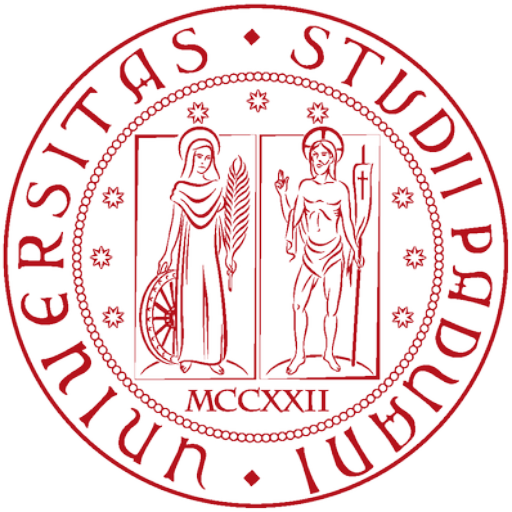
University of Padova

School of Engineering
Department of Information Engineering



Technical Report for

Control Engineering Laboratory 2

Group: 1
Shift: Friday

by

Bjørn Magnus Myrhaug - 2141821
Maximilian Pries - 2144162
Nihal Suri - 2141819

Submitted to

Prof. Dr. Francesco Ticozzi

Padova, June 19, 2025

Contents

1 Introduction

Precisely controlling DC servo motors is a crucial task for many applications, for example, the motors of a balancing robot. The Rotary Servo Base Unit SRV-02 from Quanser is used, which is an intuitive platform for rotary control experiments. The Rotary Servo Base Unit is a geared servo-mechanism system. The position of the load shaft is measured using a high-resolution optical encoder or a potentiometer. The encoder is also used to estimate the speed of the motor.

1.1 Activity Goal

The goal of the activity was to gain a deeper understanding of possible design strategies for digital controllers and their effect on the performance. The two design methodologies investigated are:

1. **Emulation**: Design continuous-time controllers and implement them using various emulation methods. The methods used are forward and backward Euler, Tustin, and exact (zero-order hold interpolation). The different controllers are:
   - PID controller with and without anti-windup.
   - Continuous-time position controllers designed using state-space techniques, with both nominal and robust tracking approaches.
2. **Direct design**: Design a discrete-time controller using a direct design approach. The system is first discretized, and then a discrete-time position controller is designed using state-space methods. Both nominal and robust tracking designs are implemented.

Another important aspect that was considered is the sampling time of the digital system. It has a major impact on performance and stability, and was considered for every designed controller. Contrary to our intuition, the experiments showed that decreasing the sampling time does not improve performance in every situation. Thus, the sampling time has to be chosen carefully, taking into account the design methods used and the system to be controlled.

1.2 System and Model

The system to be controlled consists of many single interacting parts, forming a complex, non-linear system. For the previous laboratory experiences, a detailed Simulink model for this system has been built, and its correctness has been validated. For most parts, the system's nominal parameters have been used in the model. However, due to wear and tear and misalignment, the system's viscous and static friction parameters B_{eq} and τ_{sf} and inertia J_{eq} are subject to change and have been

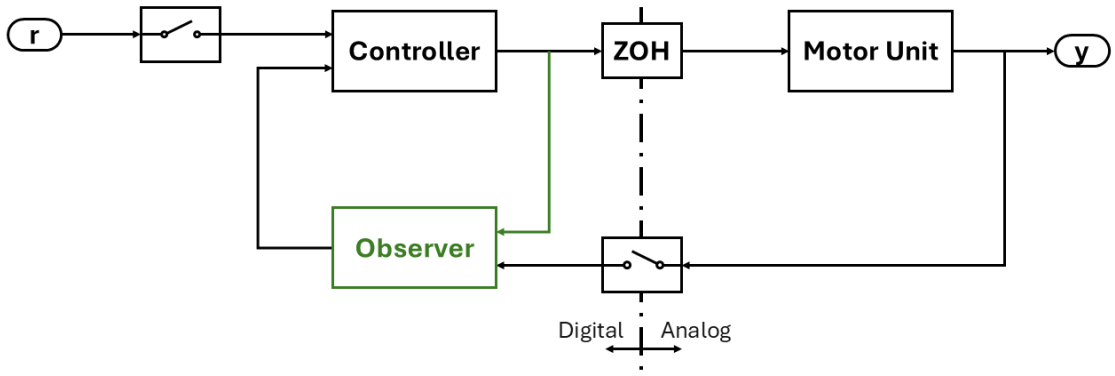


Figure 1.1: Block diagram of the closed-loop controlled system.

estimated:

$$B_{eq} = 2.56 \cdot 10^{-6} \ \frac{\text{Nm}}{\text{rad/s}} \qquad \tau_{sf} = 0.013 \text{ Nm} \qquad J_{eq} = 3.46 \ \frac{\text{kg}}{\text{m}^2}$$

The controllers were designed using these parameters, and they were incorporated into the Simulink model. This model has been used to test the designed controllers. For this, `Zero-Order-Hold` blocks have been used to implement ideal samplers before the control system and ideal holders before the plant input. A block diagram of the implementation is depicted in Figure 1.1. It shows the separation between the digital and analog settings. Besides the controller, it also includes an observer, which is used only for the state-space controllers to recover the unmeasured load's angular velocity.
For the design, the transfer function from the system's input voltage u to the load's angular position θ_l was reduced using the second-order dominant pole approximation:

$$P(s) = \frac{k_m}{T_m s + 1} \cdot \frac{1}{Ns} \tag{1.1}$$

where k_m and T_m are the motor transfer function gain and time constant, and depend on some motor intrinsic parameters, and N is the gear ratio from the motor shaft to the load. From the reduced transfer function, a state-space model to be used for the design of state-space controllers can be derived:

$$\Sigma : \begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{A}x + \boldsymbol{B}u \\ y = \boldsymbol{C}x + \boldsymbol{D}u \end{cases} \tag{1.2}$$

Where $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ are the state and input transition, output transition, and input feedthrough matrices, $\boldsymbol{x} = [\vartheta_l, \ \omega_l]^T$ is the statevector, containing the load's angular position and angular velocity, $y = \vartheta_l$ is the output and u is the input voltage applied to the motor. The numerical matrices can be found in Appendix A.1.

2 Tasks, Methodologies and Results

PID control and position state-space are implemented by emulation and then followed with, direct digital design of the position state-space controller.

2.1 Position PID-controller by Emulation

In this section, we design a sampled-data control system by emulation for the family of PID controllers. The control design is carried out by emulation, which means that it is first performed in the continuous-time domain, and then a discrete-time controller is built via a discretization procedure so as to approximate the response of the original system.

2.1.1 Backward Euler Discretization with Anti-windup

A discrete equivalent of the original continuous-time controller is obtained via the backward Euler method, which yields the following transfer function:

$$C(z) = K_D + K_I \frac{Tz}{z-1} + K_D \frac{z-1}{(T_L+T)\,z - T_L} \tag{2.1}$$

Where K_P, K_I and K_D are the proportional, integral, and derivative gains, and T_L is the time constant of the real derivative. The transfer function of the controller is obtained by using the `pid` function and exploiting the `IFormula` and `DFormula` properties, so as to obtain the desired discretization. The performance specifications on the overshoot M_p and the settling time $t_{s,5\%}$ are translated to requirements on the damping ratio δ and the natural frequency ω_n of the closed loop:

$$\delta = \frac{\log\left(\frac{1}{M_p}\right)}{\sqrt{\pi^2 + \left(\log\frac{1}{M_p}\right)^2}} \tag{2.2}$$

$$\omega_n = \frac{3}{\delta t_{s5\%}} \tag{2.3}$$

The gains are then derived by Bode's method, as done in the first laboratory, and some additional fine-tuning was done on top of it to obtain better transient responses. For the time constant T_L of the "real-derivative", as seen in equation 2.1, the value of $1/(2w_{gc})$ is used, where w_{gc} is the gain crossover frequency of the controlled system. An integrator anti-windup mechanism can also be added to the controller. It is crucial in our control scheme as we are dealing with saturations, as seen in Figure 2.1. When the output is saturated, the error doesn't go to zero, but the integrator (DTI) keeps accumulating error. This "windup" leads to a large integral component, resulting in excessive overshoot. Saturation effectively opens the loop. The anti-windup scheme that we create subtracts the difference between the actual (saturated) and desired (unsaturated) output back into the integrator to

unwind it. The calculation of the anti-windup gain K_W is done via a "rule-of-thumb" technique where its taken to be $4 \sim 5/t_{s,5\%}$. The complete set gains are as follows:

$$K_P = 7.845 \qquad K_I = 100.834 \qquad K_D = 0.076 \qquad T_L = 0.07 \qquad K_W = 30$$

.

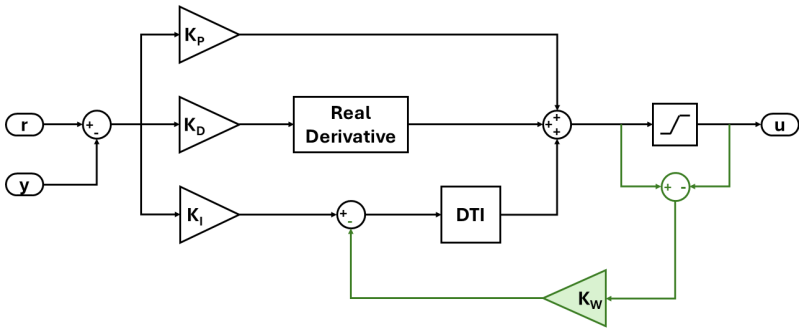
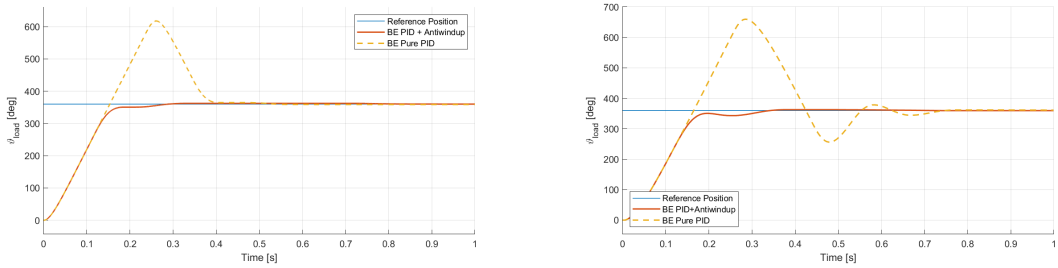


Figure 2.1: Emulated PID-controller scheme with anti-windup mechanism

The result of the discrete-time PID controller with and without the additional anti-windup mechanism via backward Euler emulation for a 360° step reference input for the sampling time $T_s = 10$ ms on the real system and model can be seen in Figure 2.2. The comparison of the control performances can also be observed in Table 2.1. Certain observations can be made after viewing the transient response of the real plant and the self-made model in both these conditions. As mentioned earlier, the addition of the anti-windup has a significant effect on the overshoot for both systems. For the real plant, we notice that it takes slightly more time to asymptotically track the step in comparison to the selfmade-model, this can be due to the fact that the self-made model is only an approximation of the plant. Note that the steady-state error e_{ss} refers to the relative error at the end of the observed time frame. In case of oscillations, it loses some meaning; however, it can still be used for comparison. This convention is applied throughout this report.

Table 2.1: Comparison of settling-time $t_{s,5\%}$[s], overshoot M_p[%] and steady-state error e_{ss}[%] for a step reference input of 360° emulated with backward Euler for T_s= 10 ms for a PID controller with and without anti-windup.

		M_p[%]	$t_{s,5\%}$ [s]	e_{ss}[%]
PID Backward Euler	self-made model	71.60	0.38	-0.4
	real plant	83.25	0.58	0.35
PID Backward Euler	self-made model	0.6	0.16	0.0
+ Anti-windup	real plant	0.75	0.17	-0.1



(a) Selfmade-model (b) Real-system

Figure 2.2: Comparison of the response of the closed-loop system to a 360° step reference input emulated with backward Euler for T_s= 10 ms for a PID controller with and without anti-windup.

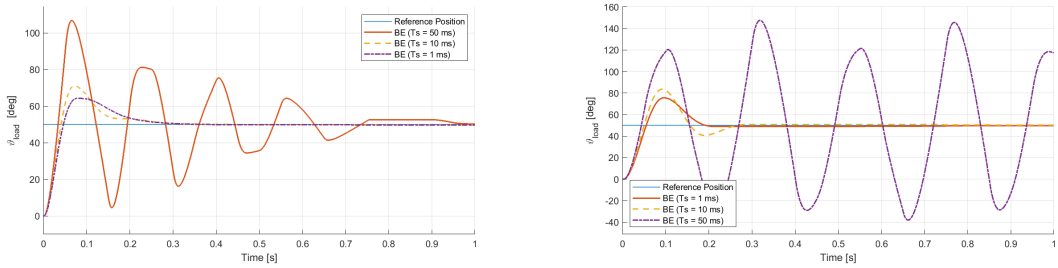
2.1.2 Comparison of Discretization Methods

The backward Euler method has already been introduced in Section 2.1. We will now build upon the same and introduce other discretization methods, which are the forward Euler, Tustin, and exact methods. The discrete controllers can be obtained by variable substitution. Table 2.2 gives an overview of the substituted transfer functions. For the implementation, MATLAB's built-in functions were used again.

Table 2.2: Variable substitution for discretizing a continuous-time transfer function C(s).

Forward Euler	Backward Euler	Tustin	Exact
$C\left(\frac{z-1}{T}\right)$	$C\left(\frac{z-1}{Tz}\right)$	$C\left(\frac{2}{T}\frac{z-1}{z+1}\right)$	$(1-z^{-1})\,\mathcal{Z}\left\{\frac{C(s)}{s}\right\}$

Each of the methods in Table 2.2 has its respective pros and cons. The backward Euler, Tustin, and exact methods always map the continuous-time poles inside the unit circle, but the forward Euler method can map them outside, leading to instability. The exact method has a higher computational cost in comparison to the other methods, as direct substitution is not used but the $\mathcal{Z}$ transform must be carried out between the convolution of the controller and the transfer function of the idealization of the DAC (Digital to Analog Converter) which is the (zero-order) holder (ZOH). The controller scheme for all the methods except the exact method follows the structure in Figure 2.1. For the exact method, we can directly use `c2d(sysC, T, 'zoh')`, where `sysC` is a continuous time `pid` object. Following this, a different scheme is considered, which consumes the MATLAB function `tfdata()` to return the numerator and denominator of the discretized transfer function.
The result of the discrete-time PID controller via backward Euler emulation for a 50° step reference input for various sampling times on the real system and model can be seen in Figure 2.3. The comparison of the corresponding control indices can be observed in Table 2.3. The same has been done for the other discretization methods, and the step responses and control performance can be viewed below.



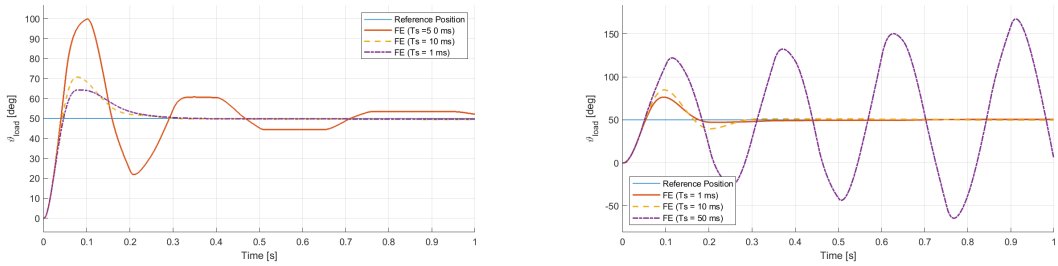
(a) Selfmade-model (b) Real-system

Figure 2.3: Response of the closed-loop system to a 50° step, with a PID controller emulated with backward Euler and T_s= 1 ms, 10 ms, 50 ms.

Table 2.3: Comparison of settling-time $t_{s,5\%}$[s], overshoot M_p[%] and steady-state error e_{ss}[%] for a step reference input of 50° emulated with backward Euler for T_s= 1 ms, 10 ms and 50 ms for a PID controller

Selfmade-model				Real-system			
T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	e_{ss}[%]	T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	e_{ss}[%]
1	28.52	0.2189	-0.64	1	51.20	0.1763	-0.64
10	42.20	0.225	-0.64	10	67.04	0.2439	0.44
50	113.48	0.9099	0.44	50	195.20	-	134

If we observe Figures 2.3, 2.4 and 2.5, it can be noticed that, in general, we have more overshoot in the real system, with similar settling times for the real system and and self-made model for 1 and 10 ms sampling times. A significant difference between the two is noticed for the 50 ms sampling time where the real-system fails to achieve asymptotic stability for any of the discretization methods, as the plant still might have some unmodeled dynamics leading to, too much of phase delay and a decrease in the phase margin takes place, leading to instability. The higher sampling time (less responsive controller) exposes this behaviour of the real system. As an additional task, we also tested the system using the exact or zero-order hold (ZOH) method for emulating $C(s)$; this method is mainly used to translate $P(s)$ to an exact discrete equivalent. The result of the discrete-time PID controller via ZOH (exact) emulation for a 50° step reference input for various sampling-times T_s on the real-system and model can be seen in Figure 2.6, and the comparison of the control performances for the ZOH method for several sampling times can be observed in Table 2.6. When performing the experiments in the laboratory, the set of gains designed for the controller with 10 ms sampling time was used for the controller with 50 ms sampling time; this was realized later on. The exact discretization method performed better than the other methods, in terms of overshoot and settling time in simulations. If the experiment had been performed correctly, we could have



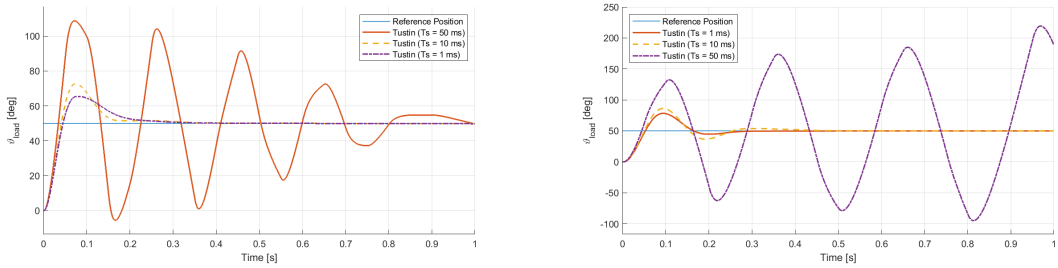
(a) Selfmade-model (b) Real-system

Figure 2.4: Response of the closed-loop system to a 50° step, with a PID controller emulated with forward Euler and T_s= 1 ms, 10 ms, 50 ms.

Table 2.4: Comparison of settling-time $t_{s,5\%}$[s], overshoot M_p[%] and steady-state error e_{ss}[%] for a step reference input of 50° emulated with forward Euler for T_s= 1 ms, 10 ms and 50 ms for a PID controller

Selfmade-model				Real-system			
T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	e_{ss}[%]	T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	e_{ss}[%]
1	28.52	0.2179	-0.64	1	52.64	0.256	1.16
10	41.48	0.1899	-1	10	69.56	0.266	-0.64
50	99.80	0.9899	4	50	234.80	-	-92

expected an asymptotically stable response. The MATLAB script implementing the calculations for this Section can be found in Appendix A.2.



(a) Selfmade-model (b) Real-system

Figure 2.5: Response of the closed-loop system to a 50° step, with a PID controller emulated with Tustin and T_s= 1 ms, 10 ms, 50 ms.

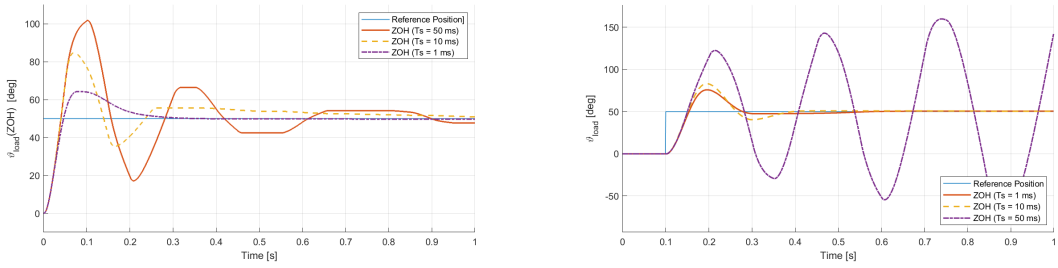
Table 2.5: Comparison of settling-time $t_{s,5\%}$[s], overshoot M_p[%] and steady-state error e_{ss}[%] for a step reference input of 50° emulated with Tustin for T_s= 1 ms, 10 ms and 50 ms for a PID controller

Selfmade-model				Real-system			
T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	e_{ss}[%]	T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	e_{ss}[%]
1	30.68	0.2019	-0.64	1	56.60	0.2479	-0.28
10	45.08	0.1559	-0.64	10	72.44	0.378	0.08
50	117.44	0.9529	-0.64	50	338.48	-	275

2.2 Position State-Space by Emulation

In this task, several discrete-time state-space controllers were implemented using static state feedback and reference feedforward to compute the control input, following both nominal and robust design approaches. Since only the load angular position θ_l is measured, a reduced-order state observer is employed to estimate the angular velocity ω_l. As a result, the combined controller-observer system realizes a dynamic output feedback. In this task, both the controller and observer were discretized using the emulation method. After the design, the control systems were tested by simulation using the mathematical model from Section 1.2, and a real motor.

The emulation method involves designing the regulator using established methods for the continuous-time regime and then discretizing it using variable substitution. For the controller, the only change is in the integrator, using the same method as in Section 2.1.2. The observer matrices are discretized as reported in Table 2.7. Here $\boldsymbol{\Phi}$, $\boldsymbol{\Gamma}$, $\boldsymbol{H}$ and $\boldsymbol{J}$ are the discretized state-space matrices. The MATLAB script implementing the calculations for this Section can be found in Appendix A.3.3.



(a) Selfmade-model (b) Real-system

Figure 2.6: Response of the closed-loop system to a 50° step, with a PID controller with ZOH emulation and T_s= 1 ms, 10 ms, 50 ms.

Table 2.6: Comparison of settling-time $t_{s,5\%}$[s], overshoot M_p[%] and steady-state error e_{ss}[%] for a step reference input of 50° emulated with ZOH for T_s= 1 ms, 10 ms and 50 ms for a PID controller

Selfmade-model				Real-system			
T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	e_{ss}[%]	T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	e_{ss}[%]
1	28.52	0.2179	-0.64	1	51.56	0.2683	1.16
10	69.56	0.7099	1.8	10	65.96	0.3659	0.8
50	103.76	0.8689	-4.6	50	219.68	-	189

2.2.1 Reduced Order Observer

The reduced-order observer is used to obtain an estimate of the load angular velocity from the load angular position and the control input. It is a dynamical system that internally simulates the model of the plant using the control input and corrects the estimate using the measured state. The advantage over full-order observers is that the measured state is fed through exactly and not estimated. The state-space representation of the observer is:

$$\hat{\Sigma}_o : \begin{cases} \dot{z} = A_o z + \boldsymbol{B}_o[u,\ y]^T \\ \hat{\boldsymbol{x}} = \boldsymbol{C}_o z + \boldsymbol{D}_o[u,\ y]^T \end{cases} \tag{2.4}$$

where

$$A_o = A_{2,2} - LA_{1,2} \qquad \boldsymbol{B}_o = \begin{bmatrix} B_2 - LB_1 & A_o + A_{2,1} - LA_{11} \end{bmatrix} \tag{2.5}$$

$$\boldsymbol{C}_o = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \boldsymbol{D}_o = \begin{bmatrix} 0 & 1 \\ 0 & L \end{bmatrix} \tag{2.6}$$

where the subscript o denotes the observer matrices, z is the observer state, and $A_{i,j}$ and B_j are the i,j-th and j-th sub-blocks of $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively. The observer feedback gain L is determined using the eigenvalue placement method. Since the

Table 2.7: Discretization of continuous time state-space matrices.

	Forward Euler	Backward Euler	Trapezoidal
$\boldsymbol{\Phi}$	$\boldsymbol{I} + \boldsymbol{A}T_s$	$(\boldsymbol{I} - \boldsymbol{A}T_s)^{-1}$	$\left(\boldsymbol{I} + \frac{\boldsymbol{A}T_s}{2}\right)\left(\boldsymbol{I} - \frac{\boldsymbol{A}T_s}{2}\right)^{-1}$
$\boldsymbol{\Gamma}$	$\boldsymbol{B}T_s$	$(\boldsymbol{I} - \boldsymbol{A}T_s)^{-1}\boldsymbol{B}T_s$	$(\boldsymbol{I} - \boldsymbol{A}T_s)^{-1}\boldsymbol{B}\sqrt{T_s}$
$\boldsymbol{H}$	$\boldsymbol{C}$	$\boldsymbol{C}(\boldsymbol{I} - \boldsymbol{A}T_s)^{-1}$	$\sqrt{T_s}\boldsymbol{C}(\boldsymbol{I} - \boldsymbol{A}T_s)^{-1}$
$\boldsymbol{J}$	$\boldsymbol{D}$	$\boldsymbol{D} + \boldsymbol{C}(\boldsymbol{I} - \boldsymbol{A}T_s)^{-1}\boldsymbol{B}T_s$	$\boldsymbol{D} + \boldsymbol{C}(\boldsymbol{I} - \frac{\boldsymbol{A}T_s}{2})^{-1}\frac{\boldsymbol{B}T_s}{2}$

state-space model of the motor is observable, the eigenvalue of the observer can be placed arbitrarily. It is chosen to be five times faster than the fastest closed-loop eigenvalue of the controlled system (see Equation 2.10), as:

$$\lambda_o = 5 \cdot \mathrm{Re}(\lambda_{1,2}) \tag{2.7}$$

Based on the eigenvalue, L is computed using MATLAB's `place` function. The resulting observer matrices are then discretized using the formulas reported in Table 2.7 and implemented in Simulink as a `Discrete State-Space` block. In the laboratory experiments, only the forward Euler method was used for discretization. The other methods are not physically implementable, as they result in non-zero entries corresponding to the input u in the feedforward matrix $\boldsymbol{J}$. This would result in an algebraic loop where the estimated state $\hat{\boldsymbol{x}}[k]$ depends on the control input at $u[k]$, which in turn is computed using $\hat{\boldsymbol{x}}[k]$. In non-real-time simulations, Simulink can handle algebraic loops, albeit more slowly. Therefore, in the simulations, the other methods were also compared. The observer gain is:

$$L = 37.6727$$

The resulting observer matrices for the forward Euler method can be found in Appendix A.3.1. The effectiveness of the observer has been validated in simulations, using continuous-time controllers at different setpoints. The results can be seen in Appendix A.3.2.

2.2.2 Nominal State-Space Controller

The nominal state-space controller consists of a state feedback and reference feedforward, with the following control law:

$$u[k] = -\boldsymbol{K}\boldsymbol{x}[k] + (N_u + \boldsymbol{N}_x\boldsymbol{K})r[k] \tag{2.8}$$

The block scheme implementing this is shown in Figure 2.7 (only the non-green-highlighted part). As actual controller input, the estimated state $\hat{\boldsymbol{x}}[k]$ is used, but thanks to the separation principle, the controller can be designed independently from the observer using the actual state.

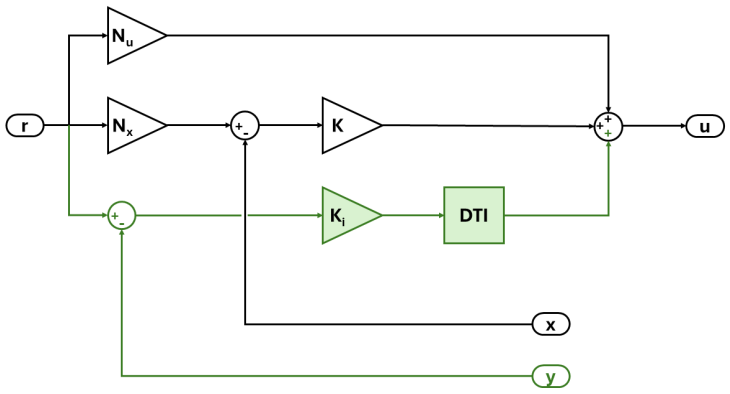


Figure 2.7: Block scheme of the state-space controller. The added integrator is highlighted in green.

The feedforward gains N_u and $\boldsymbol{N}_x$ were determined by the steady-state condition in continuous time:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ N_u \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix} \tag{2.9}$$

Since the state-space representation of the motor is reachable, all eigenvalues of the closed-loop system can be allocated freely by an appropriate feedback matrix $\boldsymbol{K}$. The desired eigenvalues were chosen based on the dominant pole approximation. For this, the performance specifications are again translated into damping and natural frequency by equations 2.2 and 2.3. From this, the desired dominant poles can be determined:

$$\lambda_{1,2} = -\delta\omega_n \pm j\omega_n\sqrt{1-\delta^2} \tag{2.10}$$

These coincide with the poles of the closed-loop state matrix $\boldsymbol{A}-\boldsymbol{B}\boldsymbol{K}$. The feedback matrix, placing these poles, can again be determined using the `place` function. The resulting gain matrices are:

$$\boldsymbol{N}_x = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad N_u = 0 \qquad \boldsymbol{K} = \begin{bmatrix} 3.7474 & -0.0731 \end{bmatrix}$$

The results of the discretization were compared with those of a continuous-time controller and among the different methods. Figure 2.8 shows the simulated response of the model to a 50° step reference with the continuous-time controller and the discrete controllers with a sampling time of $T_s = 1$ ms. It can be seen that the continuous-time controller achieves a smaller steady-state error with a slightly higher overshoot than the discrete-time controllers, which provide nearly identical responses. The continuous-time controller was also tested in simulations for step

references of different setpoints; the responses can be found in Appendix A.3.2. Figure 2.9 shows the comparison of the responses to the same input for the different emulation methods and sampling time $T_s = 10$ ms (a) and $T_s = 50$ ms (b). At small sampling times, the methods yield very similar results, differing only slightly in overshoot and steady-state error. For larger sampling times, the responses show drastically different results. While the backward Euler and Tustin methods still stabilize the system, the forward Euler method shows strong oscillations of period 50 ms, driving the input from one saturation to the other. The exact performance parameters of the simulated step responses are collected in Appendix A.3.2.

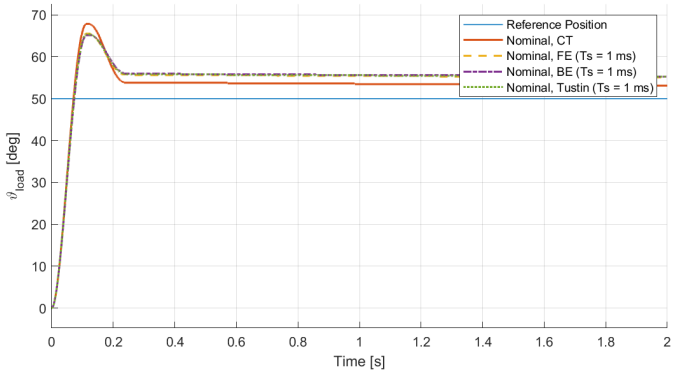
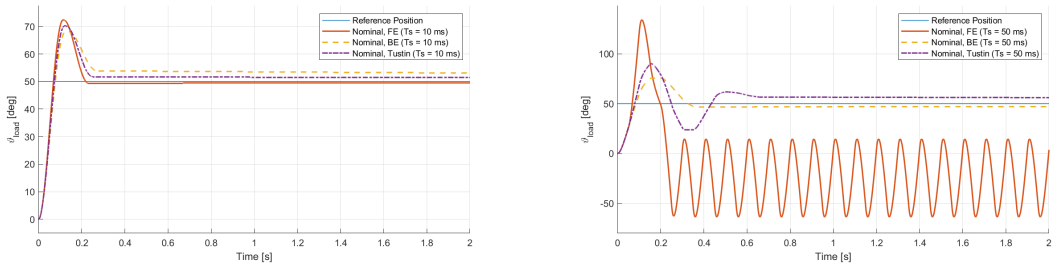


Figure 2.8: Response of the simulated model to a 50° step reference, comparison nominal continuous state-space controller vs. discretized controllers with $T_s = 1$ ms.

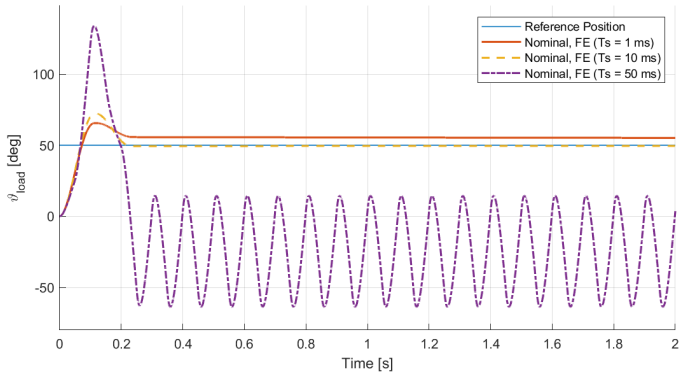


(a) $T_s = 10$ ms (b) $T_s = 50$ ms

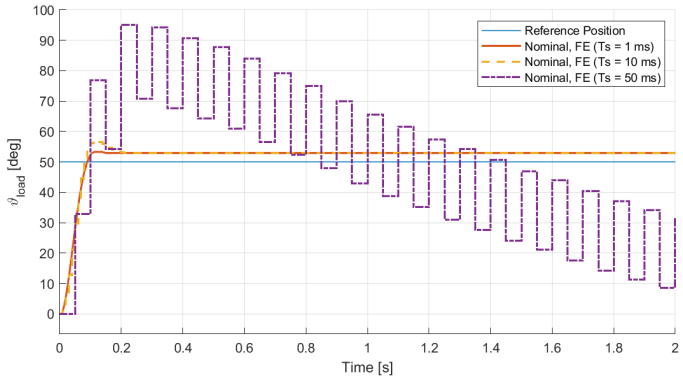
Figure 2.9: Response of the simulated model to a 50° step reference, comparison of nominal controllers, using different discretization methods, at the same sampling time.

Figure 2.10 (a) shows the simulated responses to to the 50° step reference using the forward Euler method at different sampling times and Figure 2.10 (b) shows the responses of the same controllers employed on the real motor. The responses are very similar, with just less overshoot on the real motor for $T_s = 1$ ms and 10 ms, and differing slightly for the 50 ms response, still showing similar unstable behavior. The exact data for the responses of the motor and the simulation using the forward

Euler discretization are compared in Table 2.8. They are also compared to the performance of the robust discrete controllers that are described in Section 2.2.3.



(a) Simulation on model



(b) Test on real motor

Figure 2.10: Response to a 50° step reference for nominal state-space controllers obtained with the forward Euler method with different sampling times, comparison between the model simulation and the real motor.

2.2.3 Robust State-Space Controller

The robust state-space controller is implemented as in Figure 2.7, this time including the green-highlighted part. Here, DTI stands for the discrete time integrator. The control law from Equation 2.8 is extended by the integrator state x_I multiplied by an integrator gain K_I:

$$u[k] = -\boldsymbol{K}\boldsymbol{x}[k] + (N_u + \boldsymbol{N}_x\boldsymbol{K})r[k] - K_I x_I[k] \tag{2.11}$$

The integrator state describes the discretely integrated output error:

$$x_I[k] = x_I[k-1] + T_s(y[k-1] - r[k-1]) \tag{2.12}$$

For the continuous-time design, the state-space model is extended by the integrator state as follows:

$$\begin{bmatrix} \dot{x}_I(t) \\ \dot{\boldsymbol{x}}(t) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{A} \end{bmatrix}}_{:=\boldsymbol{A}_e} \begin{bmatrix} x_I(t) \\ \boldsymbol{x}(t) \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \boldsymbol{B} \end{bmatrix}}_{:=\boldsymbol{B}_e} u(t) - \begin{bmatrix} 1 \\ \boldsymbol{0} \end{bmatrix} r(t) \tag{2.13}$$

With the extended matrices, the extended feedback matrix $\boldsymbol{K}_e$ can be computed using the `place` function. The eigenvalue corresponding to the integrator state is chosen as the real part of the other two eigenvalues. The matrix can then be partitioned as:

$$\boldsymbol{K}_e = [K_I \mid \boldsymbol{K}] \tag{2.14}$$

The feedforward gains stay identical to the nominal case. The resulting feedback matrices are:

$$K_I = 74.9486 \qquad \boldsymbol{K} = \begin{bmatrix} 6.3666 & -0.0076 \end{bmatrix} \tag{2.15}$$

The tests to validate the design were done exactly the same as in the nominal case. Figure 2.11 compares the simulated responses of the continuous-time robust controller to the discretized versions using different methods at $T_s = 1$ ms. Figure 2.12 shows the simulated responses for the discretized controllers at $T_s = 10$ ms (a) and $T_s = 50$ ms (b). For sampling times ≤ 10 ms, the methods show virtually no difference. At $T_s = 50$ ms, the forward Euler method results in an unstable response with oscillations of the same period as the sampling time, same as in the nominal case. The exact performance indices are listed in Appendix A.3.2.

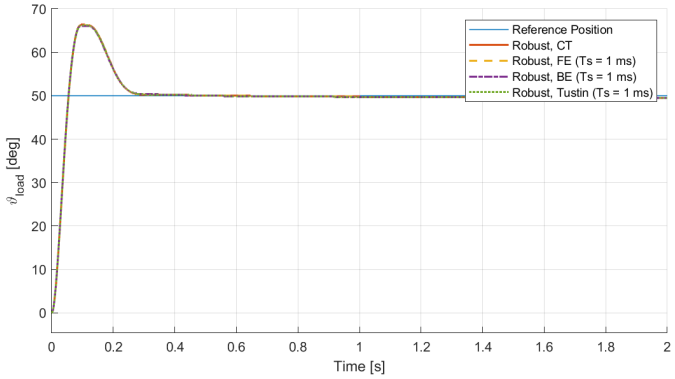
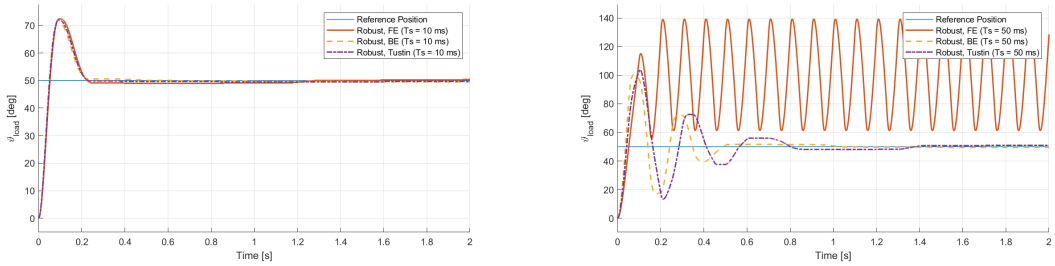


Figure 2.11: Response of the simulated model to a 50° step reference, comparison robust continuous state-space controller vs. discretized controllers with $T_s = 1$ ms.

Figure 2.13 compares again the simulation results of the forward Euler method (a) to the ones obtained from the real motor (b). At small sampling times, the overshoot increased slightly on the real motor, and some undershoot is also present, leading to



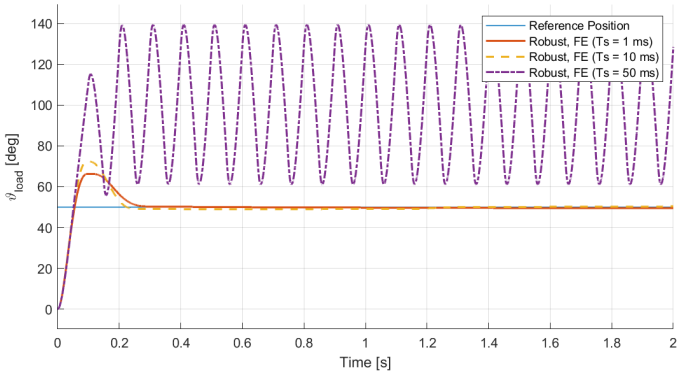
(a) $T_s = 10$ ms (b) $T_s = 50$ ms

Figure 2.12: Response of the simulated model to a 50° step reference, comparison of robust controllers, using different discretization methods, at the same sampling time.

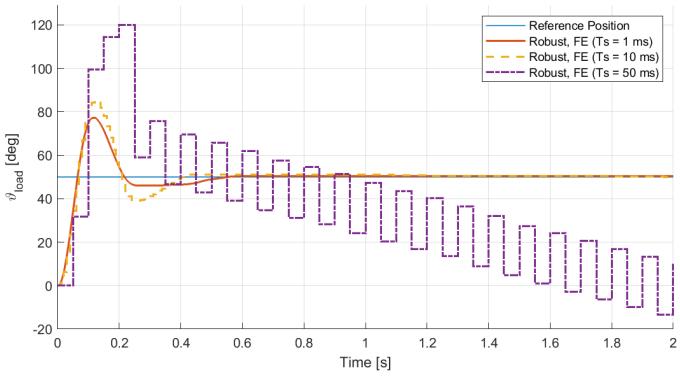
higher settling times. At $T_s = 50$ ms, the responses differ similarly as described for the nominal controllers. These differences are likely due to unmodeled nonlinearities. Table 2.8 collects all performance indices for the nominal and robust case for both simulations and the real motor for the forward Euler method.

Table 2.8: Comparison of performance indices to a 50° step reference, among simulation and the real motor, and nominal and robust controllers obtained by the forward Euler discretization method.

		Nominal		Robust	
		Simulation	Motor	Simulation	Motor
$T_s = 1$ ms	M_p [%]	31.04	6.56	32.48	54.44
	$t_{s,5\%}$ [s]	-	-	0.231	0.454
	e_{ss} [%]	10.16	5.84	-1.00	0.80
$T_s = 10$ ms	M_p [%]	44.74	13.76	44.72	70.28
	$t_{s,5\%}$ [s]	0.202	-	0.203	0.372
	e_{ss} [%]	-1.00	5.84	0.80	0.08
$T_s = 50$ ms	M_p [%]	168.2	90.08	178.64	139.76
	$t_{s,5\%}$ [s]	-	-	-	-
	e_{ss} [%]	-92.80	-37.00	157.04	-79.84



(a) Simulation on model



(b) Test on real motor

Figure 2.13: Response to a 50° step reference for robust state-space controllers obtained with the forward Euler method with different sampling times, comparison between the model simulation and the real motor.

2.3 Direct Digital Design

Instead of emulating a continuous-time controller, the control system can be designed directly in discrete time. The reduced state-space model is mapped into the discrete-time domain using an exact method, getting the zero-order hold (ZOH) equivalent of the continuous-time system.

$$\Sigma = (\mathbf{A}, \mathbf{B}, \mathbf{C}, D) \longrightarrow \Sigma_d = (\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{H}, J) \tag{2.16}$$

The discrete-time state-space matrices are computed as follows:

$$\mathbf{\Phi} = e^{\mathbf{A}T_s}, \qquad \mathbf{\Gamma} = \int_0^{T_s} e^{\mathbf{A}\eta}\mathbf{B}\, d\eta, \qquad \mathbf{H} = \mathbf{C}, \qquad J = D \tag{2.17}$$

The acquired matrices are used for the design of the two types of controllers, nominal and robust. The calculations of the plant matrices for the respective sampling times can be found in Appendix A.4.1, and the MATLAB script implementing it is in Appendix A.4.3.

2.3.1 Reduced Order Observer

The observer is done the same way as for the emulation method in Equation 2.6, just using the discrete state matrices and a pole in the z-plane. They can be found in Appendix A.4.2.

2.3.2 Nominal State-Space Controller

The nominal controller is done in a similar way as in the continuous case seen in 2.2.2. The difference is in the calculation of the gains $\mathbf{N_x}$ and N_u. The desired poles of the feedback system also need to be transformed into discrete-time. The poles get mapped from s-plane (continuous-time) to z-plane(discrete-time) with:

$$z = e^{sT_s} \tag{2.18}$$

The poles coincide with the poles of the closed-loop state matrix $\mathbf{\Phi} - \mathbf{\Gamma}\boldsymbol{K}$. The feedback matrix, placing these poles, can again be determined using the `place` function. The resulting feedforward gain matrices are calculated as follows:

$$\begin{bmatrix} \mathbf{\Phi} - \mathbf{I} & \mathbf{\Gamma} \\ \mathbf{H} & J \end{bmatrix} \begin{bmatrix} \mathbf{N_x} \\ N_u \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{2.19}$$

$$\boldsymbol{N}_x = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad\qquad N_u = 0 \tag{2.20}$$

Table 2.9 shows the state feedback gains $\boldsymbol{K}$ obtained through discrete-time pole placement for different sampling times. As T_s increases, the discrete-time poles move closer to the unit circle, resulting in slower system dynamics. To preserve the desired closed-loop performance, the feedback gains $\boldsymbol{K}$ must increase. This trend is seen in Table 2.9, where larger T_s values correspond to higher feedback gains.

Table 2.9: Different gains for the different sampling times for the nominal state-space controller from the direct digital design.

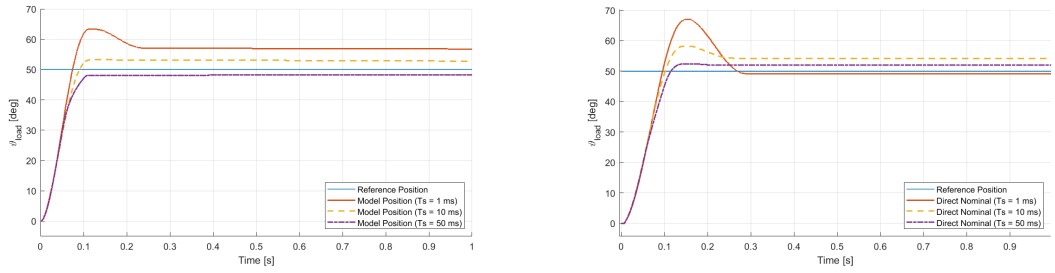
T_s [ms]	$\boldsymbol{K}$
1	$\begin{bmatrix} 3.7888 & -0.0698 \end{bmatrix}$
10	$\begin{bmatrix} 4.1112 & -0.0406 \end{bmatrix}$
50	$\begin{bmatrix} 4.2044 & 0.0383 \end{bmatrix}$

The results of the step responses are shown in Table 2.10, for both the self-made model and the real-time system. Figure 2.14 shows the responses for different sampling times.

Table 2.10: Performance indices from the step response with different sampling times using the nominal state-space controller from the direct digital design.

Selfmade-model				Real-system			
T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	e_{ss}[%]	T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	e_{ss}[%]
1	26.72	-	13.76	1	33.92	0.348	1.72
10	6.56	3.750	5.84	10	16.28	-	8.36
50	0	0.2020	3.52	50	4.76	0.206	4.04

As the sampling time T_s increases, the overshoot M_p consistently decreases in both models. This behavior is likely due to the reduced responsiveness of the discrete-time controller with larger sampling intervals, which results in a lower control action. While a smaller T_s allows faster reaction and shorter rise time, it also introduces a higher overshoot. For T_s=50 ms, the system becomes slower but more damped, leading to minimal or no overshoot.
With smaller sampling times, the controller gains are lower, leading to less aggressive control action in response to errors. This results in larger steady-state errors with lower T_s, when tracking step inputs as in the response for the self-made model. This behavior is not observed in the real-time system. For T_s= 1 ms, although the overshoot is higher, indicating a more oscillatory and potentially slower transient the steady-state error is smaller. This suggests that system nonlinearities in the real plant may assist in reducing the steady-state error, despite the lower feedback gain.



(a) Simulation on model

(b) Test on real motor

Figure 2.14: Response to a 50° step reference using the nominal state-space controller from the direct digital design, with different sampling times. Comparison between the model simulation and the real motor.

2.3.3 Robust State-Space Controller

To get rid of the steady-state error, a robust state-space controller is needed. The controller is done similar as in section 2.2.3. Except for a change in the Φ_e matrix. The discrete extended system is shown in Equations 2.21. The gains $\mathbf{N_x}$ and N_u are the same as in the nominal case.

$$\Sigma_e = \underbrace{\begin{bmatrix} \mathbf{1} & \boldsymbol{H} \\ 0 & \boldsymbol{\Phi} \end{bmatrix}}_{:=\boldsymbol{\Phi}_e} \begin{bmatrix} x_I[k] \\ x[k] \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \boldsymbol{\Gamma} \end{bmatrix}}_{:=\boldsymbol{\Gamma}_e} u[k] - \begin{bmatrix} 1 \\ 0 \end{bmatrix} r[k] \tag{2.21}$$

Table 2.11 shows the state feedback gains $\boldsymbol{K}$ obtained through discrete-time pole placement for different sampling times T_s. As T_s increases, the gains also increase the same as in the nominal case described earlier.

Table 2.11: Different gains for the different sampling times for the robust state-space controller from the direct digital design.

T_s [ms]	K_i	$\boldsymbol{K}$
1	0.0750	$\begin{bmatrix} 6.4479 & -0.0042 \end{bmatrix}$
10	0.7452	$\begin{bmatrix} 7.0746 & 0.0228 \end{bmatrix}$
50	2.6577	$\begin{bmatrix} 7.2681 & 0.0621 \end{bmatrix}$

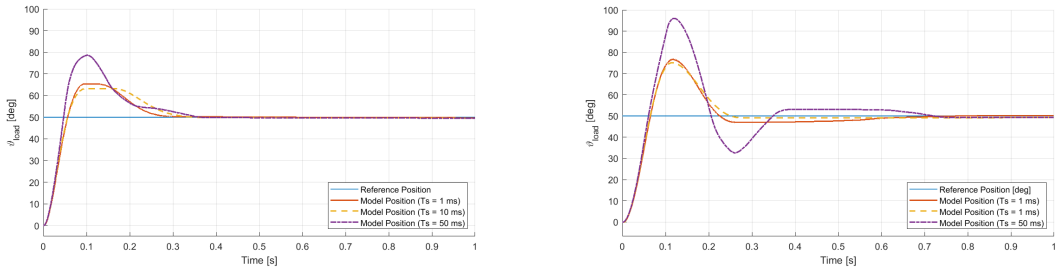
The results of the step responses are shown in Table 2.12, for both the self-made model and the real-time system. Figure 2.15 shows the responses for different sampling times.

Table 2.12: Performance indices from the step response with different sampling times using the robust state-space controller from the direct digital design.

Selfmade-model			Real-system		
T_s[ms]	M_p[%]	$t_{s,5\%}$[s]	T_s[ms]	M_p[%]	$t_{s,5\%}$[s]
1	30.68	0.240	1	53.36	0.445
10	26.36	0.279	10	50.12	0.228
50	57.32	0.302	50	91.88	0.631

For $T_s = 1$ ms and $T_s = 10$ ms, the overshoot is similar, with the 10 ms case actually showing slightly less overshoot than the 1 ms case. This suggests that the typical trend of increased overshoot with smaller sampling times as observed in the nominal controller, does not hold in the robust tracking case. Notably, the overshoot is higher in general compared to the nominal tracking setup. This is likely due to the increased overall gain, resulting from both the state feedback and the added contribution of the integrator. With smaller T_s the controller can react more quickly to accumulating tracking error, which can help reduce overshoot in some cases. The inclusion of integral action also enables the system to track step references asymptotically, effectively eliminating steady-state error.

For the self-made model, the settling time $t_{s,5\%}$ decreases when integral action is added, as expected. However, this behavior is not reflected in the response of the real motor. In that case, the settling time $t_{s,5\%}$ increases, most likely due to the higher overshoot introduced by the increased gain from both the state feedback and the integrator, which extends the time it takes for the system to settle. The real-time response shows a higher overshoot compared to the self-made model, consistent with the nominal case. For a sampling time of T_S=50 ms, the increased overshoot leads to noticeable oscillations in the system response, which in turn increases the overall settling time.



(a) Simulation on model

(b) Test on real motor

Figure 2.15: Response to a 50° step reference for the robust state-space controller using the state-space controller from the direct digital design, with different sampling times. Comparison between the model simulation and the real motor.

A Appendix

A.1 State-Space Matrices

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 0 & -62.3273 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 0 \\ 305.4383 \end{bmatrix}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad D = 0$$

A.2 PID by Emulation

Code 1: script_21_startup.m

```
% Matlab script to start the Simulink simulation of an accurate model of
% the Quanser SRV-02 + NI DAQ with a discrete time PID controller and its
% improvements such as antiwindup, and other discretization methods.
clear

%% Load Predefined Parameters
load_params_inertial_case

%% User Inputs

% Actual Parameters (estimated from Motor 1)
mld.Beq = 2.5663e-6;     % [Nm/(rad/s)]
mld.tausf = 0.013;       % [Nm]
mld.Jeq = 3.4640e-07;    % [kg m^2]

% Desired specifications
% Overshoot
specs.mp = 0.1;
% Settling Time
specs.settling_time = 0.15; %[s]

%% PID Parameters
% reduced plant transfer function
plant.km = (drv.dcgain*mot.Kt)/((mot.Req*mld.Beq) + (mot.Kt*mot.Ke));
plant.Tm = (mot.Req*mld.Jeq)/((mot.Req*mld.Beq) + (mot.Kt*mot.Ke));
plant.Ps = tf(plant.km, [(gbox.N*plant.Tm) gbox.N 0]);

% resulting gains from bode method
%PID = computePIDGains(8, specs.settling_time, specs.mp, plant.Ps, "PID");
PID.Kp = 7.845;
PID.Ki = 100.8347;
PID.Kd = 0.0763;
PID.Tl = 700e-04;
PID.Cs = pid(PID.Kp, PID.Ki, PID.Kd, PID.Tl);

% Anit Windup
PID.t_s5 = 0.15; % 5% settling time from lab0
PID.Kw = 1/(PID.t_s5/4.5); % anit windup gain: 1/Tw, Tw=t_s5/5

```

```
40
41  %% Simulation Parameters
42  % Sampling Time Vector [1 ms 10ms 50ms]
43  specs.Ts = [1e-3; 1e-2; 5e-2]; % [s]
44  % Step reference input
45  sIn.position = [50];
46  sIn.simulation_time = 5;
47
48
49  %% Discrete-Time PID
50  % Define discretization methods and their corresponding labels
51  methods = {'BackwardEuler', 'ForwardEuler', 'Trapezoidal'};
52  methodLabels = {'BE', 'FE', 'Tustin', 'zoh'};
53
54  % Initialize storage structures
55  for m = 1:length(methods)
56      label = methodLabels{m};
57      PID.Discrete.(label).Cz = cell(length(specs.Ts), 1);
58      PID.Discrete.(label).num = cell(length(specs.Ts), 1);
59      PID.Discrete.(label).den = cell(length(specs.Ts), 1);
60  end
61
62  % Loop over each sample time and discretization method
63  for i = 1:length(specs.Ts)
64      Ts = specs.Ts(i);
65
66      for m = 1:length(methods)
67          method = methods{m};
68          label = methodLabels{m};
69
70          % Create discrete-time PID controller
71          PID.Discrete.(label).Cz{i} = pid(PID.Kp, PID.Ki, PID.Kd, PID.Tl,...
72              Ts, 'IFormula', method, 'DFormula', method);
73
74          % Extract numerator and denominator coefficients
75          [PID.Discrete.(label).num{i}, PID.Discrete.(label).den{i}] = ...
76              tfdata(PID.Discrete.(label).Cz{i}, 'v');
77      end
78  end
79
80  %% ZOH
81  PID.Discrete.ZOH1 = c2d(PID.Cs, specs.Ts(1), 'zoh');
82  PID.Discrete.ZOH10 = c2d(PID.Cs, specs.Ts(2), 'zoh');
83  PID.Discrete.ZOH50 = c2d(PID.Cs, specs.Ts(3), 'zoh');
84
85  [numZOH1, denZOH1] = tfdata(PID.Discrete.ZOH1, 'v');
86  [numZOH10, denZOH10] = tfdata(PID.Discrete.ZOH10, 'v');
87  [numZOH50, denZOH50] = tfdata(PID.Discrete.ZOH50, 'v');
```

A.3 State-Space Controller by Emulation

A.3.1 Observer Matrices Forward Euler

$T_s = 1$ ms: $\quad \Phi_o = 0.9000 \quad \boldsymbol{\Gamma}_o = \begin{bmatrix} 0.3054 & -3.7673 \end{bmatrix}$

$$\boldsymbol{H}_o = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \boldsymbol{J}_o = \begin{bmatrix} 0 & 1 \\ 0 & 37.6727 \end{bmatrix}$$

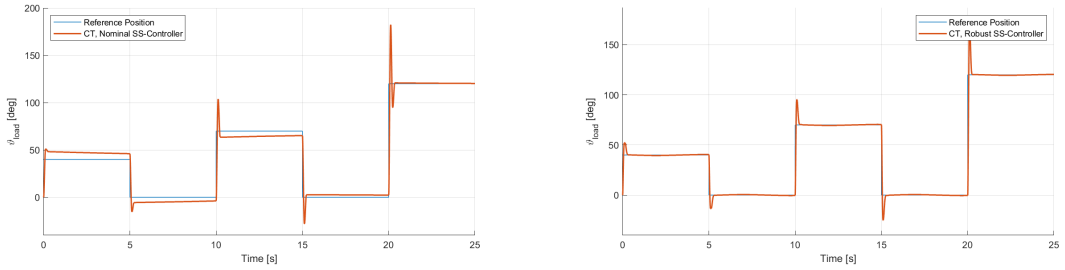
$T_s = 10$ ms: $\quad \Phi_o = 0 \quad \boldsymbol{\Gamma}_o = \begin{bmatrix} 3.0544 & -37.6727 \end{bmatrix}$

$$\boldsymbol{H}_o = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \boldsymbol{J}_o = \begin{bmatrix} 0 & 1 \\ 0 & 37.6727 \end{bmatrix}$$

$T_s = 50$ ms: $\quad \Phi_o = -4 \quad \boldsymbol{\Gamma}_o = \begin{bmatrix} 15.2719 & -188.3633 \end{bmatrix}$

$$\boldsymbol{H}_o = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \boldsymbol{J}_o = \begin{bmatrix} 0 & 1 \\ 0 & 37.6727 \end{bmatrix}$$

A.3.2 Simulation Results



(a) Nominal (b) Robust

Figure A.1: Simulated responses of the system controlled by a continuous-time, state-space controller, for the setpoints of 40°, 70° and 120°.

Table A.1: Comparison of performance indices to a 50° step reference for different discretization methods of nominal state-space controllers.

		Conti. Time	Forward E.	Backward E.	Tustin
$T_s = 1$ ms	M_p [%]	35.72	31.04	30.32	30.68
	$t_{s,5\%}$ [s]	-	-	-	-
	e_{ss} [%]	6.20	10.16	10.52	10.52
$T_s = 10$ ms	M_p [%]		44.74	36.08	40.40
	$t_{s,5\%}$ [s]		0.202	-	0.234
	e_{ss} [%]		-1.00	6.20	2.96
$T_s = 50$ ms	M_p [%]		168.2	51.20	80.00
	$t_{s,5\%}$ [s]		-	-	-
	e_{ss} [%]		-92.80	-6.04	11.96

Table A.2: Comparison of performance indices to a 50° step reference for different discretization methods of robust state-space controllers.

		Conti. Time	Forward E.	Backward E.	Tustin
$T_s = 1$ ms	M_p [%]	32.84	32.48	32.12	32.84
	$t_{s,5\%}$ [s]	0.230	0.231	0.230	0.230
	e_{ss} [%]	-1.00	-1.00	-1.00	-1.00
$T_s = 10$ ms	M_p [%]		44.72	42.92	44.00
	$t_{s,5\%}$ [s]		0.203	0.197	0.200
	e_{ss} [%]		0.80	-1.00	-0.64
$T_s = 50$ ms	M_p [%]		178.64	100.88	107.36
	$t_{s,5\%}$ [s]		-	0.467	0.774
	e_{ss} [%]		157.0	-1.00	1.88

A.3.3 MATLAB-Script

Code 2: script_22_startup.m

```
% Matlab script to start the Simulink simulation an accurate model of the
% Quanser SRV-02 + NI DAQ with a discrete time state-space controller and
% a reduced state observer, obtained by emulation.
clear


%% Load Predefined Parameters
```

```
load_params_inertial_case

%% User Inputs
% Motor Parameters
% Actual Parameters (estimated from Motor 1)
mld.Beq = 2.5663e-6;     % [Nm/(rad/s)]
mld.tausf = 0.013;       % [Nm]
mld.Jeq = 3.4640e-07;    % [kg m^2]

% Desired specifications
% Overshoot
specs.mp = 0.1;
% Settling Time
specs.settling_time = 0.15; %[s]



%% Simulation Parameters
% Sampling time
sIn.Ts =10e-3; %[s]

% Solver step time (0.1 ms)
sIn.step_size = 1e-4;

% Integration methos (1:FE,  2:BE,  3:Tustin)
sIn.integrationMethod = 2;

% Choice of nominal or robust controller (0:nominal,  1:robust)
sIn.nominal_robust = 0;

% List of reference positions [s]
sIn.position = [50];

% Time the reference positions are held [s]
sIn.sample_time = 5;

% Automatic calculation of total simulation time [s]
sIn.simulation_time = sIn.sample_time*length(sIn.position);


%% State-Space Model
% Plant Parameters
plant.km = (drv.dcgain*mot.Kt)/((mot.Req*mld.Beq) + (mot.Kt*mot.Ke));
plant.Tm = (mot.Req*mld.Jeq)/((mot.Req*mld.Beq) + (mot.Kt*mot.Ke));

% System matrices
plant.A = [0,       1;
           0,  -(1/plant.Tm)];
plant.B = [0;
           plant.km/(gbox.N*plant.Tm)];
plant.C = [1, 0];
plant.D = 0;

% Extended state-space model
plant.Ae = [0, plant.C; zeros(2,1), plant.A];
```

```
63  plant.Be = [0; plant.B];
64  plant.Ce = [0, plant.C];
65  plant.De = plant.D;
66
67
68  %% Closed Loop Eigenvalues
69  % Damping ratio from maximum overshoot
70  eigP.damping = (log(1/specs.mp)) / (sqrt((pi^2) + (log(1/specs.mp)^2)));
71
72  % Gain cut-off frequency from settling time, in [rad/s]
73  eigP.wn = 3/(eigP.damping*specs.settling_time);
74
75  % Real and imaginary parts
76  eigP.real = -eigP.damping*eigP.wn;
77  eigP.img = eigP.wn * sqrt(1 - eigP.damping^2);
78
79  % Desired eigenvalues for nominal tracking
80  eigP.values = [eigP.real + 1i*eigP.img, ...
81                 eigP.real - 1i*eigP.img];
82
83  % State feedback matrix
84  feedback.K = acker(plant.A, plant.B, eigP.values);
85
86  % State feedforward gain and input feedforward gain
87  feedback.gains = ([plant.A, plant.B; plant.C, plant.D])\[0;0;1];
88  feedback.Nx = feedback.gains(1:2);
89  feedback.Nu = feedback.gains(3);
90
91
92  % Eigenvalues for robust tracking
93  eigP.robustValues = [eigP.values, eigP.real];
94
95  % State feedback matrix frot the robust case
96  feedback.robustKe = acker(plant.Ae, plant.Be, eigP.robustValues);
97  feedback.robustKi = feedback.robustKe(1);
98  feedback.robustK  = feedback.robustKe(2:end);
99
100 % Selection
101 if sIn.nominal_robust == 1
102     feedback.K = feedback.robustK;
103 end
104
105
106 %% Reduced Observer Model
107 % Observer eigenvalue
108 eigO.damping = eigP.damping;
109 eigO.wn = 5*eigP.wn;
110 eigO.value = -eigO.damping*eigO.wn;
111
112 % Observer gain
113 obs.L = acker(plant.A(2,2), plant.A(1,2), eigO.value);
114
115 % Observer matrices (simpilifcations applied)
116 obs.AO = plant.A(2,2)-obs.L*plant.A(1,2);
117 obs.BO = [plant.B(2), obs.AO*obs.L];
```

```
obs.C0 = [0;
          1];
obs.D0 = [0,   1;
          0, obs.L];


%% Discretized Reduced Observer
% Forward Euler
obsFe.Phi0 = 1+obs.A0*sIn.Ts;
obsFe.Gamma0 = obs.B0*sIn.Ts;
obsFe.H0 = obs.C0;
obsFe.J0 = obs.D0;

% Backward Euler
obsBe.Phi0 = 1/(1-obs.A0*sIn.Ts);
obsBe.Gamma0 = 1/(1-obs.A0*sIn.Ts) *obs.B0*sIn.Ts;
obsBe.H0 = obs.C0/(1-obs.A0*sIn.Ts);
obsBe.J0 = obs.D0 + obs.C0/(1-obs.A0*sIn.Ts) *obs.B0*sIn.Ts;

% Tustin
obsTu.Phi0 = (1+(obs.A0*sIn.Ts)/2)/(1-(obs.A0*sIn.Ts)/2);
obsTu.Gamma0 = 1/(1-(obs.A0*sIn.Ts)/2) *obs.B0*sqrt(sIn.Ts);
obsTu.H0 = sqrt(sIn.Ts)*obs.C0 /(1-(obs.A0*sIn.Ts)/2);
obsTu.J0 = obs.D0 + obs.C0/(1-(obs.A0*sIn.Ts)/2) *obs.B0*sIn.Ts/2;

% Selection
obs.tmp = [obsFe, obsBe, obsTu];
obsD = obs.tmp(sIn.integrationMethod);
```

A.4 State-Space Controller by Direct Digital Design

A.4.1 Discretized Plant Matrices

$$T_s = 1 \text{ ms:} \qquad \boldsymbol{\Phi} = \begin{bmatrix} 1 & 0.0009695 \\ 0 & 0.9396 \end{bmatrix} \qquad \boldsymbol{\Gamma} = \begin{bmatrix} 0.0001496 \\ 0.2961 \end{bmatrix}$$

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad J = 0$$

The H and J values remain the same across the other sampling times.

$$T_s = 10 \text{ ms:} \qquad \boldsymbol{\Phi} = \begin{bmatrix} 1 & 0.007442 \\ 0 & 0.5362 \end{bmatrix} \qquad \boldsymbol{\Gamma} = \begin{bmatrix} 0.01254 \\ 2.273 \end{bmatrix}$$

$$T_s = 50 \text{ ms:} \qquad \boldsymbol{\Phi} = \begin{bmatrix} 1 & 0.01533 \\ 0 & 0.04432 \end{bmatrix} \qquad \boldsymbol{\Gamma} = \begin{bmatrix} 0.1699 \\ 4.683 \end{bmatrix}$$

A.4.2 Discretized Reduced Order Matrices

$$T_s = 1 \text{ ms} \qquad L = 35.8317$$

$$\Phi_o = 0.9048 \qquad \boldsymbol{\Gamma}_o = \begin{bmatrix} 0.2907 & -3.4098 \end{bmatrix}$$

$$\boldsymbol{H}_o = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \boldsymbol{J}_o = \begin{bmatrix} 0 & 1 \\ 0 & 35.83 \end{bmatrix}$$

$$T_s = 10 \text{ ms} \qquad L = 22.6171$$

$$\Phi_o = 0.3678 \qquad \boldsymbol{\Gamma}_o = \begin{bmatrix} 1.9893, -14.2967 \end{bmatrix}$$

$$\boldsymbol{H}_o = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \boldsymbol{J}_o = \begin{bmatrix} 0 & 1 \\ 0 & 22.6171 \end{bmatrix}$$

$$T_s = 50 \text{ ms} \qquad L = 2.4508$$

$$\Phi_o = 0.0067 \qquad \boldsymbol{\Gamma}_o = \begin{bmatrix} 4.2669, -2.4343 \end{bmatrix}$$

$$\boldsymbol{H}_o = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \boldsymbol{J}_o = \begin{bmatrix} 0 & 1 \\ 0 & 2.4508 \end{bmatrix}$$

A.4.3 MATLAB-Script

Code 3: script_23_startup.m

```
% Matlab script to start the Simulink simulation of the blackbox- and real
% model of the Quanser SRV-02 + NI DAQ. For descrete robust tracking where
% you can choose 3 different sampling times for the controllers
clear

%% Load Predefined Parameters
load_params_inertial_case

%% User Inputs
% Choose Nominal(0) or Robust(1)
n_r = 0;

% Define the sampler times for the controler and observer
sIn.T_s = 1e-3; %1ms
sIn.T_s2 = 1e-2; %10ms
sIn.T_s3 = 5e-2; %50ms

% Solver step time (0.1 ms)
sIn.step_size = 1e-4;

% Simulation time
sIn.simulation_time = 5;

```

```
% Actual Parameters (estimated from Motor 1)
mld.Beq = 2.5663e-6;     % [Nm/(rad/s)]
mld.tausf = 0.013;       % [Nm]
mld.Jeq = 3.4640e-07;    % [kg m^2]


%% Reduced Plant Parameters
plant.km = (drv.dcgain*mot.Kt)/((mot.Req*mld.Beq) + (mot.Kt*mot.Ke));
plant.Tm = (mot.Req*mld.Jeq)/((mot.Req*mld.Beq) + (mot.Kt*mot.Ke));


%% Closed Loop Eigenvalues
% Desired specifications
specs.mp = 0.1;
specs.settling_time = 0.15; %[seconds]

% Damping ratio from maximum overshoot
eigP.damping = (log(1/specs.mp)) / (sqrt((pi^2) + (log(1/specs.mp)^2)));

% Gain cut-off frequency from settling time, in [rad/s]
eigP.wn = 3/(eigP.damping*specs.settling_time);

% Real and imaginary parts
eigP.real = -eigP.damping*eigP.wn;
eigP.img = eigP.wn * sqrt(1 - eigP.damping^2);

% Desired eigenvalues for nominal tracking
eigP.values = [eigP.real + 1i*eigP.img, eigP.real - 1i*eigP.img];

% Desired eigenvalues for robust tracking
eigP.valuesR = [eigP.values, eigP.real];


%% Creating a SS for the reduced system
% Plant's A, B, C, D matrix
plant.A = [0 1; 0 -(1/plant.Tm)];
plant.B = [0 ;  plant.km/(gbox.N*plant.Tm)];
plant.C = [1 0];
plant.D = 0;
% Create a ss model
plant.sys_c = ss(plant.A, plant.B, plant.C, plant.D);


%% Calculate controler and observer gains
%Calculate controler and observer gains for t_s= 0.001
[feedback, obs] = calculate_controller(sIn.T_s, plant.sys_c, eigP, n_r);
%Calculate controler and observer gains for t_s = 0.01
[feedback2, obs2] = calculate_controller(sIn.T_s2, plant.sys_c, eigP, n_r);
%Calculate controler and observer gains for t_s = 0.05
[feedback3, obs3] = calculate_controller(sIn.T_s3, plant.sys_c, eigP, n_r);


%% Function to calculate controller
function [fb_out, obs_out] = calculate_controller(T_s, plant_c, eigP, n_r)
    %% Feedback
```

```
79      % Discretize system
80      sys_d = c2d(plant_c, T_s, "zoh");
81
82      % Augmented matrices for integral state
83      phi_e = [1, sys_d.C;
84               zeros(2,1), sys_d.A];
85      gamma_e = [0;
86                 sys_d.B];
87
88      % Discrete desired poles
89      eig_d = exp(T_s * eigP.valuesR);
90      eig_d = eig_d(1:end-(1-n_r)); %remove the last, if not robust
91
92      % Feedback gains
93      if n_r == 1
94          K_e = place(phi_e, gamma_e, eig_d);
95
96          fb_out.Ki = K_e(1);
97          fb_out.K  = K_e(2:end);
98      else
99          fb_out.Ki = 0;
100         fb_out.K = place(sys_d.A, sys_d.B, eig_d);
101     end
102
103     % Feedforward gains
104     ff = ([sys_d.A - eye(2), sys_d.B; sys_d.C, sys_d.D]) \ [0; 0; 1];
105     fb_out.Nx = ff(1:2);
106     fb_out.Nu = ff(3);
107
108     %% Reduced observer
109     %Calculate the eigenvalue for the observer
110     eigs_obs = exp(-eigP.damping * 5 * eigP.wn * T_s);
111
112     %Calculate the observer gain
113     obs_out.L = place(sys_d.A(2,2), sys_d.A(1,2), eigs_obs);
114
115     %Calcualte the observer matrices
116     obs_out.phi0  = sys_d.A(2,2) - obs_out.L * sys_d.A(1,2);
117
118     obs_out.gamma0 = [sys_d.B(2) - obs_out.L * sys_d.B(1), ...
119               obs_out.phi0*obs_out.L+sys_d.A(2,1)-obs_out.L*sys_d.A(1,1)];
120     obs_out.H0 = [0; 1];
121     obs_out.J0 = [0 1; 0 obs_out.L];
122 end
```