REPORT LAB 2

Group 7, Thursday shift: Grimaldi Riccardo, Krassowizkiy Michael, Peron Davide, Vitetta Emanuele

Academic Year 2022/2023

1 Introduction

1.1 Activity Goal

The goal of this Laboratory was to design, test and compare different kinds of discrete position controllers for a DC motor, both using Simulink simulations and the real DC motor in the laboratory (motor number 7). Specifically, we tested both PID and State-Space architectures using

- different discretization methodologies (emulation, direct design)
- different approximation techniques (Forward Euler, Backward Euler, Trapezoidal, Exact)
- different sampling times $T_s = (1ms, 10ms, 50ms)$

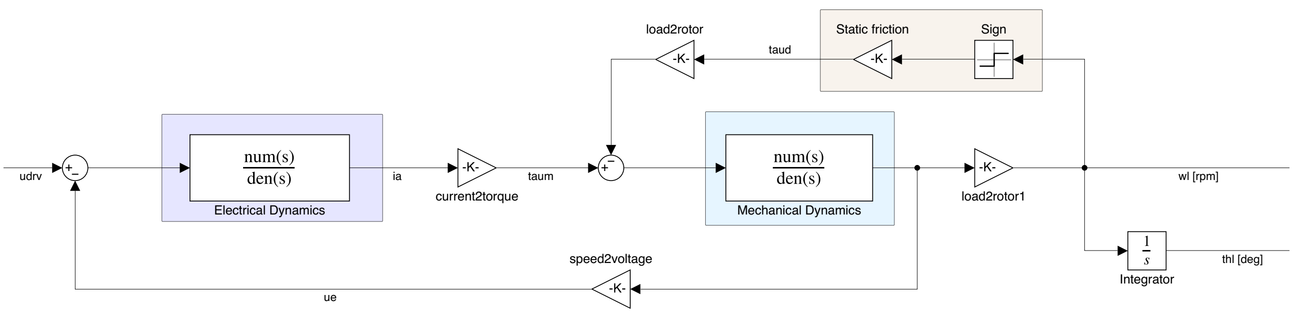
1.2 System and Model

The system of interest is a DC motor (Quanser SRV-02 gearmotor), that is driven by a voltage driver and is connected via a gearbox to a rotating disc, which we refer to as the load. The objective is to control the *output angular position* θ_l of the load through a *voltage input* u. The control commands are generated through the Simulink real-time package running on a host computer. A digital-to-analog converter (DAC) transforms the digital signals into an analog voltage signal that enter the voltage driver, which amplifies the signal. To close the loop, the position is measured with the help of an optical encoder, and the digitized signal is fed back to the host computer.

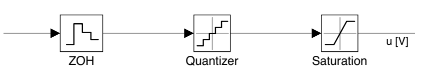
For the simulation part, we model the system in the Simulink environment using linear transfer function approximations, adding the most relevant nonlinearities, which are:

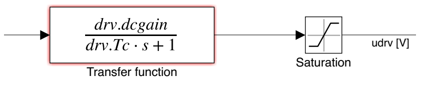
- the saturation of the input voltage introduced by the voltage driver
- the Coulomb friction introduced by the mechanical part of the motor
- the quantization introduced by the encoder and the DAC

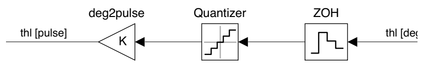
The block diagrams modeling the separate parts of the motor are shown in 1. The values of the saturation interval, the quantization resolution and the friction constant are collected in appendix A.



(a) DC Motor + Gearbox







(b) DAC (c) Voltage Driver (d) Encoder

Figure 1: Full motor model block diagram used for simulations. All parameters are to be found in the appendix A.

For the design part, to simplify the design of the controller we consider the following second order linear approximation of the plant:

$$P(s) = \frac{k_m}{T_m s + 1} \frac{1}{Ns} \tag{1}$$

which is the transfer function mapping input $U(s)$ to output $\Theta_l(s)$. The gains $k_m = 77.8211 \frac{rad}{Vs}$, $T_m = 0.0297s$ are obtained by simplification of the model, and $N = 14$ is the gearbox ratio.

Equivalently, for the state space controllers we consider the following 2 dimensional state space realization $\Sigma = (A, B, C, D)$:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{1}{T_m} \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ \frac{k_m}{NT_m} \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \qquad D = 0 \tag{2}$$

and the associated state vector is $x = \begin{bmatrix} \theta_l & \dot{\theta}_l \end{bmatrix}^T$.

2 Task, Methodologies and Results

2.1 Task

The Goal of the assignments was to design a digital position controller for the DC motor, using both emulation and direct design approaches. Moreover, the effects of different kinds of discretization on the controller performances were evaluated. The desired requirements for the controllers are

- perfect steady state tracking of step position (load side) references
- perfect steady state rejection of constant torque disturbances
- step response (at load side) with settling time $t_{s,5\%} \leqslant 0.2s$ and overshoot $M_p \leqslant 10\%$

2.2 Discrete PID Design via Emulation

2.2.1 Methodology

In the first assignments we designed a discrete time PID controller, using the emulation method. That is, we designed the PID in continuous time (see figure 2 and eq. (3)) and then used different discrete time approximations (Forward Euler, Backward Euler, Trapezoidal) and sampling times ($1ms$, $10ms$, $50ms$) to implement the controller in discrete time.

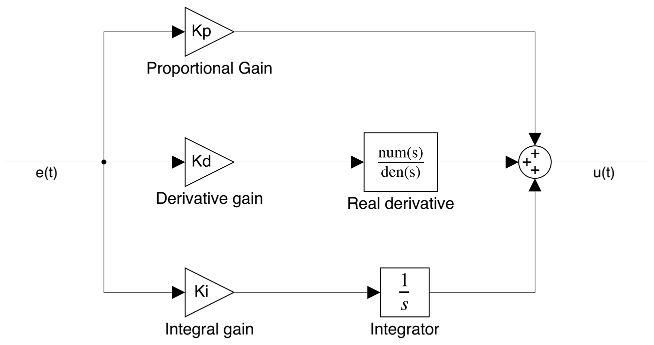


Figure 2: Block schematic of a realizable PID in continuous-time

$$C(s) = K_p + K_i\frac{1}{s} + K_d\frac{s}{T_l s + 1} \tag{3}$$

Specifically, $C(s)$ in (3) is the transfer function of the continuous time PID Controller, where K_p, K_i, K_d, T_l are design parameters. Let T_s be the sampling time, then the different discretized controllers are obtained as summarized in table 1.

Forward Euler	Backward Euler	Trapezoidal	Exact
$C(\frac{z-1}{T_s})$	$C(\frac{z-1}{T_s z})$	$C(\frac{2}{T_s}\frac{z-1}{z+1})$	$(1-z^{-1})\mathcal{Z}\{\frac{C(s)}{s}\}$

Table 1: Discretization of a continuous–time transfer function $C(s)$

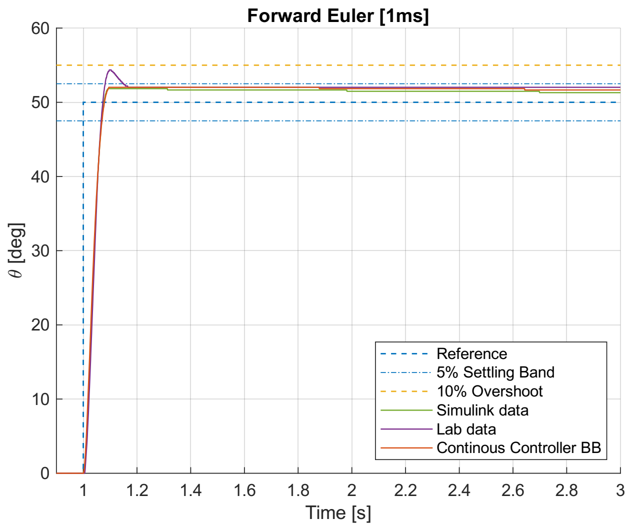
The PID gains were computed using the Bode Method, and $T_l = 0.0197s$ was designed according to the desired bandwidth of the system. For different T_s the gains change, as the delay introduced by the sampled system of approximately $\frac{T_s}{2}$ is taken into account in the continuous time design. The gains for different T_s are summarized in table 2.

2.2.2 Assignments 1,2 and 5 - Step Reference Tracking

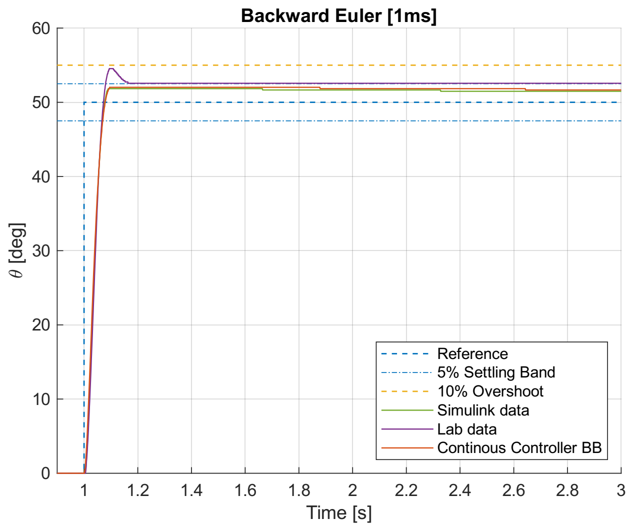
In this assignment we have tested the controller described above with a 50° step reference, using all discretization methods in table 1 with $T_s = (1ms, 10ms, 50ms)$. The step responses of the simulink model and the real system for $T_s = 1ms$ are shown in 3.

T_s	K_p	K_d	K_i
$1ms$	6.4060	0.0404	0.2542
$10ms$	6.2493	0.0687	0.1421
$50ms$	4.6174	0.1796	0.0297

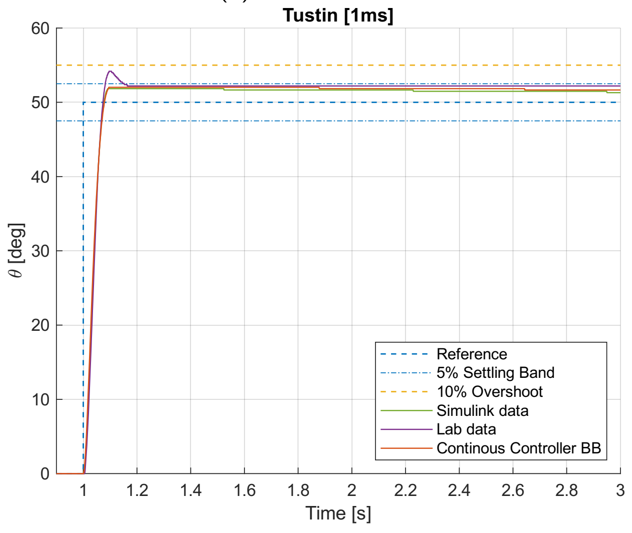
Table 2: Gains for the discretized PID for different sampling times



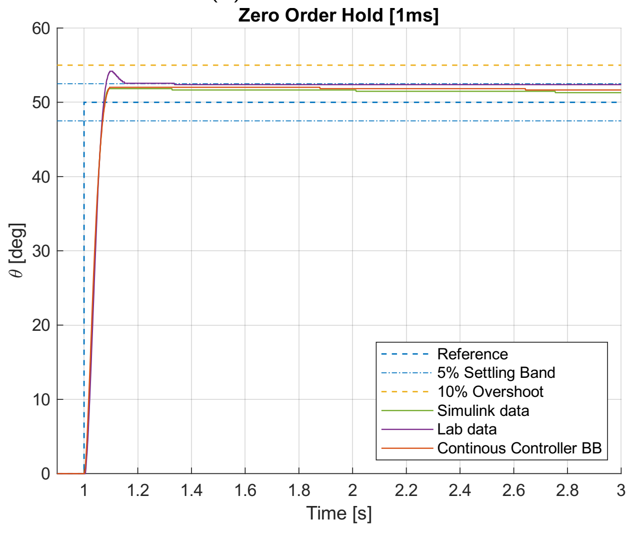
(a) Forward Euler



(b) Backward Euler



(c) Trapezoidal



(d) Zero Order Hold

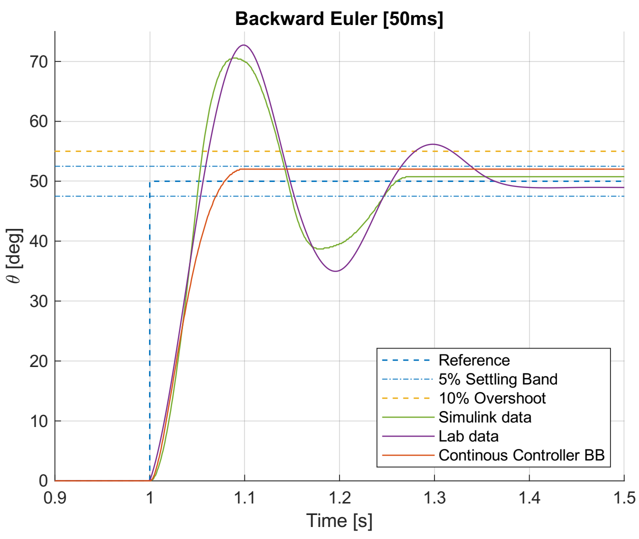
Figure 3: Step responses for a 50° step reference with $T_s = 1ms$ with different discretizations of a PID controller, compared with the continuous time response.

First of all, the controllers seem to have a small steady state error. This is an unexpected result, as we have an integrator action which should guarantee steady state asymptotic tracking. The problem is that only a small time window is shown. When considering a longer time interval of $100s$ for a simulation, we asymptotically reach the set point with a small chattering due to static friction. We conclude that the integrator gain was chosen too small, causing the slow asymptotic convergence to the set point. This is true for all the considered PID controllers.

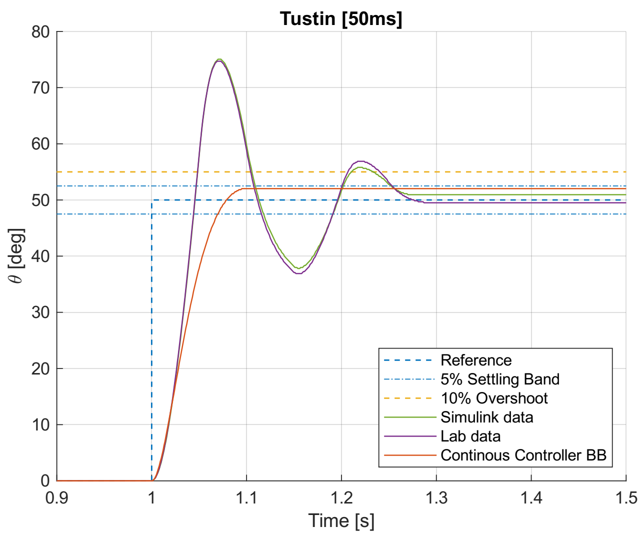
Going on with the analysis of the effects of discretization, we do not observe any significant difference

between the continuous controller and the discretizations. This result is expected: The sampling time is so small, with $T_s = t_{s,5\%}/150$, that the differences are negligible. In this case, the choice of the discretization method does not influence the performance. Moreover, we see that the simulink model and the real system show essentially the same behavior, except for a small overshoot in the real system. This is true for most of the results of this laboratory. From now on, we will highlight differences between the simulink model and real system responses only when they are relevant. The performance indeces are summarized in table 3.
For the case $T_s = 10ms$, there are no important qualitative differences. All discretizations give similar responses to the continuous time one. The performance indeces are summarized in table 3.

The interesting case is $T_s = 50ms$. There is still asymptotic tracking for the Backward Euler and Trapezoidal discretizations 4, with higher overshoot and longer settling time. A degradation in performance that is expected from the coarse discretization. The effects of discretization using Forward Euler and Exact method 5 are even worse. Exact method is no longer able to track the step reference asymptotically for the real system, and shows oscillating behavior. This was not predicted by the simulink model, where it still tracks the step reference, even if showing a bad behaving transient. Forward Euler is not able to track the reference asymptotically neither in the simulink model nor in the real system. The performance indeces are summarized in table 3.



(a) Backward Euler



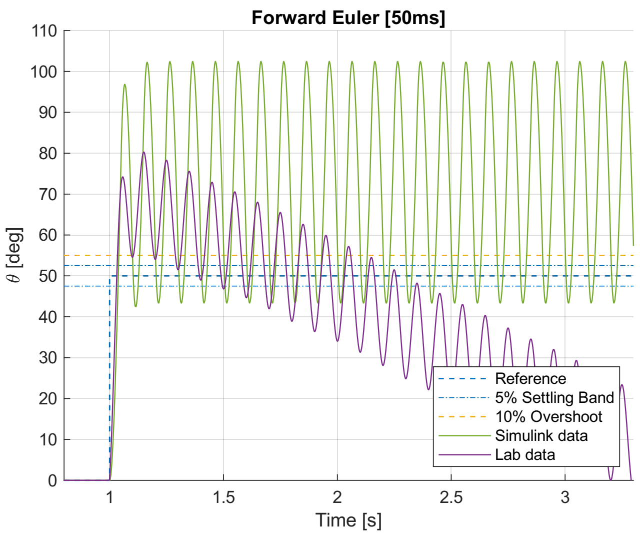
(b) Trapezoidal

Figure 4: Step responses for a 50° step reference with $T_s = 50ms$ with Backward Euler and Trapezoidal discretization of a PID controller, compared with the continuous time response.

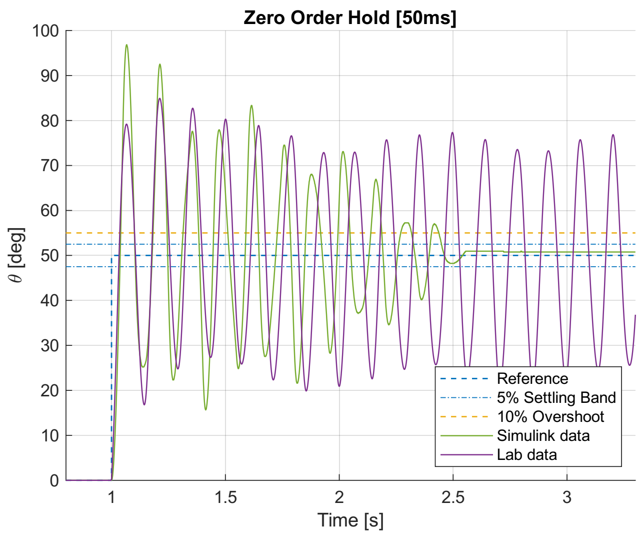
From the theory, we know that the Forward Euler method does not necessarily map stable continuous poles to stable discrete poles. Indeed the map $z = 1 + sT_s$, is more likely to end up outside of the unit circle when T_s is large. Therefore, the discretized controller may become unstable, even if the original continuous controller is stable. Indeed, the Forward Euler discretized controller has an unstable pole in $z = -1.537$. This shows that the control system is not internally stable and could explain the degraded performance.

From the theory we also know that the exact discretization, performed as in table 1, matches the behavior of the continuous time controller perfectly only when the tracking error is a stair signal with step time T_s. When T_s gets larger, the stair approximation of the tracking error gets coarser, causing the performance degradation.

Unlike the Forward Euler method, both Backward Euler and Trapezoidal map are guaranteed to map stable poles in continuous time into stable poles in discrete time, irrespective of the sampling time. So we expect them to be more robust to larger T_s than Forward Euler. In both cases, we nevertheless see a degradation with respect to the continuous time controller. Still the step reference is tracked asymptotically.



(a) Forward Euler



(b) Zero Order Hold

Figure 5: Step responses for a 50° step reference with $T_s = 50ms$ with Forward Euler and Exact discretization of a PID controller

Discretization method	System	$t_{s,5\%}$ $[ms]$	M_p $\%$	$t_{s,5\%}$ $[ms]$	M_p $\%$	$t_{s,5\%}$ $[ms]$	M_p $\%$
Sampling time		$1ms$		$10ms$		$50ms$	
Forward Euler	Simulink model	69	3.7	96	5.1	-	-
	Real plant	150	8.7	137	9.8	-	-
Backward Euler	Simulink model	68	3.7	893	9.1	247	41.1
	Real plant	-[1]	9.1	137	13.8	336	50.8
Trapezoidal	Simulink model	68	3.7	1091	7.3	252	50.1
	Real plant	152	8.4	138	11.6	253	49.4
Exact	Simulink model	68	3.7	103	6.6	1449	93.7
	Real plant	336	8.4	126	11.0	-	-

Table 3: Performance indeces for the different discratizetions and different T_s with a 50° step reference.

[1]The steady state error in the real plant is larger than the 5% error band

2.2.3 Assignment 3 - Anti-Windup

In this assignment we implement anti-windup architecture seen in figure 6. It is supposed to reduce undesired effects of saturation on the integral action, which leads to overshoot.

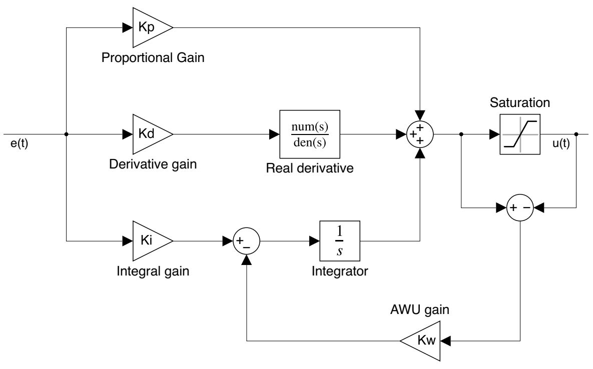
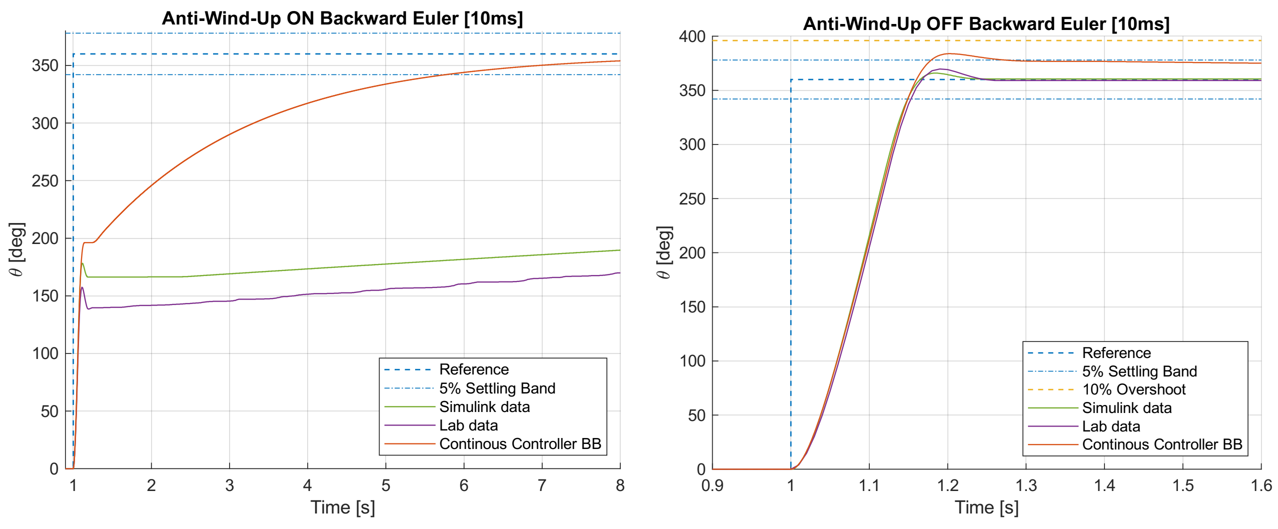
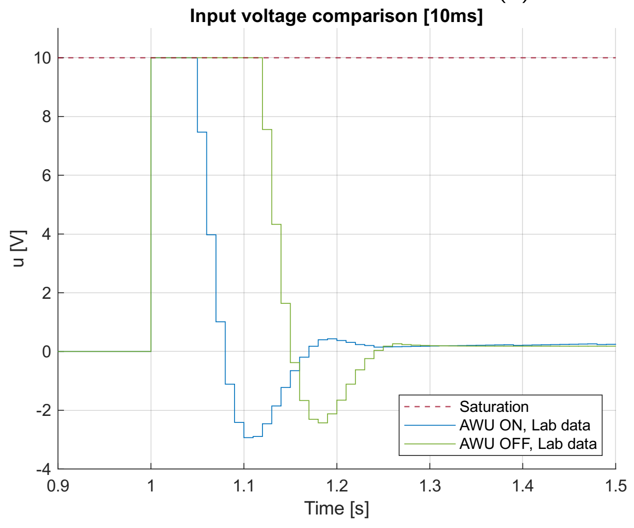


Figure 6: Block schematic of a realizable PID in continuous-time with Anti-Windup.



(a) Backward Euler AWU on (b) Backward Euler AWU off



(c) Input

Figure 7: (a)-(b):Step responses for a 360° step reference with $T_s = 10ms$ with Backward Euler discretization of a PID controller with and without Anti-Windup. (c):Input voltage comparison between Anti-Windup on and off in the real system data.

To test the Anti-Windup controller, we used a step reference of 360° and sampling time of $10ms$ using the Backward Euler discretization. The Anti-Windup gain was chosen to be $K_w = 25$. The step responses, with and without Anti-Windup, are shown in 7.

Both the continuous time step response and the step responses discretized at $T_s = 10ms$ show a very large rise time. From the plots the continuous time step response show a faster rise time than the discretized one. This is an unexpected phenomenon, as we just discretized the integrator. A possible reason is that a mistake was made in the setting of the Anti-Windup gain K_w, leading to different Anti-Windup gains in discrete and continuous time experiments. As expected, the input voltage saturates for a shorter amount of time when Anti-Windup is active. Clearly, the Anti-Windup architecture is not necessary as we are inside the overshoot performance specification even without anti-windup for the discretized architectures.

2.2.4 Assignment 4 (Optional) - Feedforward

In this assignment we implement a feedforward architecture seen in figure 8, to improve the tracking performance of a given signal. We still use Backward Euler discretization for $T_s = 10ms$. In particular, we consider the following reference signal:

$$a(kT) = \begin{cases} 900\,rpm/s & \text{if } 0 \leqslant kT < 0.5s \\ 0\,rpm/s & \text{if } 0.5s \leqslant kT < 1s \\ -900\,rpm/s & \text{if } 1s \leqslant kT < 2s \\ 0\,rpm/s & \text{if } 2s \leqslant kT < 2.5s \\ 900\,rpm/s & \text{if } 2.5s \leqslant kT < 3s \end{cases} \qquad \text{(acceleration reference)} \tag{4}$$

$$\omega(kT) = T\sum_{n=0}^{k} a(nT) \qquad \text{(velocity reference)} \tag{5}$$

$$\theta(kT) = T\sum_{n=0}^{k} \omega(nT) \qquad \text{(position reference)} \tag{6}$$

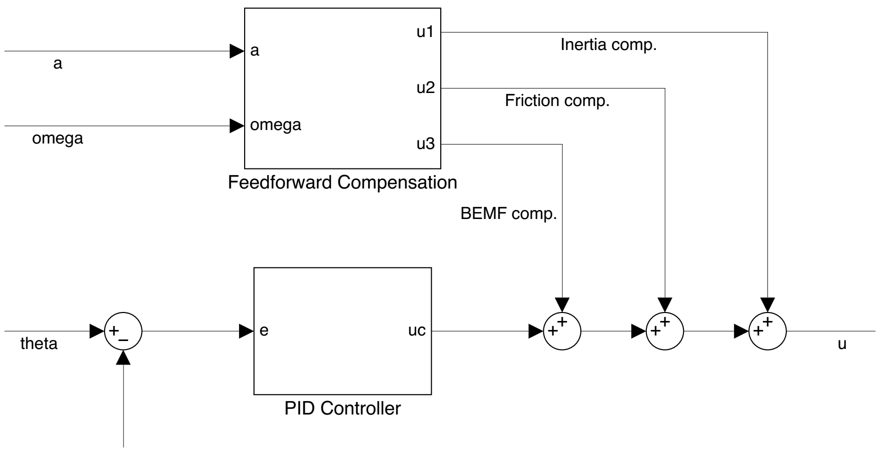
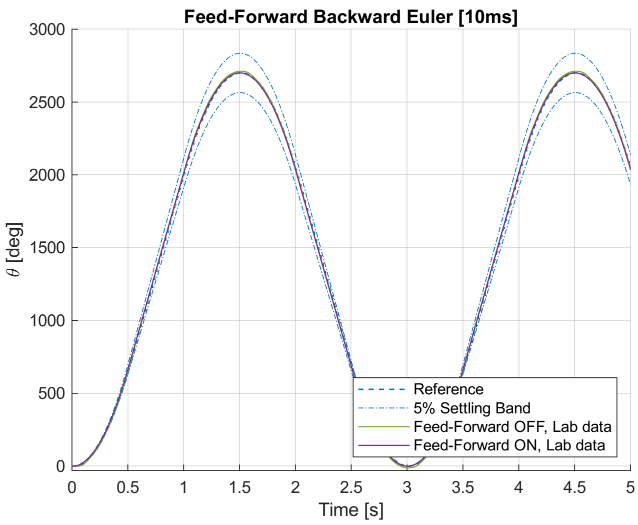
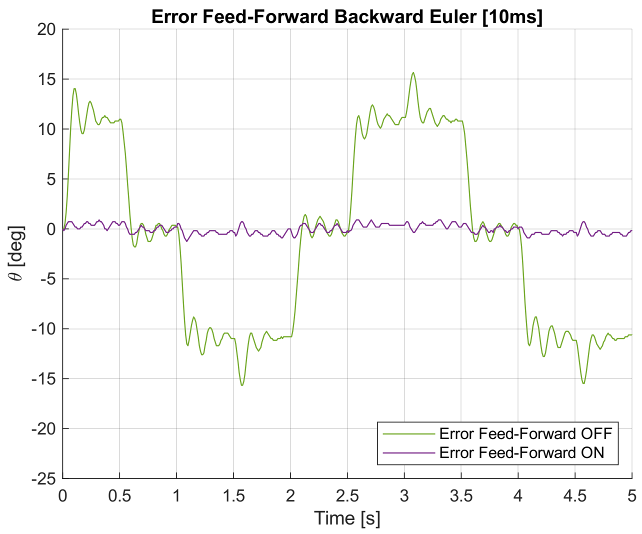


Figure 8: Block schematic of a PID controller in continuous-time with Feedforward compensation.

The reference signal and the system responses with and without feedforward are shown in figure 9a. The tracking error is compared in figure 9b. Again, $T_s = 10ms$ is sufficiently fast to not see a significant difference between the discretized and the continuous behavior observed in Laboratory 1. The feedforward architecture improves the tracking error in discrete time in the same way as we have seen in Laboratory 1 with a continuous time controller.



(a) Reference signal tracking



(b) Tracking errors

Figure 9: (a):Tracking of the reference signal (6) with and without the use of discretized Feedforward architecture with Backward Euler discretization and $T_s = 10ms$ (b):Tracking errors for the plot in (a)

2.3 Discrete State-Space Control Design via Emulation

2.3.1 Methodology

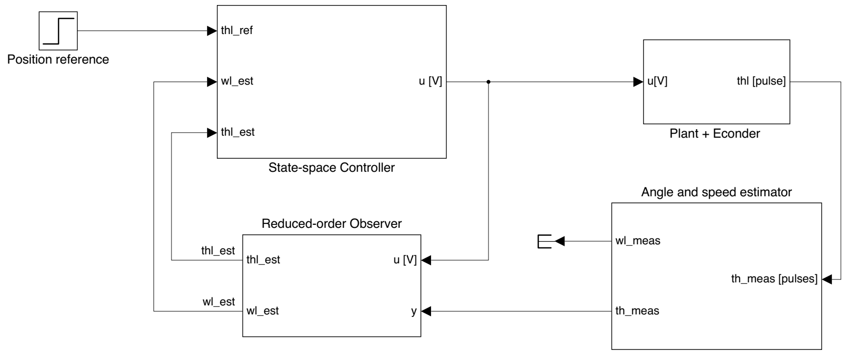
In these assignments, we designed a discrete time State-Space Controller using the emulation method. Specifically, we first design a nominal and robust state-space controller in continuous time. To reconstruct the state we used a one dimensional Reduced Order Observer, described in Handout 2, chapter 5. To discretize this continuous design, we kept the state feedback gain K the same and discretized the Reduced Order Observer (and the integral action for the robust approach) using different discretization methods (Forward Euler, Backward Euler) and different sampling times $T_s = (1ms, 10ms, 50ms)$. The architecture is shown in figure 10.

The design parameters for the continuous time design are the matrices N_x, N_u, K, K_i, L, and the detailed procedure for their computation is described in Handout 1, chapter 3.2-3. The unique pole $\lambda = -155.8239$ of the reduced order observer was chosen to be faster than the poles of the feedback system. The observer is then discretized according to the procedure described in Handout 2, chapter 4.1. The other parameters remain unchanged and are summarized in table 4.

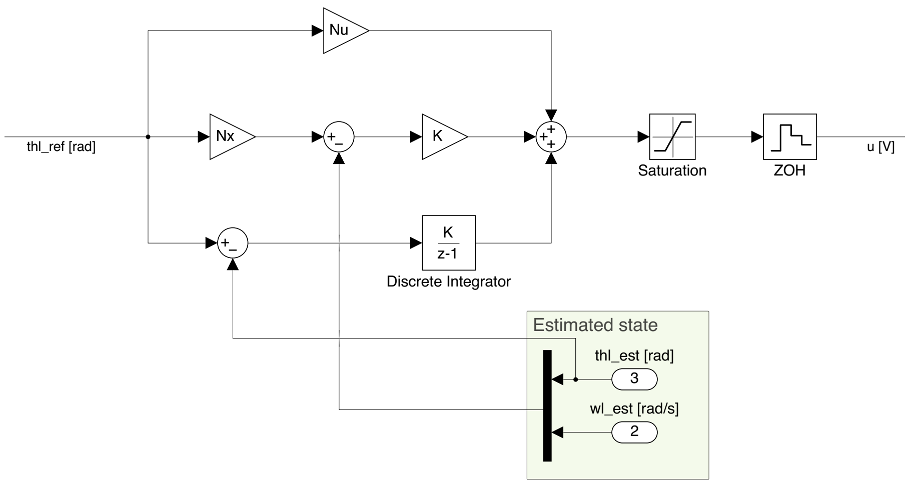
	N_x	N_u	K	K_i
Nominal	$\begin{bmatrix}1 & 0\end{bmatrix}^T$	0	$\begin{bmatrix}4.0954 & -0.0074\end{bmatrix}$	-
Robust	$\begin{bmatrix}1 & 0\end{bmatrix}^T$	0	$\begin{bmatrix}25.563 & 0.4697\end{bmatrix}$	377.5

Table 4: Emulation method nominal and robust State-Space parameters

We note that the discretization methods Backward Euler and Trapezoidal are not implementable in the real time system, due to an algebraic loop, that cannot be solved online. To overcome this issue, the throughput matrix of the observer for Backward Euler was modified, to not make the observer output dependent on the current control input. This is a necessary approximation to make the controllers work online.



(a) Complete Model



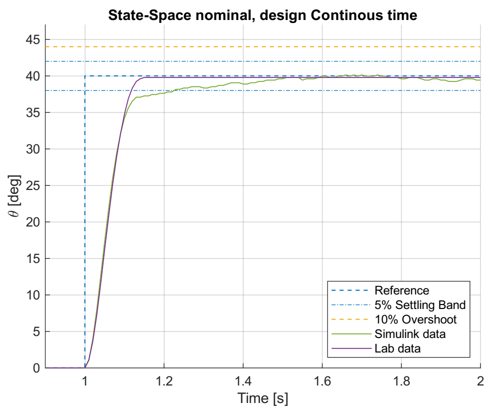
(b) State-space Controller

Figure 10: State-Space design with reduced observer (the integral action is active only in the robust approach)

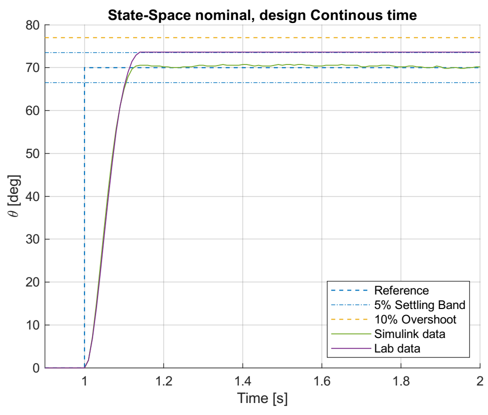
2.3.2 Assignments 1, 2 and 5 – Nominal and Robust design in continuous time

Before discretizing, the continuous time design for the nominal and robust State-Space approach was tested with for step references of 40°, 70° and 120°. The used design parameters are summarized in table 4. The simulation results for nominal are shown in 11 and for robust in 12.

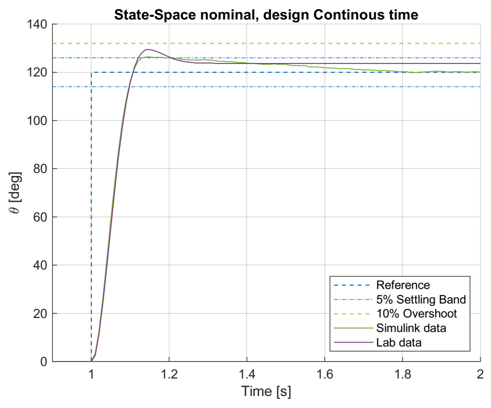
As we can see the continuous time nominal design tracks the step reference with a small constant tracking error. Instead the robust design is able to asymptotically track the references as expected, at the price of a significant overshoot. This is the expected behavior, which is the consequence of the added integrator. The behavior is in general comparable to the one we observed in Laboratory 1, using a real derivator to estimate the angular velocity instead of the reduced observer.



(a) 40°

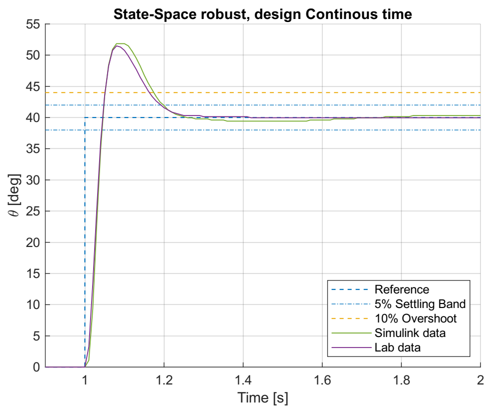


(b) 70°

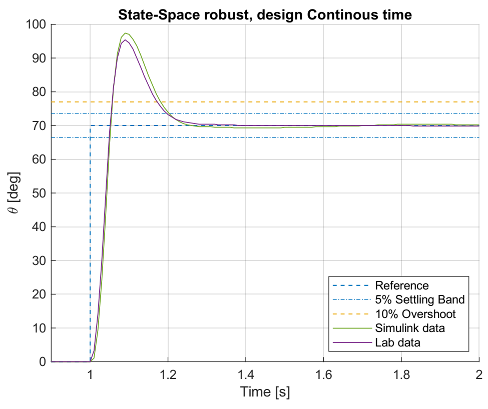


(c) 120°

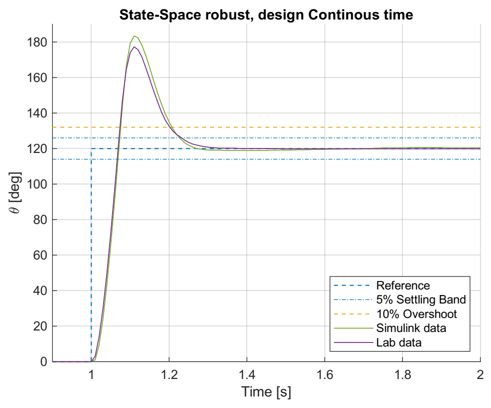
Figure 11: Step responses of $40^{\circ}, 70^{\circ}, 120^{\circ}$ references of the nominal State-Space continuous-time controller.



(a) 40°



(b) 70°



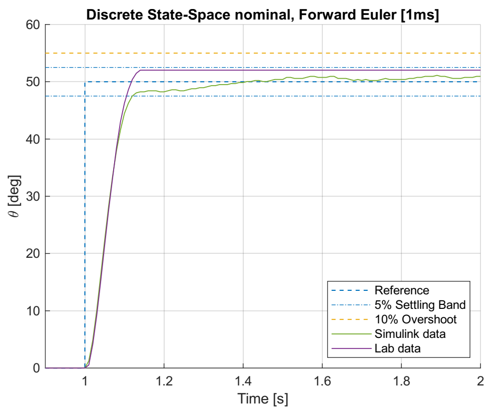
(c) 120°

Figure 12: Step responses of $40^{\circ}, 70^{\circ}, 120^{\circ}$ references of the robust State-Space continuous-time controller.

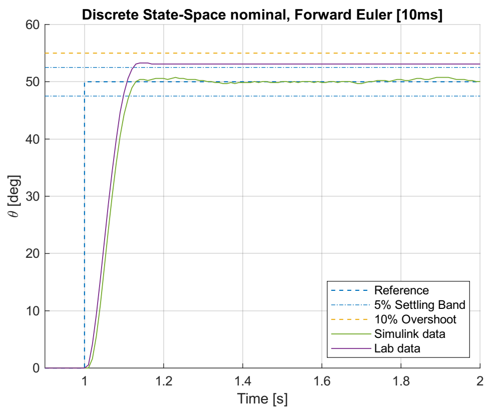
2.3.3 Assignments 3, 4 and 8 (Optional) – Nominal Discrete Time Control

In these assignments we tested the discretized nominal State-Space Controller using the different discretizations (Forward Euler, Backward Euler) and sampling times $T_s = (1ms, 10ms, 50ms)$ with a 50° step reference. As before the used design parameters are summarized in table 4.

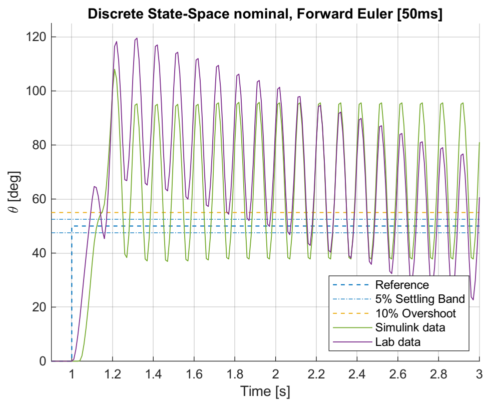
The simulation results and the lab data for different T_s are shown in figure 13 for Forward Euler and in figure 14 for Backward Euler.



(a) $1ms$



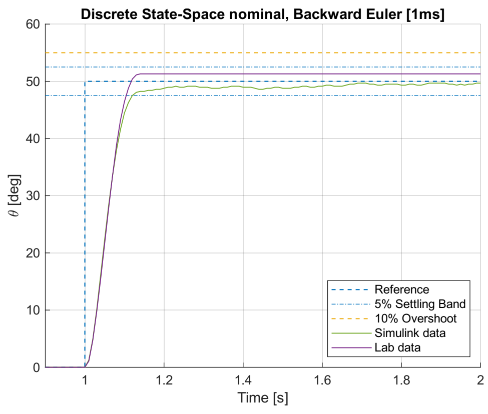
(b) $10ms$



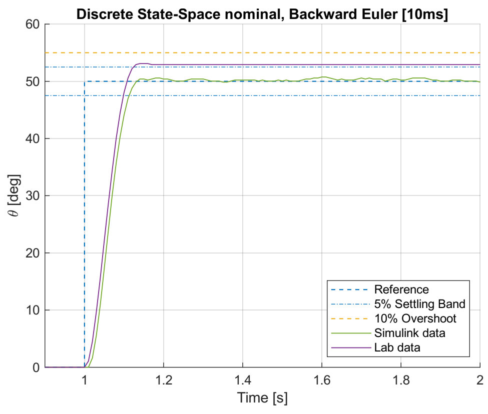
(c) $50ms$

Figure 13: Step responses of 50° reference of the discretized nominal State-Space controller using Forward Euler for $T_s = (1ms, 10ms, 50ms)$.

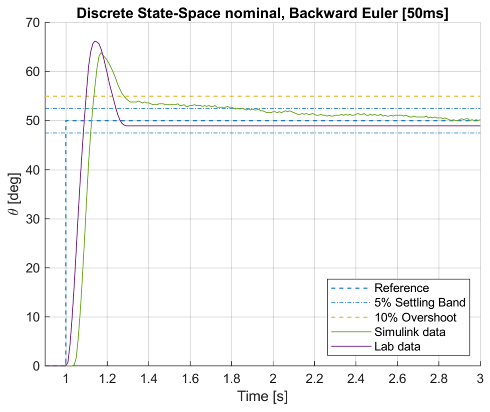
What we can observe, that similar to the PID emulation design, $T_s = 1ms$ and $T_s = 10ms$ do not significantly degrade the behavior. But for $T_s = 50ms$ the Forward Euler method gives rise to instability, that was expected given the results of the PID emulation. Instead Backward Euler is still able to track step references asymptotically, confirming the results from PID emulation. We note once again, the similar qualitative behavior of the simulation and experimental results. The performance indeces are summarized in table 5.



(a) $1ms$



(b) $10ms$



(c) $50ms$

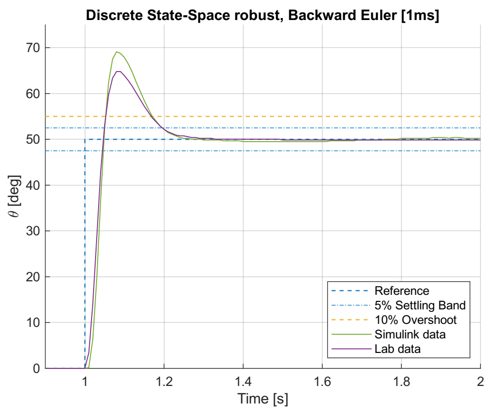
Figure 14: Step responses of 50° reference of the discretized nominal State-Space controller using Backward Euler for $T_s = (1ms, 10ms, 50ms)$.

Discretization method	System	$t_{s,5\%}$ $[ms]$	M_p $\%$	$t_{s,5\%}$ $[ms]$	M_p $\%$	$t_{s,5\%}$ $[ms]$	M_p $\%$
Sampling time		$1ms$		$10ms$		$50ms$	
Forward Euler	Simulink model	120	1.9	112	1.5	-	-
	Real plant	105	4.0	-	6.6	-	-
Backward Euler	Simulink model	1	0.0	113	1.5	853	27.8
	Real plant	105	2.6	-	6.2	242	32.5

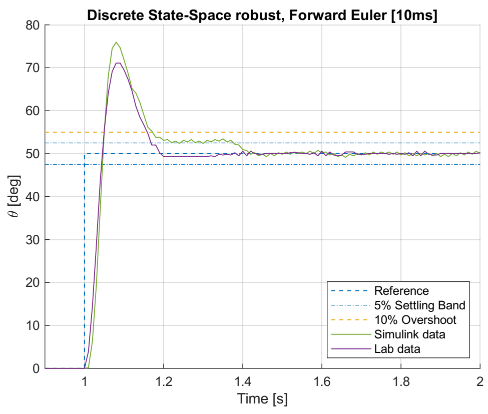
Table 5: Performance indeces for the different discretizetions and different T_s with a 50° step reference (nominal design)

2.3.4 Assignments 6,7 and 8 (Optional) – Robust Discrete Time Control

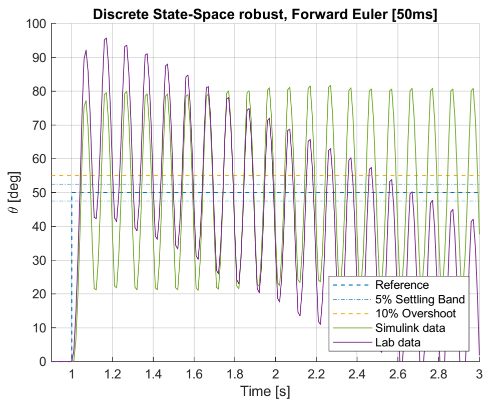
In these assignments we tested the discretized robust State-Space Controller using the different discretizations (Forward Euler, Backward Euler) and sampling times $T_s = (1ms, 10ms, 50ms)$ with a 50° step reference. Again, the used design parameters are summarized in table 4.



(a) 1ms



(b) 10ms



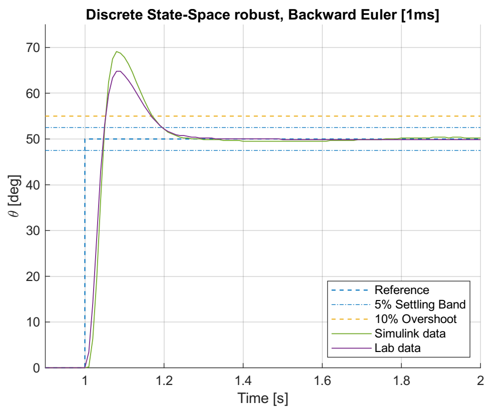
(c) 50ms

Figure 15: Step responses of 50° reference of the discretized robust State-Space controller using Forward Euler for $T_s = (1ms, 10ms, 50ms)$.

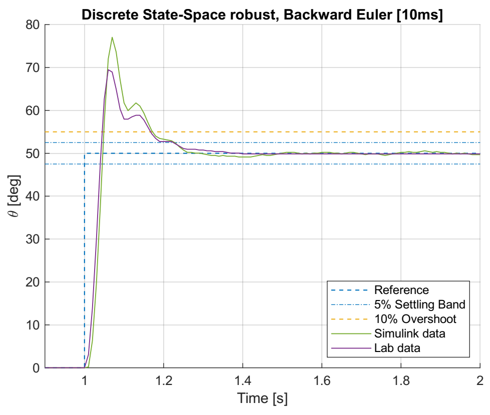
We expected to observe similar results as in the nominal case. Instead, we noticed two clear anomalies. The first one is that Forward Euler discretization at $T_s = 10ms$ is showing high frequency oscillations around the set point, which are visible when considering a longer time horizon than the one in the plot. The second one is that Backward Euler at $T_s = 50ms$ is showing high amplitude oscillations around the set point. From the previous results, we expected the Forward Euler discretization to show asymptotic tracking for $T_s = 10ms$ and the same for Backward Euler for $T_s = 50ms$. Still, the simulink model and the real system showed a similar qualitative behavior.

We hypothesized that the problem could be in the choice of the observer pole, leading to bad estimates of the state. Choosing a faster observer pole only worsened the behavior. We can also rule out that the problem comes from the choice of the controller gains, since they show good behavior in continuous time. Removing saturation non-linearities in the model, made the simulation results in the two anomalous cases unstable.

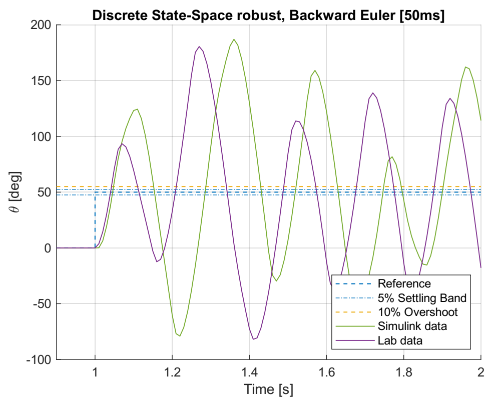
We do not have final conclusions on the cause, but we think it may be related to the delays introduced by the sampling architecture when T_s is large, that together with the integrator action in the robust design leads to these phenomena. The performance indeces are summarized in table 6.



(a) 1ms



(b) 10ms



(c) 50ms

Figure 16: Step responses of 50° reference of the discretized robust State-Space controller using Backward Euler for $T_s = (1ms, 10ms, 50ms)$.

Discretization method	System	$t_{s,5\%}$ $[ms]$	M_p $\%$	$t_{s,5\%}$ $[ms]$	M_p $\%$	$t_{s,5\%}$ $[ms]$	M_p $\%$
Sampling time		$1ms$		$10ms$		$50ms$	
Forward Euler	Simulink model	193	39.0	377	51.9	-	-
	Real plant	196	30.3	168	42.2	-	-
Backward Euler	Simulink model	196	38.2	228	54.1	-	-
	Real plant	196	29.6	224	39.0	-	-

Table 6: Performance indeces for the different discratizetions and different T_s with a 50° step reference (robust design)

2.4 Direct Design

2.4.1 Methodology

In this part, we design a discrete State-Space controllers using direct design. We first discretize the plant exactly with sampling time T_s, according to equation 7. The discretized system is described by the quadruple (Φ,Γ,H,J). Then we use nominal and robust State-Space control directly in discrete time. To reconstruct the state we use a one dimensional reduced order observer.

$$\Phi = e^{AT_s}, \qquad \Gamma = \int_0^{T_s} e^{A\tau} B d\tau, \qquad H = C, \qquad J = D \tag{7}$$

The resulting architecture is the same as for the State-Space emulation represented in 10. The design parameters are different.

The design parameters that we obtain following this approach are the matrices N_x, N_u, K, K_i, L, and the detailed procedure for their computation is described in Handout 1 for the controller parameters, and in Handout 2 for the reduced order observer gain L.

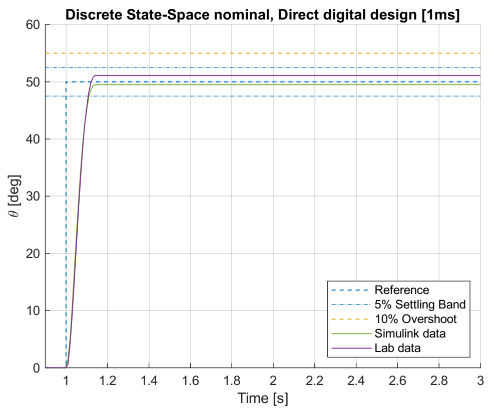
2.4.2 Assignments 1,2,3 - Direct Nominal Design

In these first assignments we tested the nominal approach with a step reference of 50° with sampling times $T_s = (1ms, 10ms, 50ms)$. The used design parameters are summarized in table 7, where the observer pole is chosen to be 5 times faster than the ones of the controlled system. The step responses are shown in figure 17.

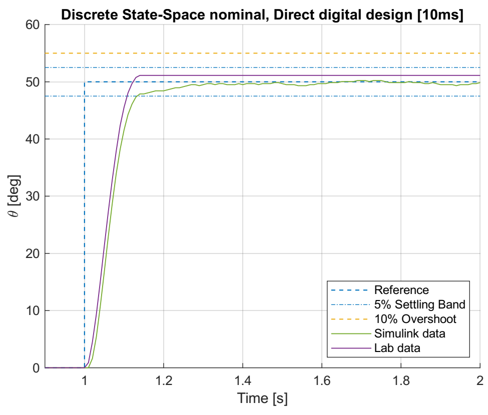
T_s	N_x	N_u	K	L
$1ms$	$\begin{bmatrix}1 & 0\end{bmatrix}^T$	0	$\begin{bmatrix}4.0975 & -0.0053\end{bmatrix}$	110.3090
$10ms$	$\begin{bmatrix}1 & 0\end{bmatrix}^T$	0	$\begin{bmatrix}4.0961 & 0.0131\end{bmatrix}$	59.2555
$50ms$	$\begin{bmatrix}1 & 0\end{bmatrix}^T$	0	$\begin{bmatrix}3.6726 & 0.0657\end{bmatrix}$	8.2878

Table 7: Direct design nominal state-space approach parameters

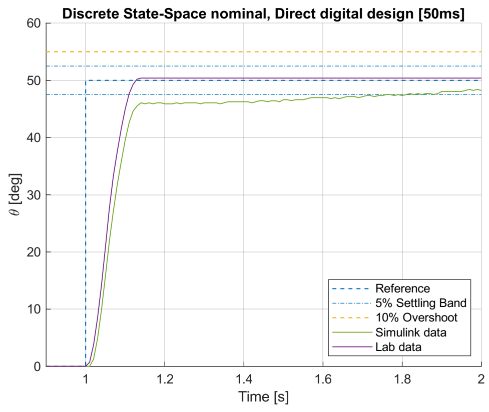
Unlike the emulation method, we observe that even for $T_s = 50ms$ we get tracking of the step reference with a constant tracking error and a reasonable performance. Therefore we see that direct design is much more robust with respect to emulation method when T_s gets larger. This is expected because emulation relies on approximations that gets less accurate when T_s gets larger, while direct design discretizes the plant exactly. The performance indeces are summarized in table 9.



(a) 1ms



(b) 10ms



(c) 50ms

Figure 17: Step responses of 50° reference of the discretized nominal State-Space controller using direct design for $T_s = (1ms, 10ms, 50ms)$

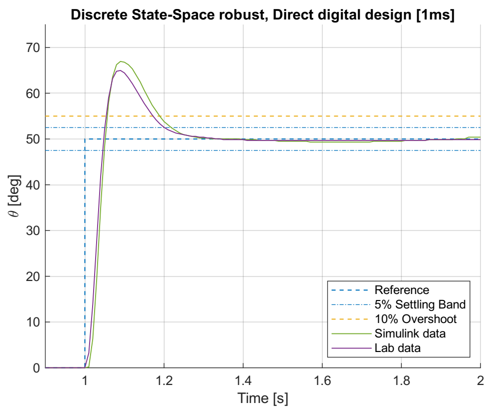
2.4.3 Assignments 4,5 - Robust

In these assignments we tested the robust approach, adding an integral action for the cumulative error, again with a step reference of 50° with sampling times $T_s = (1ms, 10ms, 50ms)$. The used design parameters are summarized in table 8, where the poles are chosen to be 5 times faster than the ones of the controlled system. The step responses are shown in figure 18.

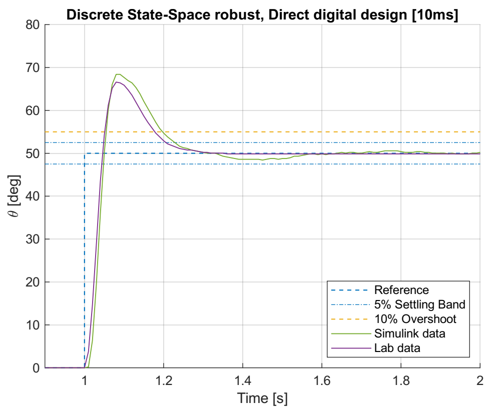
T_s	N_x	N_u	K	K_i	L
$1ms$	$\begin{bmatrix}1 & 0\end{bmatrix}^T$	0	$\begin{bmatrix}90.5096 & 0.3674\end{bmatrix}$	0.2444	110.3090
$10ms$	$\begin{bmatrix}1 & 0\end{bmatrix}^T$	0	$\begin{bmatrix}16.8123 & 0.3119\end{bmatrix}$	1.8823	59.2555
$50ms$	$\begin{bmatrix}1 & 0\end{bmatrix}^T$	0	$\begin{bmatrix}9.0086 & 0.1592\end{bmatrix}$	3.1892	8.2878

Table 8: Direct design robust state-space approach parameters

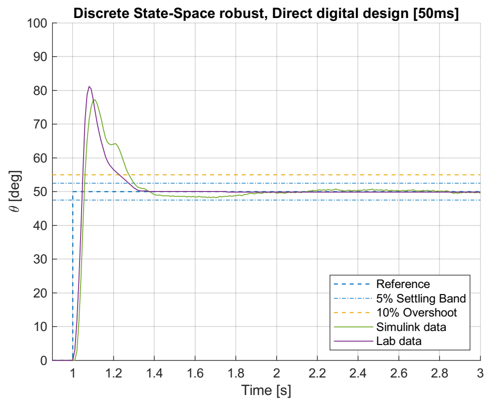
As for the nominal case, we see that the direct design is much more robust than corresponding emulation based design. As expected, with the robust design we have an overshoot but we reach asymptotic tracking without steady state error. As most of the times in this Laboratory, the simuliations and the real system data show similar qualitative behaviour. The performance indeces are summarized in table 9.



(a) 1ms



(b) 10ms



(c) 50ms

Figure 18: Step responses of 50° reference of the discretized robust State-Space controller using direct design for $T_s = (1ms, 10ms, 50ms)$

	System	$t_{s,5\%}$ $[ms]$	M_p $\%$	$t_{s,5\%}$ $[ms]$	M_p $\%$	$t_{s,5\%}$ $[ms]$	M_p $\%$
Sampling time		$1ms$		$10ms$		$50ms$	
Nominal	Simulink model	111	0.0	133	0.4	809	0.0
	Real plant	107	2.2	108	2.2	109	0.8
Robust	Simulink model	218	33.9	228	36.8	301	54.4
	Real plant	201	30.0	206	33.2	274	62.4

Table 9: Direct design performances with nominal and robust designs and different T_s with a 50° step reference

A Datasheet of DC servomotor with inertial load

```
%% General parameters and conversion gains
%   conversion gains
rpm2rads = 2*pi/60;                 %   [rpm]   —> [rad/s]
rads2rpm = 60/2/pi;                 %   [rad/s] —> [rpm]

rpm2degs = 360/60;                  %   [rpm]   —> [deg/s]
degs2rpm = 60/360;                  %   [deg/s] —> [rpm]

deg2rad = pi/180;                   %   [deg]   —> [rad]
rad2deg = 180/pi;                   %   [rad]   —> [deg]

ozin2Nm = 0.706e—2;                 %   [oz*inch] —> [N*m]
%% DC motor nominal parameters
%   brushed DC—motor Faulhaber 2338S006S
mot.R    = 2.6;                     %   armature resistance
mot.L    = 180e—6;                  %   armature inductance
mot.Kt   = 1.088 * ozin2Nm;         %   torque constant
mot.Ke   = 0.804e—3 * rads2rpm;     %   back—EMF constant
mot.J    = 5.523e—5 * ozin2Nm;      %   rotor inertia
mot.B    = 0.0;                     %   viscous friction coeff (n.a.)
mot.eta  = 0.69;                    %   motor efficiency
mot.PN   = 3.23/mot.eta;            %   nominal output power
mot.UN   = 6;                       %   nominal voltage
mot.IN   = mot.PN/mot.UN;           %   nominal current
mot.tauN = mot.Kt*mot.IN;           %   nominal torque
mot.taus = 2.42 * ozin2Nm;          %   stall torque
mot.w0   = 7200 * rpm2rads;         %   no—load speed

%% Gearbox nominal parameters
%   planetary gearbox Micromotor SA 23/1
gbox.N1   = 14;                     %   1st reduction ratio (planetary gearbox)
gbox.eta1 = 0.80;                   %   gearbox efficiency

%   external transmission gears
gbox.N2   = 1;                      %   2nd reduction ratio (external trasmission gears)
gbox.J72  = 1.4e—6;                 %   inertia of a single external 72 tooth gear
gbox.eta2 = 1;                      %   external trasmission efficiency (n.a.)

%   overall gearbox data
gbox.N   = gbox.N1*gbox.N2;         %   total reduction ratio
gbox.eta = gbox.eta1*gbox.eta2;     %   total efficiency
gbox.J   = 3*gbox.J72;              %   total inertia (at gearbox output)

%% Mechanical load nominal parameters
%   inertia disc params
mld.JD = 3e—5;                      %   load disc inertia
mld.BD = 0.0;                       %   load viscous coeff (n.a.)

%   overall mech load params
mld.J     = mld.JD + gbox.J;        %   total inertia
mld.B     = 2e—6;                   %   total viscous fric coeff (estimated)
mld.tausf = 1.0e—2;                 %   total static friction (estimated)

%% Voltage driver nominal parameters
%   op—amp circuit params
drv.R1 = 7.5e3;                     %   op—amp input resistor (dac to non—inverting in)
drv.R2 = 1.6e3;                     %   op—amp input resistor (non—inverting in to gnd)
drv.R3 = 1.2e3;                     %   op—amp feedback resistor (output to  inverting in)
drv.R4 = 0.5e3;                     %   op—amp feedback resistor (inverting in to gnd)
drv.C1 = 100e—9;                    %   op—amp input capacitor
drv.outmax = 12;                    %   op-amp max output voltage

%   voltage driver dc-gain
drv.dcgain = drv.R2/(drv.R1+drv.R2) * (1 + drv.R3/drv.R4);

%   voltage driver time constant
drv.Tc = drv.C1 * drv.R1*drv.R2/(drv.R1+drv.R2);

%% Sensors data
%   shunt resistor
sens.curr.Rs = 0.5;

%   Hewlett-Packard HEDS-5540#A06 optical encoder
sens.enc.ppr = 500*4;                           %   pulses per rotation
sens.enc.pulse2deg = 360/sens.enc.ppr;          %   [pulses] -> [deg]
sens.enc.pulse2rad = 2*pi/sens.enc.ppr;         %   [pulses] -> [rad]
sens.enc.deg2pulse = sens.enc.ppr/360;          %   [deg] -> [pulses]
sens.enc.rad2pulse = sens.enc.ppr/2/pi;         %   [rad] -> [pulses]

%   potentiometer 1 (Spectrol 138-0-0-103) - installed on motor box
sens.pot1.range.R      = 10e3;                                      %   ohmic value range
sens.pot1.range.V      = 5;                                         %   voltage range
sens.pot1.range.th_deg = 345;                                       %   angle range [deg]
sens.pot1.range.th     = sens.pot1.range.th_deg * deg2rad;          %   angle range [rad]
sens.pot1.deg2V        = sens.pot1.range.V / sens.pot1.range.th_deg;    %   sensitivity [V/deg]
sens.pot1.rad2V        = sens.pot1.range.V / sens.pot1.range.th;        %   sensitivity [V/rad]
sens.pot1.V2deg        = 1/sens.pot1.deg2V;                         %   conversion gain [V] -> [deg]
sens.pot1.V2rad        = 1/sens.pot1.rad2V;                         %   conversion gain [V] -> [rad]

%% Data acquisition board (daq) data
%   NI PCI-6221 DAC data
daq.dac.bits = 16;                              %   resolution (bits)
daq.dac.fs   = 10;                              %   full scale
daq.dac.q    = 2*daq.dac.fs/(2^daq.dac.bits-1);     %   quantization

%   NI PCI-6221 ADC data
daq.adc.bits = 16;                              %   resolution (bits)
daq.adc.fs   = 10;                              %   full scale (as set in SLDRT Analog Input block)
daq.adc.q    = 2*daq.adc.fs/(2^daq.adc.bits-1);     %   quantization

%% Sampling time
Ts = 1e-3;

%% Parameters estimated for the real motor in the lab
real.Jeq = 6.73e-7;
real.Beq = 1.95e-6;
real.tausf = 6.2e-3;
```

B Matlab code implementation

B.1 Discrete PID Design via Emulation

```
%% PID controllers emulation

% Define auxilliary parameters
paramP.Jeq = real.Jeq;
paramP.Beq = real.Beq;

% Define transfer function parameters
paramP.Req = mot.R + sens.curr.Rs;
paramP.km = mot.Kt*drv.dcgain/(paramP.Req*paramP.Beq+mot.Kt*mot.Ke);
paramP.Tm = paramP.Req*paramP.Jeq/(paramP.Req*paramP.Beq+mot.Kt*mot.Ke);

%% Define performace indexes

perf.ts5 = 0.2;
perf.mp = 0.1;

perf.d = log(1/perf.mp)/sqrt(pi^2+log(1/perf.mp)^2);
perf.wg = 3/(perf.d*perf.ts5);
perf.phi = 100*perf.d;              %[deg] approximation for phi<70 deg

%% Design controllers using different discretization for Ts = 1ms

Ts = 0.001;

z = tf('z',Ts);
% define transfer function with delay
Pdelay = exp(-s*Ts/2)*(paramP.km/(gbox.N*s*(paramP.Tm*s+1)));

% Get magnitue and phase at the desired frequency
[magPwg, phasePwg] = bode(Pdelay, perf.wg);

dK = abs(magPwg)^-1;
dPh = -pi + perf.phi*deg2rad - phasePwg*deg2rad;

% Estimation of the paramters of the PID-discrete time controller (system with time delay)
a = 4000;
pidz.K = dK * cos(dPh);

Td = (tan(dPh)+sqrt(tan(dPh)^2+4/a))/(2*perf.wg);
Ti = a*Td;
pidz.T_l = 1/(2*perf.wg);

pidz.Kd = pidz.K*Td;
pidz.Ki = pidz.K/Ti;
% Define discretization variables
s_forward = (z-1)/Ts;
s_backward = (z-1)/(Ts*z);

% Define PID controller continous transfer function
s = tf('s');
pidz.C = pidz.K + pidz.Ki*(1/s) + pidz.Kd * (s/(pidz.T_l*s + 1));

% Forward Euler Discretization
pidz.Cf = pidz.K + pidz.Ki*(1/s_forward) + pidz.Kd * (s_forward/(pidz.T_l*s_forward + 1));
% Backward Euler Discretization
pidz.Cb = pidz.K + pidz.Ki*(1/s_backward) + pidz.Kd * (s_backward/(pidz.T_l*s_backward + 1));
% Tustin/Trapezoidal Discretization
```

```
pidz.Ctust = c2d(pidz.C,Ts,'tustin');
% Exact Discretization
pidz.Czoh = c2d(pidz.C,Ts,'zoh');

% Transfer functions of controllers for different discretizations used in PID-emulation.
pidz.controllers = [pidz.Cf, pidz.Cb, pidz.Ctust, pidz.Czoh];

% get numerator and denominator for simulations
% in this case for Forward Euler
[pidz.Cnum, pidz.Cdenom] = tfdata(pidz.controllers(1), 'v');

%% Anti-Wind-UP

% define the anti-wind-up gain
pidz.Kw = 5/perf.ts5;
```

B.2 State Space Control Design via Emulation

```
%% State Space Controller emulation

% Define auxilliary parameters
paramP.Jeq = real.Jeq;
paramP.Beq = real.Beq;

% Define state space parameters
paramP.Req = mot.R + sens.curr.Rs;
paramP.km = mot.Kt*drv.dcgain/(paramP.Req*paramP.Beq+mot.Kt*mot.Ke);
paramP.Tm = paramP.Req*paramP.Jeq/(paramP.Req*paramP.Beq+mot.Kt*mot.Ke);

%% Define performance indexes for pole placement

perf.ts5 = 0.2;
perf.mp = 0.1;

perf.d = log(1/perf.mp)/sqrt(pi^2+log(1/perf.mp)^2);
perf.wg = 3/(perf.d*perf.ts5);
perf.phi = 100*perf.d;              %[deg] approximation for phi<70 deg

% Specifications for pole Placement
perf.sigmamin = 3/perf.ts5;
perf.wn = 3/(perf.ts5*perf.d);

%% Define continous State-Space for nominal controller (simplified plant model)

ssP.A = [0, 1; 0 -1/paramP.Tm];
ssP.B = [0; paramP.km/(gbox.N*paramP.Tm)];
ssP.C = [1 0];
ssP.D = 0;

% Calculate prefilter gains for both nominal and robust
ssNominal.N = [ssP.A ssP.B; ssP.C ssP.D]\[zeros(2,1); 1];
ssNominal.Nx = ssNominal.N(1:2);
ssNominal.Nu = ssNominal.N(3);

%% Place Poles for nominal controller

% Define poles postitions
ssNominal.poles = [-perf.d*perf.wn+1i*perf.wn*sqrt(1-perf.d^2), -perf.d*perf.wn-1i*perf.wn*sqrt(1-perf.d^2)];

```

```matlab
% Find gains for pole placement
ssNominal.K = place(ssP.A, ssP.B, ssNominal.poles);

%% Define extended continous State−Space for robust controller (simplified plant model)

ssP.Ae = [0, ssP.C; zeros(2,1), ssP.A];
ssP.Be = [0; ssP.B];

%% Place Poles for robust controller

% define auxilliary performance index
perf.wdmax = perf.sigmamin/perf.d * sqrt(1−perf.d^2);

% Define poles postitions
ssRobust.poles = [−2*perf.sigmamin + 1i*perf.wdmax, −2*perf.sigmamin − 1i*perf.wdmax, −3*perf.sigmamin];

% Find gains for pole placement
ssRobust.Ke = place(ssP.Ae, ssP.Be, ssRobust.poles);

ssRobust.Ki = ssRobust.Ke(1);
ssRobust.K = ssRobust.Ke(2:3);

%% Reduced order Observer
% defining contrinous reduced order observer
redObs.L = 4/paramP.Tm;                      % defyning observer gain manually
redObs.A0 = −(1/paramP.Tm + redObs.L);       % Manually placing EV to −5/paramP.Tm
redObs.B0 = [paramP.km/(gbox.N*paramP.Tm), −(1/paramP.Tm + redObs.L)*redObs.L];
redObs.C0 = [0; 1];
redObs.D0 = [0 1; 0 redObs.L];

% Define a sampling time
Ts=0.001;

% Forward Euler Discretization
redObs.Ph0 = 1+ redObs.A0*Ts;
redObs.Gamma0 = redObs.B0*Ts;
redObs.H0 = redObs.C0;
redObs.J0 = redObs.D0;

% Backward Euler Discretization
redObs.Ph0 = inv(1 − redObs.A0*Ts);
redObs.Gamma0 = (1 − redObs.A0*Ts)\redObs.B0*Ts;
redObs.H0 = redObs.C0/(1 − redObs.A0*Ts);
redObs.J0 = redObs.D0 + redObs.C0/(1 − redObs.A0*Ts)*redObs.B0*Ts;
```

B.3 State Space Control via Direct Design

```matlab
%% State Space Controller direct figital design

% Define auxilliary parameters
paramP.Jeq = real.Jeq;
paramP.Beq = real.Beq;

% Define state space parameters
paramP.Req = mot.R + sens.curr.Rs;
paramP.km = mot.Kt*drv.dcgain/(paramP.Req*paramP.Beq+mot.Kt*mot.Ke);
paramP.Tm = paramP.Req*paramP.Jeq/(paramP.Req*paramP.Beq+mot.Kt*mot.Ke);

%% Define performance indexes for pole placement
```

```matlab

perf.ts5 = 0.2;
perf.mp = 0.1;

perf.d = log(1/perf.mp)/sqrt(pi^2+log(1/perf.mp)^2);
perf.wg = 3/(perf.d*perf.ts5);
perf.phi = 100*perf.d;              %[deg] approximation for phi<70 deg

% Specifications for pole Placement
perf.sigmamin = 3/perf.ts5;
perf.wn = 3/(perf.ts5*perf.d);

%% Define continous State-Space for nominal controller (simplified plant model)

ssP.A = [0, 1; 0 -1/paramP.Tm];
ssP.B = [0; paramP.km/(gbox.N*paramP.Tm)];
ssP.C = [1 0];
ssP.D = 0;
ssNominalDT.SysCT = ss(ssP.A,ssP.B,ssP.C,ssP.D);

%% Discretize the model
% Set a sampling time
Ts=0.001;

% exact discretization of the plant
ssNominalDT.SysD = c2d(ssNominalDT.SysCT,Ts,'zoh');

% Calculate prefilter gains for both nominal and robust
ssNominalDT.N = [ssNominalDT.SysD.A-eye(2), ssNominalDT.SysD.B; ssNominalDT.SysD.C, ssNominalDT.SysD.D]\[zeros(2,1);
     1];
ssNominalDT.Nx = ssNominalDT.N(1:2);
ssNominalDT.Nu = ssNominalDT.N(3);

%% Place Poles for nominal controller

% Define poles postitions in continous time
ssNominal.poles = [-perf.d*perf.wn+1i*perf.wn*sqrt(1-perf.d^2), -perf.d*perf.wn-1i*perf.wn*sqrt(1-perf.d^2)];

% Translate poles in discrete time
ssNominalDT.poles = exp(ssNominal.poles*Ts);

% Find gains for pole placement
ssNominalDT.K = place(ssNominalDT.SysD.A, ssNominalDT.SysD.B, ssNominalDT.poles);

%% Define extended State-Space for robust controller (simplified plant model)

ssRobustDT.Phe = [1 ssRobustDT.SysD.C ;zeros(2, 1) ssRobustDT.SysD.A];
ssRobustDT.Gammae =[0; ssRobustDT.SysD.B];

%% Place Poles for robust controller

% Define poles postitions in continous time
ssRobust.poles = [-2*perf.sigmamin + 1i*perf.wdmax, -2*perf.sigmamin - 1i*perf.wdmax, -2*perf.sigmamin];

% Translate poles in discrete time
ssRobustDT.poles = exp(ssRobust.poles*Ts);

% Find gains for pole placement
ssRobustDT.Ke = place(ssRobustDT.Phe, ssRobustDT.Gammae, ssRobustDT.poles);

ssRobustDT.K = ssRobustDT.Ke(2:3);
ssRobustDT.Ki = ssRobustDT.Ke(1);

```

```
%% Reduced Order Observer

% Set poles 5 times faster than the faster set in the controller and discretize them
redObs.pole = exp(5*(-2*perf.sigmamin)*Ts);

% Placing observer poles
redObs.L = place(ssNominalDT.SysD.A(2,2)',ssNominalDT.SysD.A(1,2)',redObs.pole);

% Define Reduced order observer matrices
ssRobustDT.redObs.Ph0 = ssNominalDT.SysD.A(2,2) - redObs.L*ssNominalDT.SysD.A(1,2);
ssRobustDT.redObs.Gamma0 = [ssNominalDT.SysD.B(2)-redObs.L*ssNominalDT.SysD.B(1), redObs.Ph0*redObs.L+sssNominalDT.
     SysD.A(2,1)-redObs.L*ssNominalDT.SysD.A(1,1)];
ssRobustDT.redObs.C0 = [0; 1];
ssRobustDT.redObs.D0 = [0 1; 0 redObs.L];
```