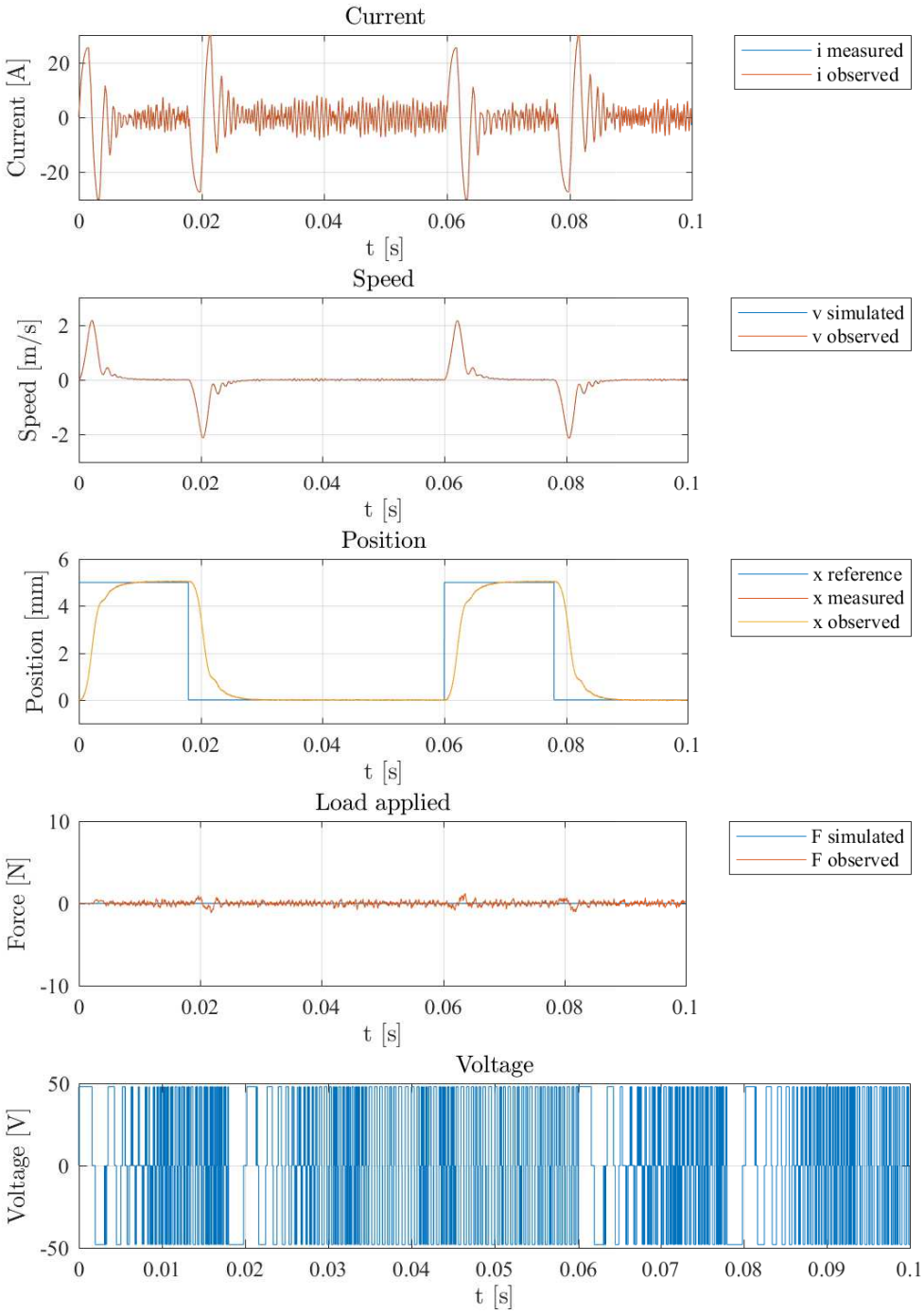

Figure 3.12: FS-MPC simulation results, with position reference modification. No load is applied.

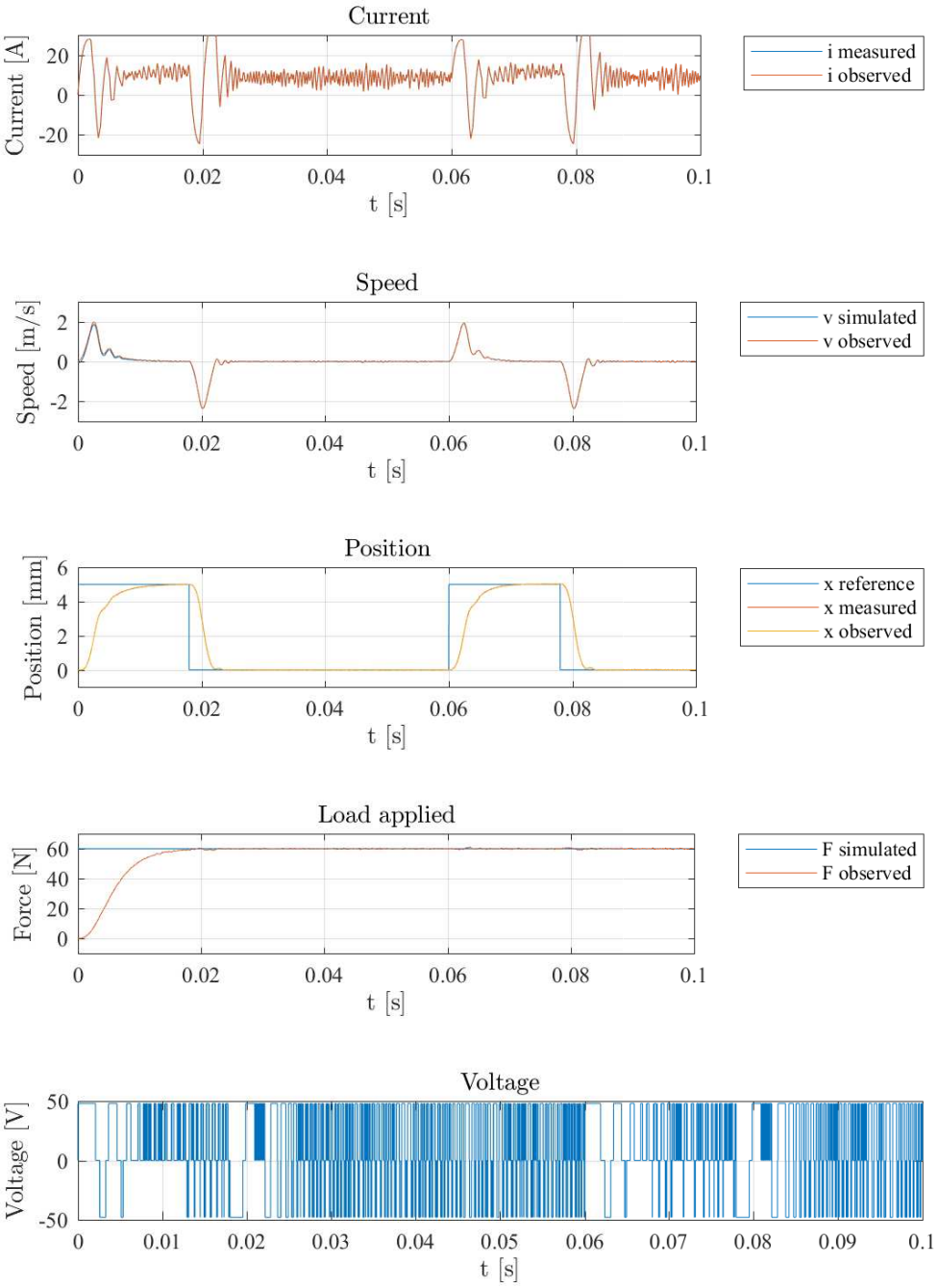

Figure 3.13: FS-MPC simulation results, with position reference modification. Constant load $F = 60\,\mathrm{N}$ is applied.

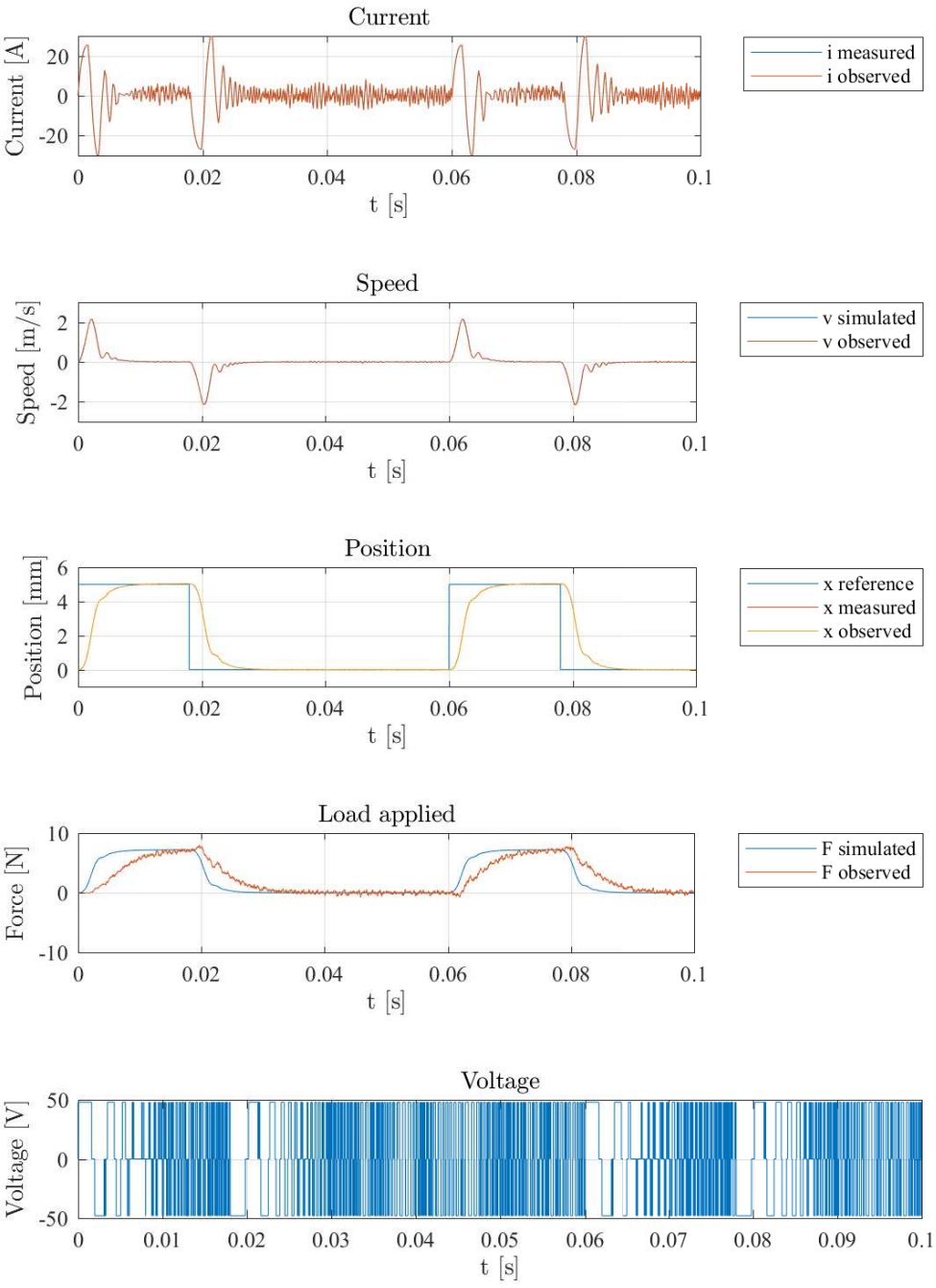

Figure 3.14: FS-MPC simulation results, with position reference modification. Spring load $K_{spring} = 1.4310\,\mathrm{N/mm}$ is applied.

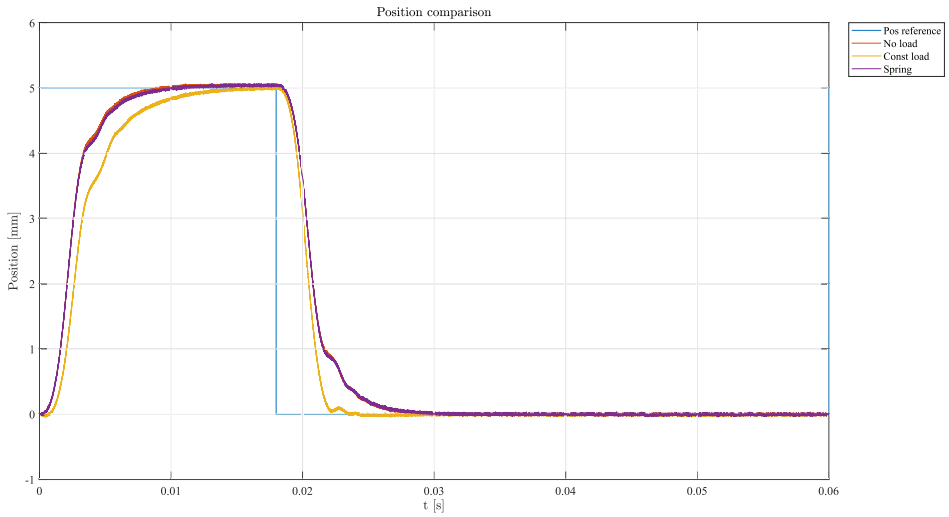

Figure 3.15: FS-MPC with position reference modification: Position response comparison.

3.4 Performance Comparison

After having presented thoroughly the three structures, it is possible to compare the position response in the three described load situations.

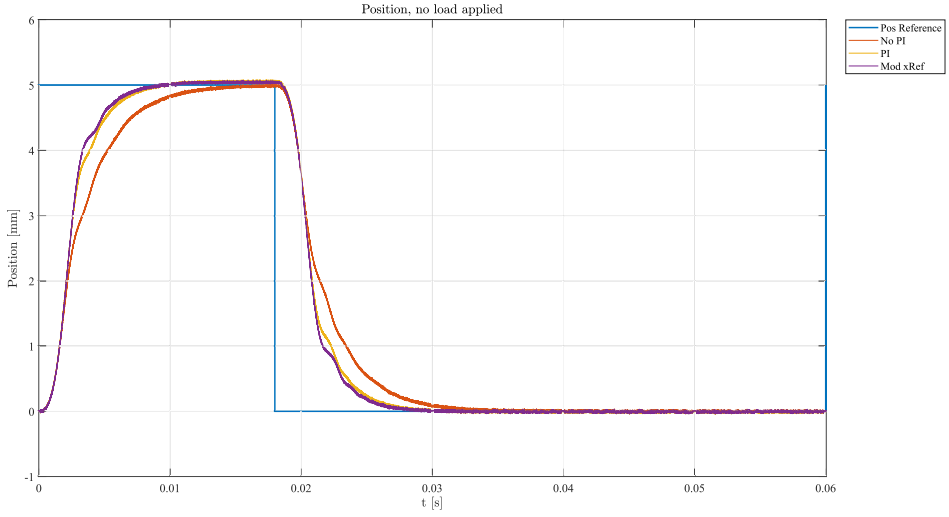

Figure 3.16: FS-MPC position response comparison. No load is applied.

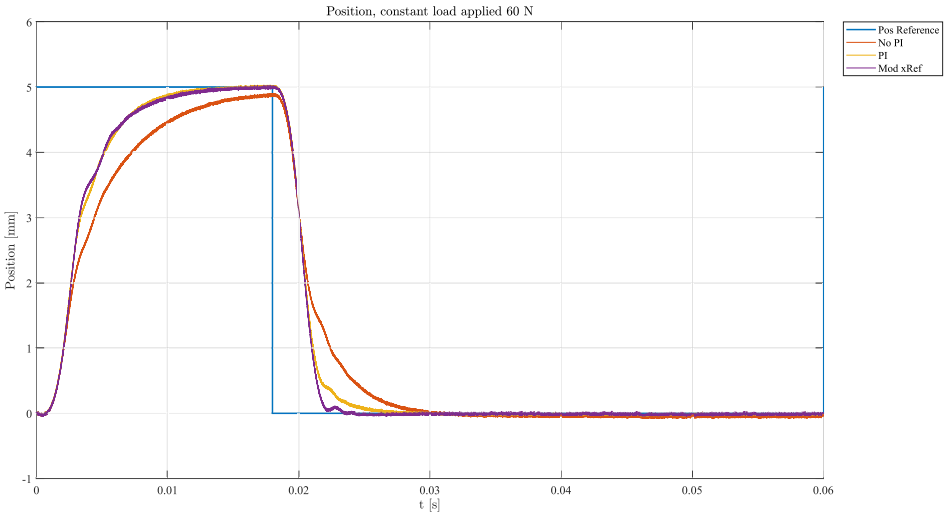

Figure 3.17: FS-MPC position response comparison. Constant load $F = 60\,\mathrm{N}$ is applied.

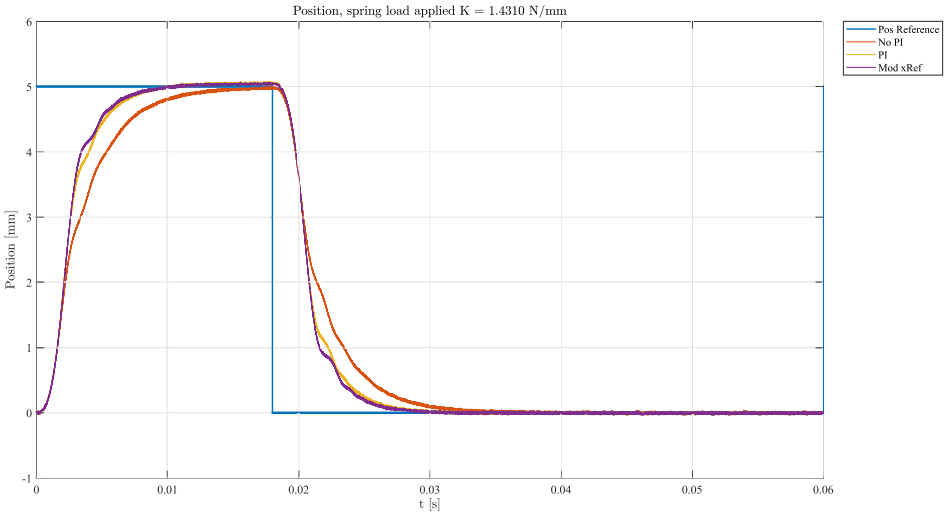

Figure 3.18: FS-MPC position response comparison. Spring load $K_{spring} = 1.4310\,\mathrm{N/mm}$ is applied.

As it can be clearly seen in Figures 3.16, 3.17 and 3.18, the FS-MPC without any integral action is the worst system. Its settling time is barely sufficient to track the position reference and it does not even reach it in the under load case. The system with position PI and position reference modification are comparable, both under the settling time and the steady state error point of view. The only relevant difference can be seen in the under load case, Figure 3.17, around $t = 0.02\,\mathrm{s}$, when the position reference goes back to 0. In this case the MPC with the position PI is slightly slower than the other one.

Summing up, it can be stated that the best structure is the FS-MPC with the modification of the position reference, which provides short settling times and good disturbance rejection, even though the cost function calculation is more resource demanding.

Chapter 4

Implementation

The central processing unit of the control system is a FPGA Altera Cyclone-III EP3C40Q240C8. [18] Both the FS-MPC and the EKF are implemented on this board. The VHDL code of the control system was generated using the HDL Coder tool available in Simulink. This tool is capable of converting a block diagram into VHDL code directly. In order to make it work some conditions have to be respected:

- The diagram has to be in discrete time, with a fixed time step.
- No matrices or calculations using matrices are allowed.
- Only fixed point or boolean data types can be used in the diagram.

The first step was already done, since the controller was designed as a digital system, dealing with the discrete model of the actuator. In the following sections the other steps of the conversion process are analysed in detail.

4.1 Elimination of Matrix Operations

The first task was to avoid the usage of matrices inside the design.
While the FS-MPC does not need to use matrices, since all the calculations can be carried out in separate equations, the Kalman Filter presented some problems, since the variance matrix $\mathbf{P}$ and the gain matrix $\mathbf{K}$ have to be calculated multiplying matrices at every iteration. This problem has two possible solutions:

1. Transform the matrices multiplications into scalar operations, obtaining long and complicated equations

2. Try to eliminate the variance matrix update and use a constant gain matrix

The second solution has the great advantage of reducing both the complexity of the system and the computational effort, moving the gain calculation to the simulation phase. On the other hand, it can lead to a loss of performances.
In order to choose the best solution, the values of the gain matrix were analysed during the simulation. In Figure 4.1 it is possible to see how the eight coefficients vary in time. This analysis was carried out under the simulation parameters presented in Chapter 3, which can be assumed as valid under the normal operation conditions.

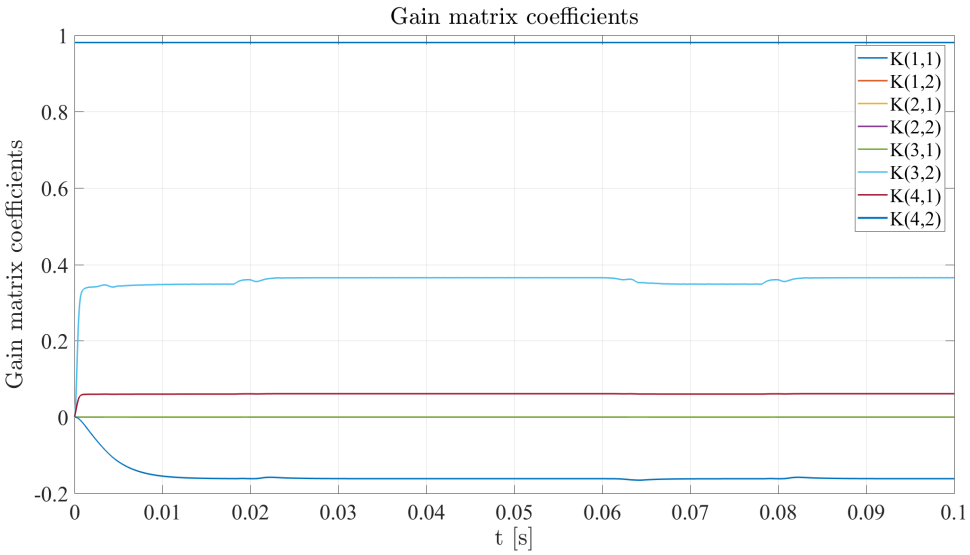

Figure 4.1: Kalman gain matrix coefficients

As it is clear from the picture, the coefficients rapidly converge to a fixed value, that can be chosen as constant and implemented in the EKF scheme. This way the linearisation of the system and the calculation of the **P** and **K** matrices is avoided. The prediction phase is still using the non linear model. Moreover, many coefficients are much smaller than the others, and can therefore be approximated as

0. The chosen values for the **K** matrix are:

$$\mathbf{K} = \begin{bmatrix} 0.9808 & 0 \\ 0 & 0.3654 \\ 0 & 0.06132 \\ 0 & -0.1615 \end{bmatrix} \tag{4.1}$$

The simulation performances obtained with this simplification are identical to the ones obtained with the full calculation and confirm the validity of the solution.

4.2 Conversion Using Fixed Point Tool

The second step in the preparation for VHDL code generation is the choice of fixed point data types for every value in the system. This process is necessary, since FPGAs don't support floating point data values.
The word length was chosen according to:

- The valued measured from the sensors are converted to 28 bits values.
- A pre-existing control used 32 bits values.
- Resources available on the FPGA.

4.2.1 EKF Conversion

For the EKF a default word length of 32 bits was chosen. Then, using the Fixed Point Tool the data types were chosen and applied to the single blocks of the system. After this process the system was simulated again to be sure of the functionality and then it was ready to be processed by the HDL Coder tool.

4.2.2 FS-MPC conversion

The conversion of the FS-MPC was a longer process, because a compromise between resource utilization on the FPGA and accuracy of the results had to be found. After some tests, it became clear that it was impossible to fit the 3 steps prediction system,

because of the limited resources available. Therefore a simpler system, with only one step prediction ($N = 1$) was implemented. In order to make this system fit into the board, a 26 bits word length was chosen after an iterative process.

4.3 FPGA in the Loop

Before implementing the code on the real hardware, an intermediate simulation step was done. Simulink offers the possibility to interface the FPGA with the model in the so called "FPGA in the loop" using the HDL Verifier tool. This way the control algorithm runs on a real FPGA board, while the physical system is simulated inside Simulink, as it is possible to see in Figure 4.2.

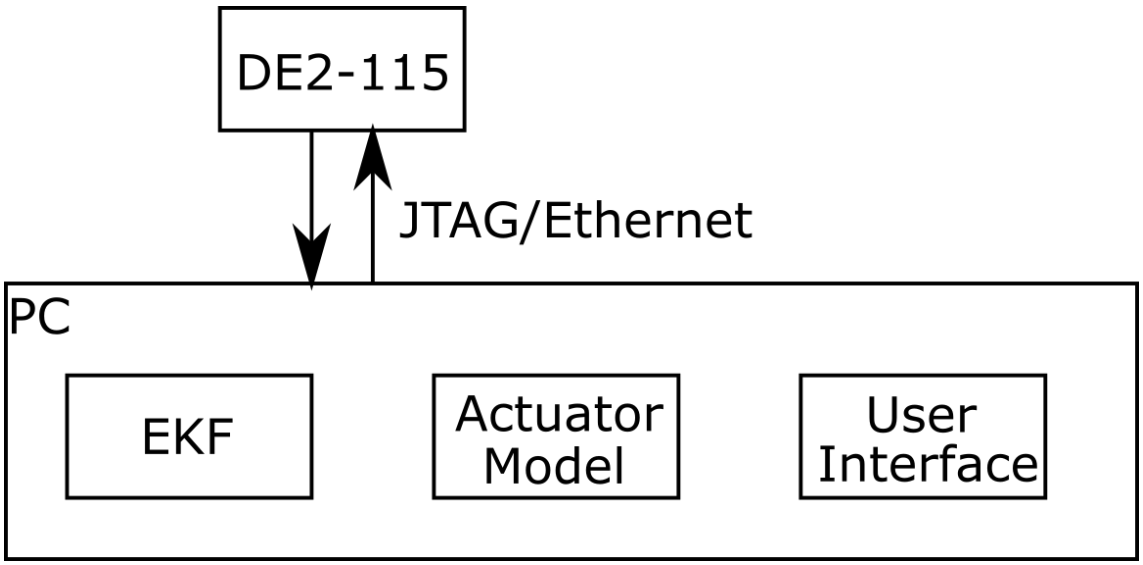

Figure 4.2: FPGA in the loop principle.

Only some FPGA boards are supported by Simulink, and in this case the DE2-115 development kit was used. This board comes with an Altera Cyclone IV FPGA, along with some peripherals to interface it (e.g. Ehernet and USB connections).
To successfully set up the system these steps were followed:

1. The first step was to configure the connection between the board and Simulink, which was established following the wizard available in Simulink itself. Both Ethernet and JTAG communications are available and equivalent.

2. Then it was necessary to provide the VHDL code previously generated to the HDL Verifier, in order to allow the creation of the program file to upload on the FPGA.

3. After the compilation process, a new Simulink block, representing the FPGA is available and can replace the control block.

Running the software on the DE2-115 some timing issues, related to the propagations delays inside the physical board, which aren't considered in the Simulink blocks by default were noticed. The consequence of those delays is the de-synchronisation between signals and thus the presence of random spikes on the signals. In Figure 4.3 the nature of those disturbances on the cost function values is shown. This issue also lead to wrong numerical values of the voltage reference sent to the H-bridge. Adding some "Memory" blocks before the cost function calculation circuit solved the problem and the software running on the FPGA behaved exactly as the simulated control algorithm.

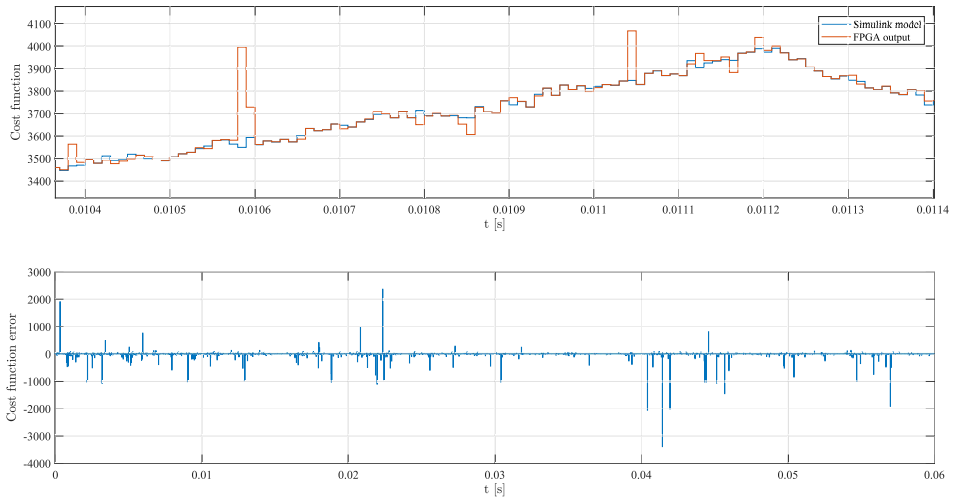

Figure 4.3: FPGA in the loop: errors due to bad timing.

4.4 Final Implementation

Last step in the implementation process was to interface the control and EKF code with the pre existing VHDL code for the Cyclone III board. The communication

between the FPGA and both an external processor, the power elements and the user interface were already available. [6]

The code generated by Simulink was then added into the existing Quartus II[1] project and added to the main program entity.

Since the board is a FPGA it is impossible to use the conventional debugging techniques, but instead it is necessary to deal with the low level signals inside the board. Quartus II also provide the "Signal-Tap" tool to record the values of some signals and then send them to the PC. This tool provide the logic state of every bit (in case of fixed point numbers) and also includes a converter to read directly the numeric value of the recorded signal.

The graphs were then drawn using MATLAB, after having exported a .csv file with the recorded values. It has to be noted that the recorded values must be then divided by the bit length of the fractional part of the number analysed.[2]

The signal tap is also a practical way to acquire the position and current data from the sensors when the control is running without using additional measurement or logging tools.

It is of interest to compare the resource usage of the different solutions, as shown in Table 4.1.

Table 4.1: FS-MPC with position reference modification: Factors values.

Resource type	**Position PI**	**Position ref. correction**
Total logic elements	23,822/39,600 (60%)	34,956/39,600 (88%)
Total combinational functions	23,307/39,600 (59 %)	34,434/39,600 (87%)
Dedicated logic registers	2,825/39,600 (7%)	2,916/39,600 (7%)
Total registers	2841	2932
Total memory bits	8,192/1,161,216 (1%)	8,192/1,161,216 (1%)
Embedded Multiplier 9-bit elements	252/252 (100%)	252/252 (100%)
Total PLLs	2/4 (50%)	2/4 (50%)

[1]This is the IDE provided by Intel/Altera to develop the code for their boards. In includes an editor a synthesiser, a compiler and an optional FPGA simulator, ModelSim.

[2]For example: a signal coded as 16.8, so with a fraction length of 8 bits, is exported as two complement's integer value. This value has to be divided by 2^8 to get the decimal representation of this signal.

The usage is presented in both absolute value and relative to the total resources available on the Cyclone III. As it can be seen, the solution with the position reference correction needs more resources, due to the more complex cost function and prediction part.

4.5 Early Testing

Before trying the complete control system on the board, some tests were carried out with the EKF and a simple cascaded PI controller for position and speed, without applying any load. The measurements were taken with a software included in Quartus II: the SignalTap II Logic Analyzer. [19] Since the FPGAs are similar to electric circuit, no real time debugging tool, like those for microcontrollers, are available. The only possible solution is analysing and logging the logical values of some signals in the board. Then these values can be converted from the binary codes into the corresponding numerical value and eventually plotted using MATLAB. This technique has a limit, which depends on the built-in memory bits available. These bits are used to store the logged data and are in limited number, so not all the quantities can be recorded at the same time.
The quality of the position observation is quite high, and as it's possible to see in Figure 4.4, the error is negligible.

The current observation is not as good, since a lot of disturbance is present in the observation. The control uses the measured values, so it is not affected.

Both speed and load observation can't be compared with the actual physical values, since no sensor was available. From a qualitative point of view, the values make sense and are comparable with those predicted in the simulations.

Unfortunately, after these first tests, the H-Bridge broke due to a shorted component and it was not possible to use it any more, so no further experimental tests could be carried out.

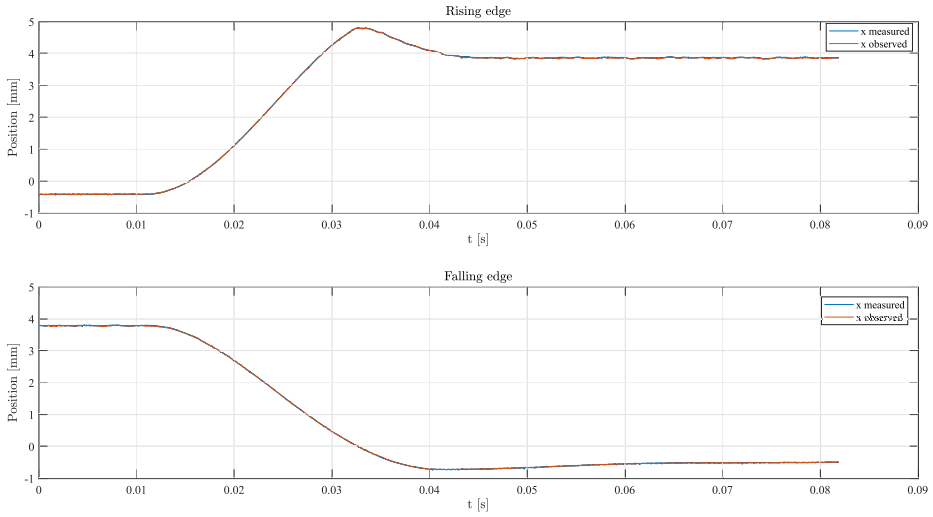

Figure 4.4: EKF measurements: position.

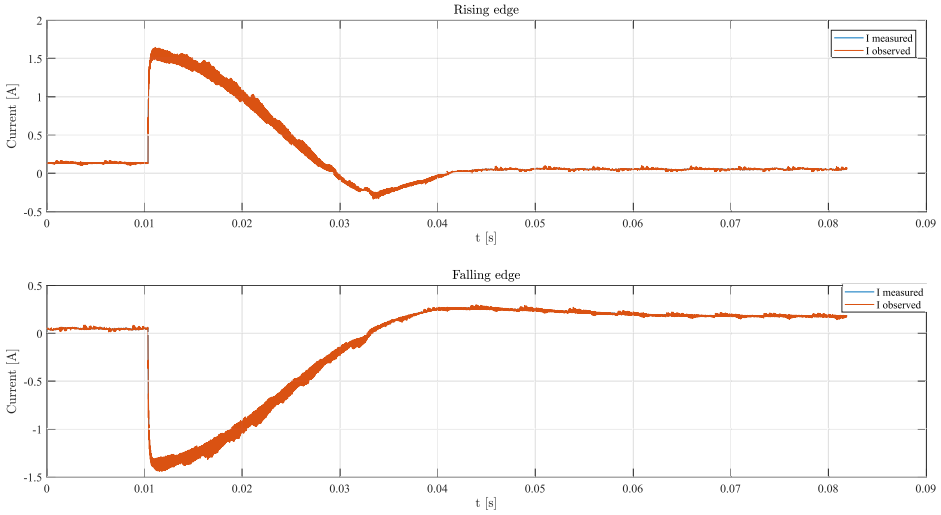

Figure 4.5: EKF measurements: current.

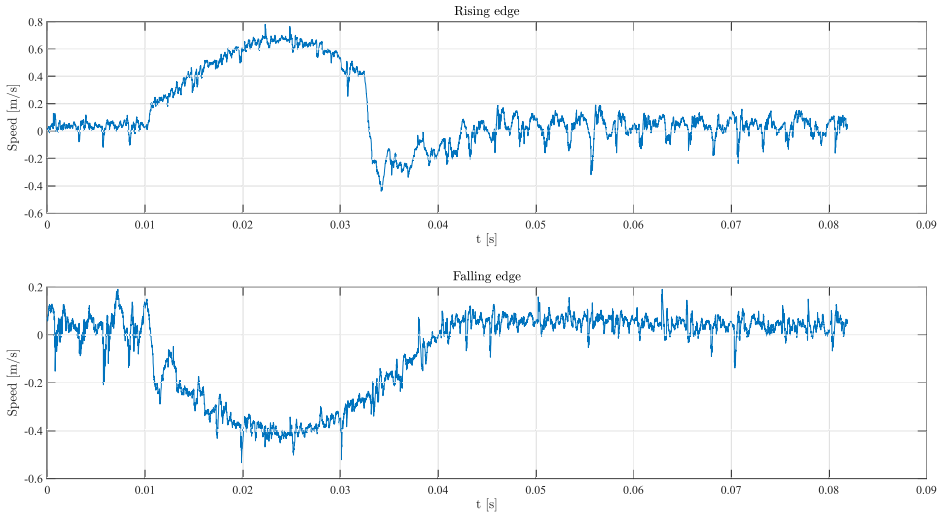

Figure 4.6: EKF measurements: speed.

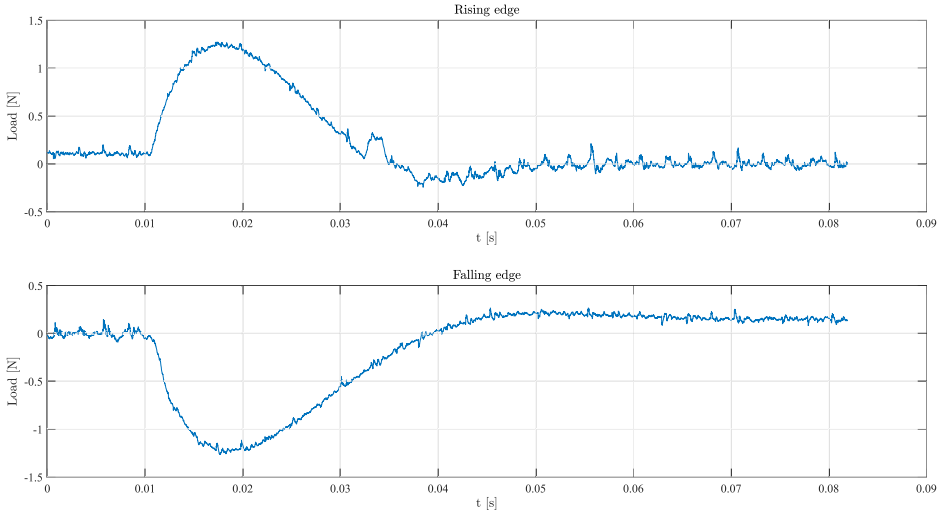

Figure 4.7: EKF measurements: load observation.

4.6 Intel Processor

A new H-Bridge, using the module PM100RL1A060 [20], was found and put into operation using a control board EVBL1S1 [21]. These switches cannot be operated at a switching frequency greater that 20 kHz, so it was decided to run the PWM signal at 10 kHz and run the algorithm at twice this frequency. At this point a FPGA is not needed any more and a sequential processor programmed in C was used.

The processor used is a 64-bits Intel Pentium 4 CPU running at 3.40 GHz, mounted on a PC-104 board and connected with an ISA socket to the FPGA sending the optical signals to the board. The complete setup is shown in Figure 4.8.

The processor runs a Linux Ubuntu operative system and uses the RTAI library to operate as a real time system.

The implementation in C loses the benefits of parallel calculation, that are present on the FPGA, but, on the other hand, the coding and the debugging are much easier.

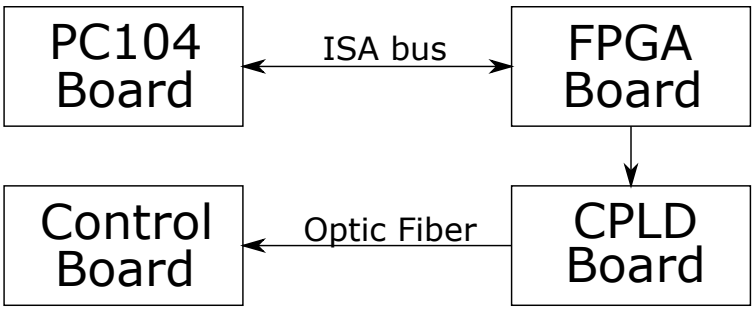

Figure 4.8: Setup of the processor board.

4.6.1 Code stucture

The code is divided into two main modules, as shown in Figure 4.9: one real time and one running inside Linux.

The real time module executes both the control and the EKF function when triggered by an interrupt signal coming from the FPGA. The Linux module takes care of logging the measured (or calculated) data into a csv file, using the shared memory

functions provided by the RTAI library. The csv file can be later imported in MATLAB (or any other custom software) and the results analysed.

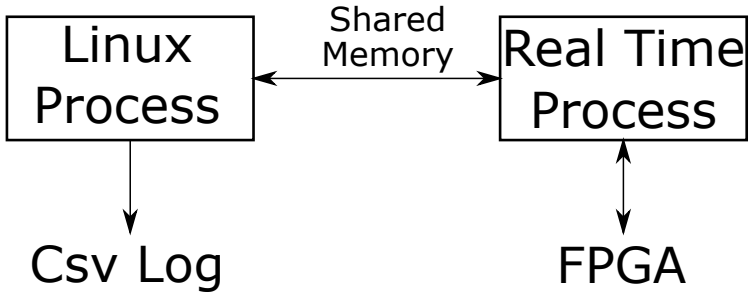

Figure 4.9: Processor code structure.

The significant sections of the RTAI and Linux code are available in appendix B.

4.6.2 Experimental Results

After the implementation several tests were carried out, with different tuning and structures. In this section, the behaviour of all the three control structures is presented and the differences highlighted.

4.6.2.1 FS-MPC Without PI

The first test was carried out with the base FS-MPC structure, without any integral action provided. The cost function weighting factor were tuned after an iterative process of trial and error. Both tests with a prediction horizon of one and three were carried out.
First the one step prediction horizon algorithm was tested. The optimal weighting factors are shown in Table 4.2.

Table 4.2: FS-MPC with no integral action structure: weighting factors. Experimental case.

Factor	Value
λ_x	$1 \cdot 10^6$
λ_v	1
λ_i	0

As it can be seen in Figure 4.10, the position target is reached quickly, but overshoot is present both in rising and falling position transitions.

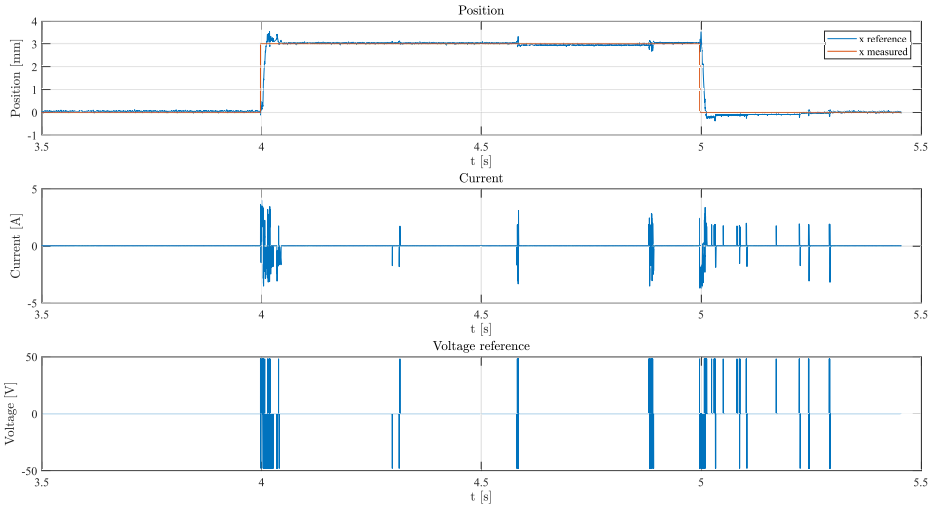

Figure 4.10: Experimental results: FS-MPC, one step prediction horizon.

As for the weighting factors choice, the weighting factor for the current was set to 0, otherwise the system would recover slower from the overshoot; the other two are lower than those obtained in the simulations, because the disturbances on the actual measurements were higher than expected. A bigger factor would lead to more oscillations, as the controller would try to compensate to much any small deviation from the reference.

Then the three steps prediction horizon was also tried. The factors were the same. As it is possible to see in Figure 4.11, the overshoot is much smaller than in the one step case. That should prove the benefits of having a longer horizon, even though the computational error increases exponentially.

4.6.2.2 FS-MPC with Position PI

The second test was carried out with the speed and current FS-MPC with cascaded position PI. The cost function weighting factor were again tuned after an iterative process of trial and error. In this case the performance with only one step prediction horizon was too poor to be mentioned, so only the results with three steps are presented. The optimal parameters are shown in Table 4.3.

As it can be seen in Figure 4.12, the dynamical behaviour is comparable to the case presented before. The overshoot though could not be avoided and it is quite

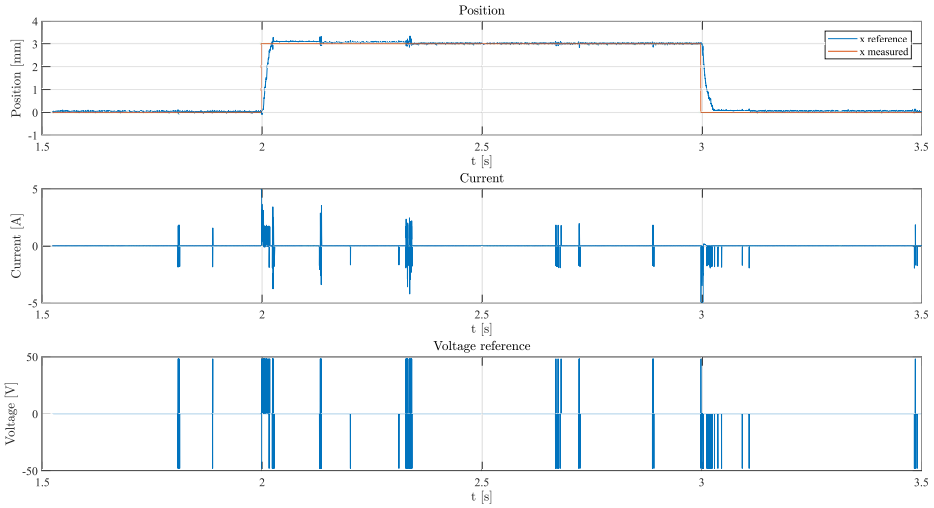

Figure 4.11: Experimental results: FS-MPC, three steps prediction horizon.

Table 4.3: FS-MPC with cascaded position PI: parameters. Experimental case.

Factor	Value
λ_v	250
λ_i	1
k_p	600
k_i	0.8

large. The only advantage of this structure is the less sensitivity to disturbances, in fact the controller, when it reaches the reference, remains idle. In the previous case it could be seen that even at steady state, the controller was setting the voltage reference to non zero values.
It has to be noted that without the integral part of the PI controller the system does not work at all, not reaching the reference value.

4.6.2.3 FS-MPC with Position Reference Modification

Finally the FS-MPC with position modification was tested. The cost function weighting factor were tuned after an iterative process of trial and error in this case too. Both tests with a prediction horizon of one and three were carried out.
First the one step prediction horizon algorithm was tested. The optimal weighting factors are shown in Table 4.4.

As it can be seen in Figure 4.13, the position target is reached quickly, but a

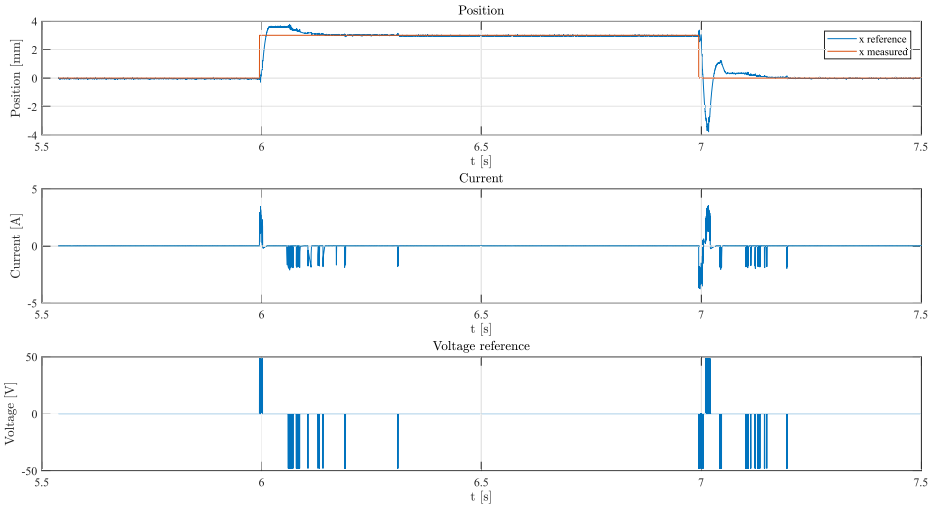

Figure 4.12: Experimental results: FS-MPC with cascaded position PI, three steps prediction horizon.

Table 4.4: FS-MPC with position reference modification: weighting factors, one step. Experimental case.

Factor	Value
λ_x	$1 \cdot 10^5$
λ_v	5
λ_i	0
k_p	100
k_i	50

significant overshoot is present both in rising and falling position transitions.
Then the three steps prediction horizon was also tried. The factors were changed to those in Table 4.5. As it is possible to see in Figure 4.14, the overshoot is eliminated. The overall dynamic is though slower, due to the reduce position weighting factor.

From the experimental tests, a general behaviour of the system, depending on the weighting factors choice can be inferred.
The current factor is not significant neither in the FS-MPC nor in the Position Reference Modification structures: changing it only brings minor modifications and mostly unfavourable. On the other hand, in the Position PI solution, it is a term necessary to ensure a proper operation of the machine, since the control of the speed alone would not lead to a sufficient stability and to too big oscillations.
The speed factor can never be neglected, since it is necessary to eliminate an unsta-

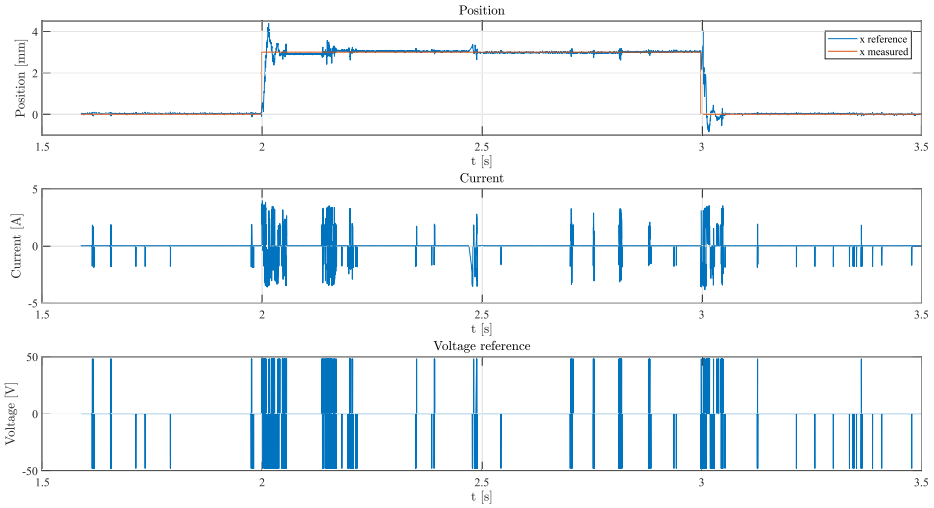

Figure 4.13: Experimental results: FS-MPC with position reference modification, one step prediction horizon.

Table 4.5: FS-MPC with position reference modification: weighting factors, three steps. Experimental case.

Factor	Value
λ_x	$1 \cdot 10^4$
λ_v	5
λ_i	0
k_p	200
k_i	80

ble response, caused by an overcompensation of the position error by the system. Unfortunately this cost function coefficient depends on the observed speed, so it would be interesting to reduce its importance in the control's dynamics.

Finally, the position factor is the most important. It is necessary to set it to a large value, both because the position is expressed in meters and because the response is supposed to be fast. A too large factor is to avoid, since it would cause too a sensitive system, reacting even to small disturbances or measurement errors.

The proportional and integral terms are less difficult to tune, but nevertheless crucial to the system. In the solution with the position PI they are both essential for the correct operation of the machine. In the position modification structure the integral part may be omitted, but a worse quality of the operation is then obtained. The proportional gain is set to large values to get a quicker transient response.

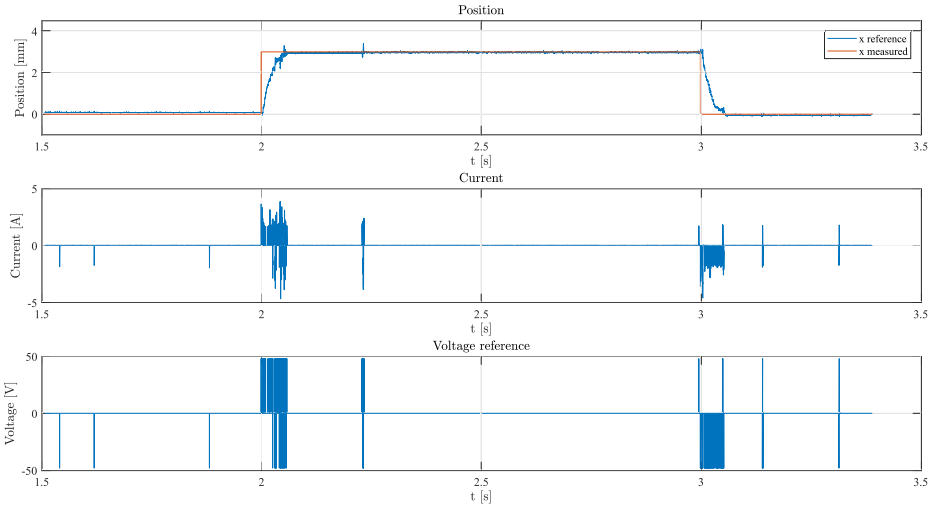

Figure 4.14: Experimental results: FS-MPC with position reference modification, three steps prediction horizon.

Chapter 5

Conclusion

In this thesis, three FS-MPC control structures with a EKF observer for an electromagnetic actuator are presented and simulated. The code generation of the algorithm is also discussed and tested in a FPGA in the loop configuration, showing that a good performance algorithm can fit into even a low cost FPGA, such as the Cyclone III.

The results show that a stand-alone FS-MPC structure is not suitable to react to disturbances applied to the system, even when a load observer is implemented. The usage of a PI structure can greatly improve the overall performance of the system, not only by assuring a zero error steady state position reference tracking, but also a faster transient behaviour.

In either solution, the tuning of the cost function parameter is not an easy task and requires a trial and error process.

The pros and cons of the FS-MPC structures can be summarised as:

Table 5.1: FS-MPC structures comparison summary.

Pros	**Cons**
Multivariable control in one single structure	Long design time, especially in comparison to a PI controller
Variable constraints easy implemented	Need of a good knowledge of the system to be controlled
Good dynamical behaviour	Difficult tuning
- -	Hardware requirement Need an observer to estimate speed

Finally, in Table 5.2 a more detailed comparison between the three experimentally tried structure is presented.

Table 5.2: FS-MPC structures comparison summary.

	FS-MPC	**Position PI**	**Pos. ref. correction**
Transient response	Quick	Mediocre	Quick
Steady state behaviour	Mediocre	Good	Mediocre
Computational effort	High	Medium	High
Parameters to be tuned	3	2 + 2	3 + 2

5.1 Personal Contribution

My personal contribution can be summarised as:

- Literature review about MPC and FS-MPC.
- Design of a suitable FS-MPC based control structure.
- Choice and design of a state observer with load estimation.
- Comparison between the different control structures.
- Conversion of the block diagram into VHDL code.
- Implementation of the observer on an FPGA with a simple cascaded control structure.
- Integration of the control in C language into the Pentium system.
- Implementation of the three MPC based controllers with the state observer on the Intel processor.

5.2 Future Work

Several are the future possibilities: a first step may be make the FPGA and the processor working together, running the EKF on the FPGA at a higher sampling time and the controller on the processor, as it is already programmed.

Moreover, in case a new H-bridge is available, capable of working at higher switching frequency, it would be interesting to test the functionality of the controller running directly on the FPGA.

Also it is necessary to test the behaviour of the system when an external load is applied.

Finally, a performance comparison with other predictive structures, such as LQR, can also be carried out.

Bibliography

[1] A. Linder and R. Kennel. Model predictive control for electrical drives. In *2005 IEEE 36th Power Electronics Specialists Conference*, pages 1793–1799, June 2005.

[2] Arne Linder, Rahul Kanchan, Peter Stolze, and Ralph Kennel. *Model-Based Predictive Control of Electric Drives*. Cuvillier Verlag, 2010.

[3] P. Cortes, M. P. Kazmierkowski, R. M. Kennel, D. E. Quevedo, and J. Rodriguez. Predictive control in power electronics and drives. *IEEE Transactions on Industrial Electronics*, 55(12):4312–4324, Dec 2008.

[4] J. Rodriguez, M. P. Kazmierkowski, J. R. Espinoza, P. Zanchetta, H. Abu-Rub, H. A. Young, and C. A. Rojas. State of the art of finite control set model predictive control in power electronics. *IEEE Transactions on Industrial Informatics*, 9(2):1003–1016, May 2013.

[5] E. Fuentes, D. Kalise, J. RodrÃguez, and R. M. Kennel. Cascade-free predictive speed control for electrical drives. *IEEE Transactions on Industrial Electronics*, 61(5):2176–2184, May 2014.

[6] Tino Müller. *FPGA based real-time-system construction and user manual*. Lehrstuhl für Elektrische Antriebssysteme & Leistungselektronik - Technische Universität München Arcistr. 21 München.

[7] S. Bolognani, S. Bolognani, L. Peretti, and M. Zigliotto. Design and implementation of model predictive control for electrical motor drives. *IEEE Transactions on Industrial Electronics*, 56(6):1925–1936, June 2009.

[8] Shakib Hasan. Position control of an electromagnetic actuator using model predictive control. mathesis, Lehrstuhl für Elektrische Antriebssysteme & Leistungselektronik - Technische Universität München, October 2014.

[9] Dan Simon. Using nonlinear kalman filtering to estimate signals.

[10] Rachel Kleinbauer. *Kalman filtering implementation with Matlab*. 2004.

[11] Gary Bishop Greg Welch. *An Introduction to the Kalman Filter*. 2001.

[12] A. Radke and Zhiqiang Gao. A survey of state and disturbance observers for practitioners. In *2006 American Control Conference*, pages 6 pp.–, June 2006.

[13] D. Janiszewski. Extended kalman filter based speed sensorless pmsm control with load reconstruction. In *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics*, pages 1465–1468, Nov 2006.

[14] S. C. Lee and H. S. Ahn. Sensorless torque estimation using adaptive kalman filter and disturbance estimator. In *Proceedings of 2010 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, pages 87–92, July 2010.

[15] S. Braune, S. Liu, and P. Mercorelli. Design and control of an electromagnetic valve actuator. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 1657–1662, Oct 2006.

[16] R. M. Hermans, M. Lazar, S. Di Cairano, and I. Kolmanovsky. Low-complexity model predictive control of electromagnetic actuators. In *IEEE EUROCON 2009*, pages 1972–1977, May 2009.

[17] P. Mercorelli. A sensorless control using a sliding-mode observer for an electromagnetic valve actuator in automotive applications. In *6th IET International Conference on Power Electronics, Machines and Drives (PEMD 2012)*, pages 1–6, March 2012.

[18] Altera Corporation. *Altera Cyclone III Product Table*.

[19] Altera. *Quartus II Handbook*.

[20] Mitsubishi. Pm100rl1a060 datasheet, 2009.

[21] Mitsubishi Electric Europe BV. Evbl1s1, 2009.

Appendix A

List of Symbols

Observed value	$\hat{}$
Scaled quantity, in per unit	^{s}
Per unit value	$_{pu}$
Value at time instant k	$_k$
Jacobian matrix	$\mathbf{J}$

R	Coil resistance $[\Omega]$
L	Coil inductance $[H]$
m	Moving mass of the actuator $[kg]$
T_s	Time step of actuator discrete time model $[s]$
T_{se}	Time step of the EKF $[s]$
K_e	Actuator back e.m.f. constant $[\frac{Vs}{m}]$
K_f	Actuator force constant $[\frac{N}{A}]$
x_0	Zero stroke position $[m]$
I_{max}	Actuator current base value $[A]$
v_{max}	Actuator speed base value $[\frac{m}{s}]$
x_{max}	Actuator position base value $[m]$
F_{max}	Load base value $[N]$

U_{max}	Voltage base value $[V]$
V_{dc}	DC supply voltage $[V]$
x	Actuator position $[m]$
i	Actuator current $[A]$
v	Actuator speed $[\frac{m}{s}]$
F_L	Load applied on the actuator $[N]$
U	Stator voltage of the actuator $[V]$
$\mathbf{A}$	State matrix of the actuator
$\mathbf{B}$	Output matrix used in the EKF
$\mathbf{C}$	Output matrix of the actuator
$\mathbf{x}$	State vector of the actuator
$\mathbf{y}$	Output vector of the actuator
$\mathbf{u}$	Input vector of the actuator
$\mathbf{F}$	Jacobian matrix used in the EKF
$\mathbf{P}$	Variance used in the EKF
$\mathbf{K}$	EKF gain matrix
$\mathbf{I}$	Identity matrix
$\mathbf{Q}$	Systems noise variance matrix used in the EKF
$\mathbf{R}$	Measurements noise variance matrix used in the EKF
J	FS-MPC cost function
N	FS-MPC Prediction horizon
λ_x	Position error weighting factor in FS-MPC cost function
λ_v	Speed weighting factor in FS-MPC cost function
λ_i	Current weighting factor in FS-MPC cost function

Appendix B

Simulink diagrams and C codes

B.1 EKF diagrams

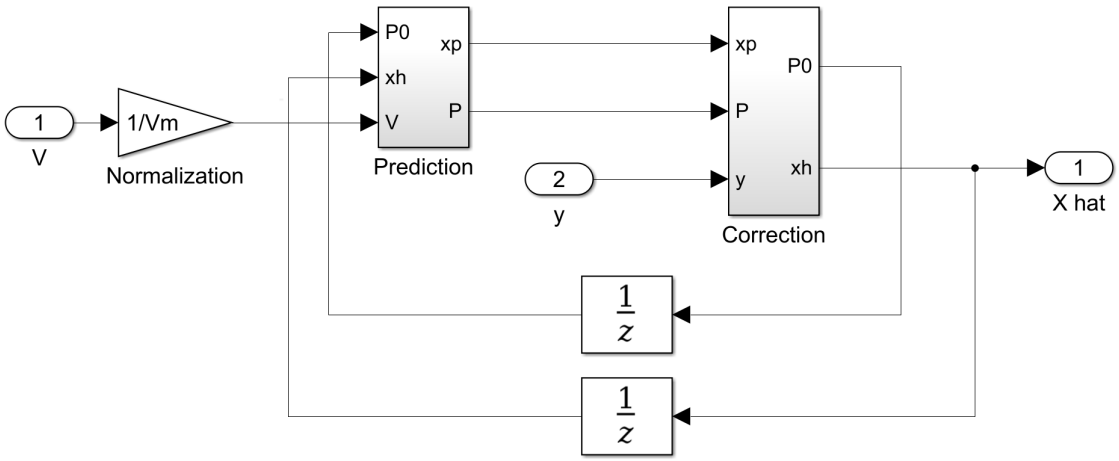

Figure B.1: EKF main block.

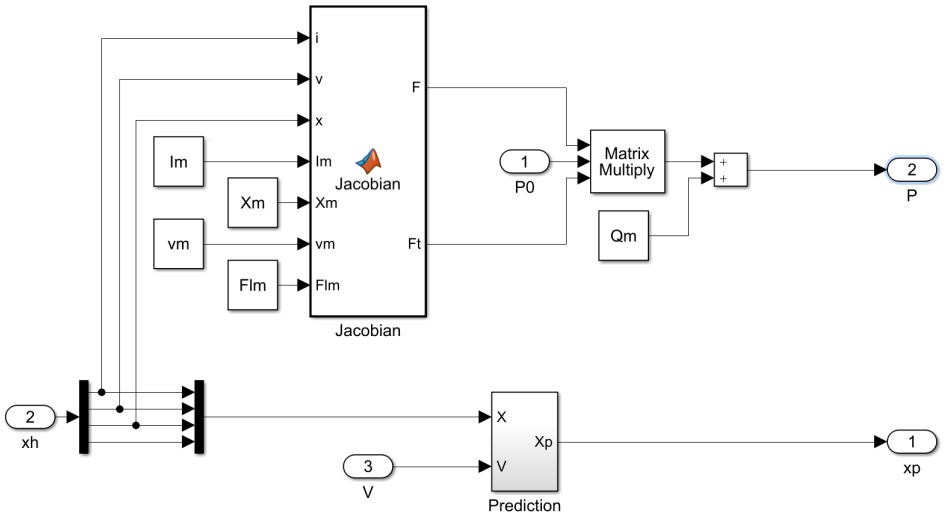

Figure B.2: EKF prediction section.

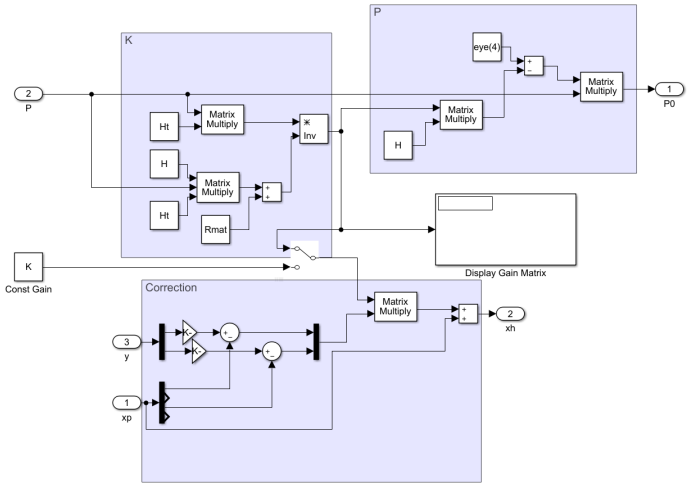

Figure B.3: EKF correction section.

B.2 RTAI and Linux codes

Code of the Linux module to log data.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
//This library is used by the shared memory functions
```

```
#include <rtai_shm.h>
//Time library to add the date to file name
#include <time.h>
//General definitions. The used ones are the shared data declarations.
#include "./../wc1/definitions.h"
//Here the shared memory data type is declared:
/*
typedef struct {
  char print;
  float time;
  float V;
  float imeas;
  float vmeas;
  float xmeas;
  float Flmeas;
  float xRef;
} SHMEM;
*/

static int end;

//To end the program: hit Ctrl+c (SIGINT)
static void endme(int dummy) { end=1; }

int main (void)
{
  //This is the shared data structure declared in parameters.h. It
    contains the data logged
  SHMEM *data;
  //Pointer to log file
  FILE *fp;
  //Name of the log and format
  char Name[] = "./../Log_PC104", Format[] = ".csv", fName[256] = "";
  time_t timeAbs = time(NULL);
  struct tm LogDate = *localtime(&timeAbs);
  //File name is in format Log_PC104_DAY_MONTH_YEAR_HOUR_MINUTE.csv
```

```
sprintf(fName, "%s_%d_%d_%d_%d_%d%s", Name, LogDate.tm_mday, LogDate.
 tm_mon + 1, LogDate.tm_year + 1900, LogDate.tm_hour, LogDate.tm_min
 , Format);

//Connects the SIGINT signal (Ctrl+c) to the function endme to close
 the program
signal(SIGINT, endme);

//Open the results file in writing mode
if((fp = fopen(fName, "w")) == NULL)
{
  fprintf(stderr, "Error in writing the results file.");
  return -1;
}
else
{
  fprintf(stderr,"File created\n");
  //Writes the first line, with the header (quantities logged).
 Remember to put every string between " if saving a .csv, so the
 softwares can tell it's a string.
  fprintf(fp, "\"Time\";\"V\";\"i\";\"v\";\"x\";\"F\";\"xRef\";\n");

  //Reserve a memory area for the datas from the RTAI app.
  data = rtai_malloc(nam2num(SHMNAM), sizeof(SHMEM));
  if(data == NULL)
  {
    fprintf(stderr, "Error in rtai_malloc.");
    return -1;
  }
    fprintf(stderr,"Shared memory initialized\n");
}

//Loop forever, until ctrl+c is pressed.
while (!end) {
//If there is a value to print
  if(data->print)
  {
```

```
    //It is printed in the format NUMBER;NUMBER and so on. It is
    possible to change the printing format modifying this function.
    fprintf(fp, "%1.5f;%f;%f;%f;%f;%f;\n", data->time, data->V, data
    ->imeas, data->vmeas, data->xmeas, data->Flmeas, data->xRef);
    //Resets the print flag
    data->print = 0;
    }
  }

  //Closing operations.
  rtai_free(nam2num(SHMNAM), &data);
  fclose(fp);
  fprintf(stderr,"Closing...\n");
  return 0;
}
```

Code of the RTAI module. Part of it was written by Peter Stolze.

```
//System files
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/stringify.h>
#include <rtai_sched.h>
#include <rtai_fifos.h>
#include <rtai_proc_fs.h>
#include <rtai_math.h>
#include "rtai_prog.h"
#include "rtai.h"
#include <rtai_shm.h>
#include <rtai_nam2num.h>

//User created header files
#include "inout.h"
#include "toolbox.h"
#include "Controllers.h"
```

```
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("PICMG 1.0 FPGA RTAI");
MODULE_AUTHOR("Fabio Mandrile");

SHMEM *data;
float a;
long long t0, t = 0;

int isa_rtirq(void)
{
  int i;
  float I,x;
  float xRef, xErr, K;
  float Ap, An;
  static long hfsin_sample = 0;

  isr_start();
  //Beginning of RTAI code

  //Enable PWM
  PWM_enable();

  //Acknowledge Interrupt
  Interrupt_Ack();

  //Get the time when the real time operation starts
  if(!start)
  {
    start = 1;
    t0 = rt_get_cpu_time_ns();
  }

  //Generation of a square wave reference signal. Frequency 1 Hz.
  xRef = blockwave(3.0, 0.0, 0.5, 0.0, &hfsin_sample,20e3);

  //Sensor reading and conversion
  I = (float) (ReadADC(6) - 2067) / 263.16;
```

```
x = (float) (ReadADC(7) - 2069) / -393.16;

//Any FSMPC function goes here
Vref = FSMPC_PosMod_1S(I, x, vh * vm, Flh * Flm, xRef, Vref);

//Scaled system
x = x / 5.0;
I = I / Im;
//Kalman
xErr = x;
//A constant back emf value is used
//K = kq_s * xErr * xErr + ks_s * xErr + k;
K = k;
ip = Ts / L * Vref * Vm / Im + (1 - R/L*Ts) * ih - Ts/L * K * vh * vm
   / Im;
vp = Ts / mass * K * ih * Im / vm + vh - Ts/mass * Flh * Flm/vm;
xp = Ts * vh * vm / Xm + xh;

Di = I - ip;
Dx = x - xp;
ih = ip + 1.0 * Di;
vh = vp + 0.3388 * Dx;
xh = xp + 0.04371 * Dx;
Flh = Flh - 0.1602 * Dx;

Ap = Vref / 50.0 * MAXDUTY2 + MAXDUTY2;
An = -Vref / 50.0 * MAXDUTY2 + MAXDUTY2;



//**********************Data log*********************************
//This counter can be used for data decimation.
if(cont % 1 == 0)
{
  //Sets the print data flag
  data->print = 1;
  //Time stamp. The time is given from control start
  t = rt_get_cpu_time_ns();
```

```
    a = (float) (t - t0) / 1E9;
    data->time = a;
    data->V = Vref;
    data->imeas = I * Im;
    data->vmeas = vh * vm;
    data->xmeas = x * Xm * 1000.0;
    data->Flmeas = Flh;
    data->xRef = xRef;
  }
  else
  //Nothing to print. It avoids repetitions of the same value.
    data->print = 0;

  cont++;
  if(cont >= 20000)
    cont = 0;

  //End of RTAI code
  isr_end();

  SetPWMDutycycle(0, (int) Ap);
  SetPWMDutycycle(1, (int) An);

  return(IRQ_HANDLED);
}
  //RTAI inizialization function. Run once at the beginning
int init_module(void)
{
  printk("Start %s!\n", cfile);

  //Enables interrupt function
  rt_request_global_irq(nr_irq, (void*)isa_rtirq);
  rt_enable_irq(nr_irq);

  //Enable PWM
  PWM_enable();
  //Set PWM mode and max. duty cycle
```

```
  Set_PWM_mode(PWM_MODE, MAX_DUTYCYCLE);

  Interrupt_Ack();

  //Set AD filter usage
  ADToSend(AD_FILTERED_VALUES);

  //Allocate shared memory for data logging
  data = rtai_kmalloc(nam2num(SHMNAM), sizeof(SHMEM));

  return(0);
}
//Cleanup function. Runs when real time operation is disabled.
void cleanup_module(void)
{
  //Reset hex display to 0x000
  WriteHex(0);

  //Set inverter switching state to "000"
  Set3LNPCSwitchingState(1, 1, 1);

  //Disable PWM
  PWM_disable();

  //Disable interrupt
  rt_disable_irq(nr_irq);
  rt_free_global_irq(nr_irq);

  //Free shared memory
  rtai_kfree(nam2num(SHMNAM));

  return;
}

//Begin and end of interrupt function.
void isr_start(void)
{
```

```
  save_cr0_and_clts(cr0);        //To save Linux cr0 state. Always to be
    done.
  save_fpenv(linux_fpe);         //To save Linux FPU environment

  if(isr_first_time)
    restore_fpenv(task_fpe);     //To restore your FPU environment
  else
    isr_first_time = 1;
}

void isr_end(void)
{
  save_fpenv(task_fpe);          //To save your FPU environment. Needed
   only if there
  restore_fpenv(linux_fpe);      //To restore the previously saved Linux
    FPU environment
  restore_cr0(cr0);              //To restore Linux cr0. Always to be
   done.
  rt_ack_irq(nr_irq);
}
```