Laboratory Report 2: Digital position control of a DC servomotor

Noyan Erdin Kilic; email: noyanerdin.kilic@studenti.unipd.it
Group: 5 (69), Shift: Friday, components:
Noyan Erdin Kilic
Andrea Gentile
Cristian Voltan
Jan Luca Michael Cordes

June 19, 2025

Contents

1 Introduction

1.1 Activity Goal

This laboratory activity focuses on the design and experimental validation of digital position controllers for the Quanser SRV-02 DC servomotor, using two distinct methodologies. The first is Design by Emulation, where a continuous-time controller is initially developed and subsequently discretized using numerical methods. The second is Direct Digital Design, in which the controller is formulated directly in the discrete-time domain, explicitly taking into account the sampling constraints inherent to digital control platforms.

Both approaches aim to ensure accurate reference tracking, effective disturbance rejection, and compliance with specified transient performance criteria. To mitigate the effects of actuator saturation, anti-windup mechanisms are implemented. Additionally, state-space techniques are employed to investigate more advanced control strategies.

1.2 System and Model

The Quanser SRV-02 servomechanism is modeled as a second-order linear time-invariant (LTI) system that captures the dominant mechanical dynamics while neglecting faster electrical transients, which have negligible impact on position control.

The system input is the motor drive voltage, and the output is the angular position of the load, measured via the encoder. Using the physical parameters of the motor, gearbox, and driver, the open-loop transfer function is defined as:

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

$$k_m = \frac{k_{drv} k_t}{R_{eq} B_{eq} + k_t k_e} \qquad T_m = \frac{R_{eq} J_{eq}}{R_{eq} B_{eq} + k_t k_e} \tag{2}$$

where k_m is the motor-driver gain, T_m is the time constant characterizing the combined mechanical and electrical dynamics, and N is the gear ratio. The plant parameters are given in Table 2.

For the purpose of controller synthesis, the system is also expressed in state-space form (given in a later section). The state vector is composed of the angular position and angular velocity of the load-side shaft, enabling the application of both classical and modern control design techniques.

1.3 Experimental Setup

Controller designs are first verified through numerical simulations and then validated experimentally using a real-time control platform based on Simulink Desktop Real-Time.

The experimental system is built around the Quanser SRV-02 servomechanism, which consists of a permanent-magnet DC motor coupled to an inertial load via a planetary gearbox with a 14:1 reduction ratio. Angular position feedback is primarily provided by an optical encoder, with a potentiometer sensor available as an alternative. Both sensors are connected to the output shaft through an anti-backlash gear mechanism, which serves to minimize mechanical play.

The motor is driven by a linear voltage amplifier, modeled as a first-order low-pass filter with a static gain of approximately 0.6 and a cut-off frequency close to 1.2 kHz. A shunt resistor is included in the circuit to enable current measurement.

Data acquisition and I/O operations are handled by a National Instruments DAQ board, either the PCI-6221 or PCIe-6321 model, which supports high-resolution analog and digital signals, counters, and encoder interfacing. All connections are routed through a BNC-2110 terminal board using shielded cabling. The system is powered by a regulated DC adapter and operated from a PC running MATLAB/Simulink. The use of the Simulink Desktop Real-Time toolbox guarantees deterministic execution of control tasks, enabling real-time interaction between the controller and the physical plant.

2 Tasks, Methodologies and Results

2.1 PID controller design by Emulation

The goal of this section is to describe the design and results of PID controllers designed using the emulation method. Three types of controllers are considered: a regular PID controller, a PID controller with an anti-windup mechanism, and a PID controller with feedforward compensation of the reference signal.

2.1.1 Regular PID controller

The transfer function of a PID controller in continuous-time is as follows:

$$C(s) = K_P + \frac{K_I}{s} + K_D s = K_P \left(1 + \frac{1}{T_I s} + T_D s\right) \tag{3}$$

$$T_I = \alpha T_D \tag{4}$$

where $\alpha = 4$.

The discrete equivalent of such a controller (with a "real" derivative component) can be derived using the backward euler discretization method as:

$$C(z) = K_P + K_I \frac{Tz}{z-1} + K_D \frac{z-1}{(T_L + T)z - T_L} \tag{5}$$

where T is the sampling time, and the time constant T_L is chosen using

$$T_L = \frac{1}{2\omega_{gc}} \tag{6}$$

with ω_{gc} being the gain crossover frequency of the continuous-time control design.

The design requirements for the controller were:

- perfect steady state tracking of step position (load side) references.
- perfect steady state rejection of constant torque disturbances.
- Step response (load side) with settling time $t_{s,5\%} \leqslant 0.15\,\text{s}$ and overshoot $M_p \leqslant 10\%$.

By using a second-order dominant pole approximation for the closed-loop system, these requirements were converted into open-loop transfer function requirements. Then, by using Bode's method, the values for the controller parameters were calculated. These values are shown in Table 1.

An example implementation of this controller can be seen in Figure 1.

Parameter	Value
K_P	10.4326
K_I	5
K_D	0.1996
T_I	0.0765
T_D	0.0191
T_L	0.0129

Table 1: PID Controller Parameters

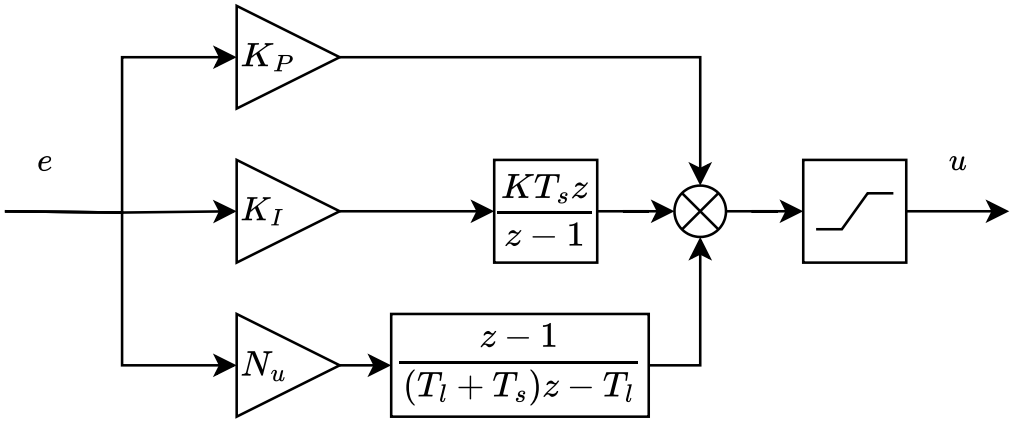

Figure 1: Implementation of the regular PID controller

This controller was tested for three different sampling times (1 ms, 10 ms, and 50 ms) and three different discretization methods (Euler Backward, Euler Forward, and Tustin). With a reference of 50 degrees, the output of the controlled system, along with performance measurements are summarized in Figures 2, 3, and 4.

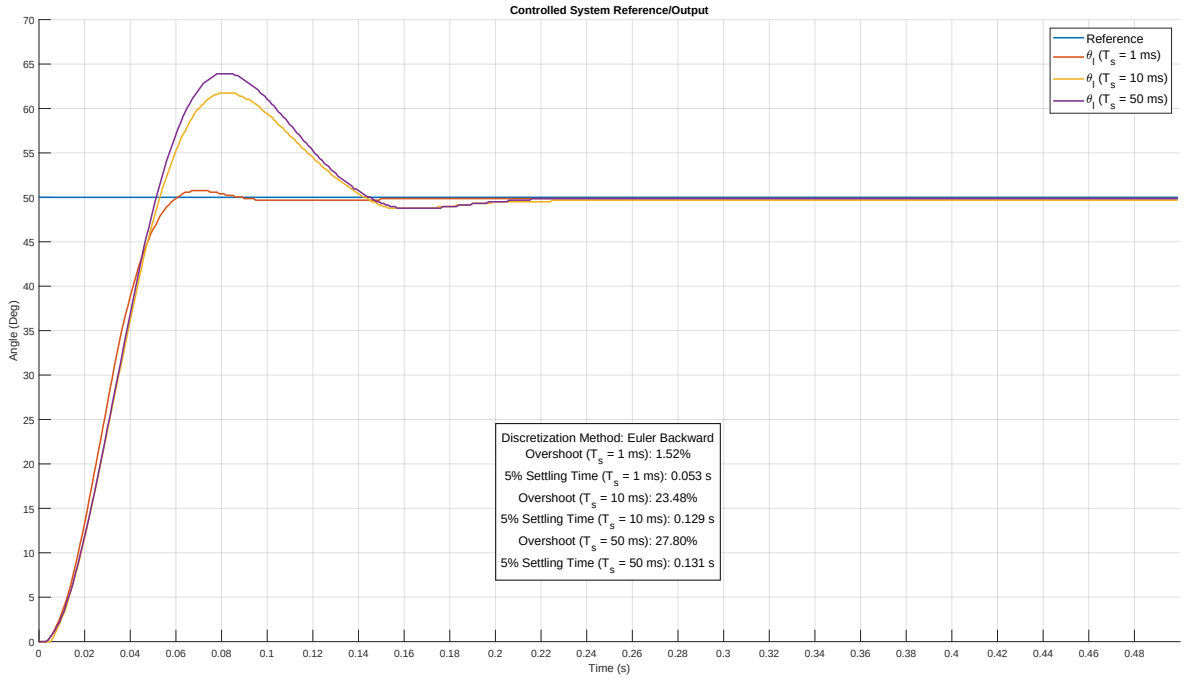

Figure 2: Response of the Controlled System (Euler Backward)

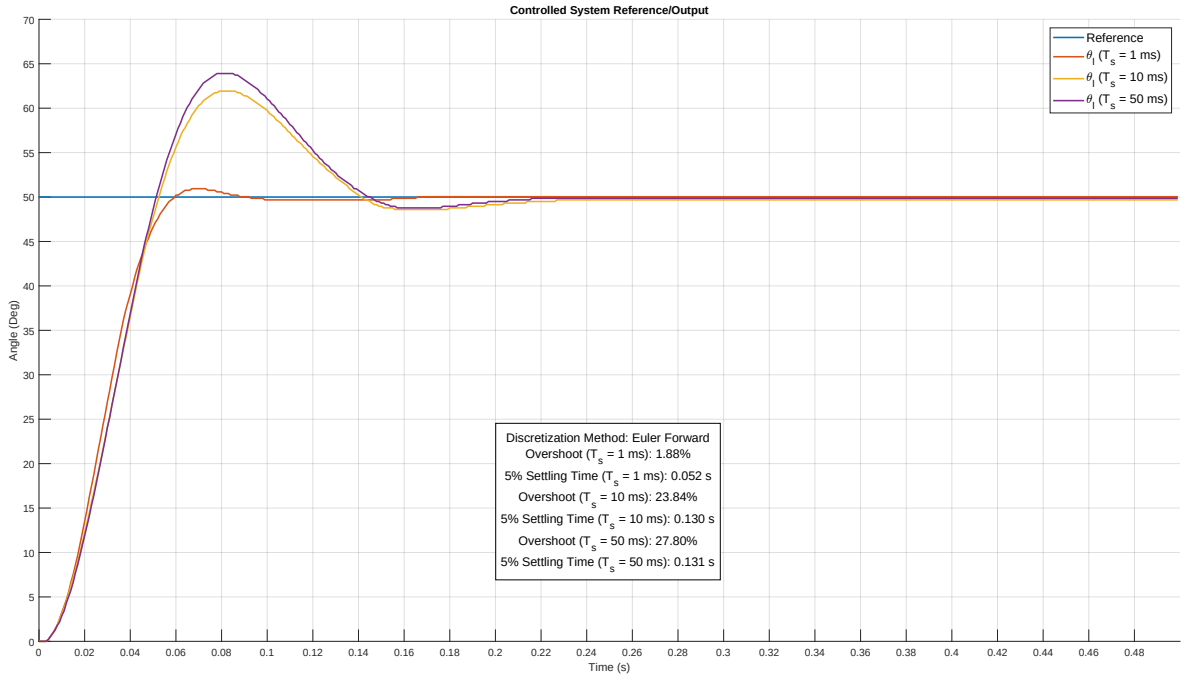

Figure 3: Response of the Controlled System (Euler Forward)

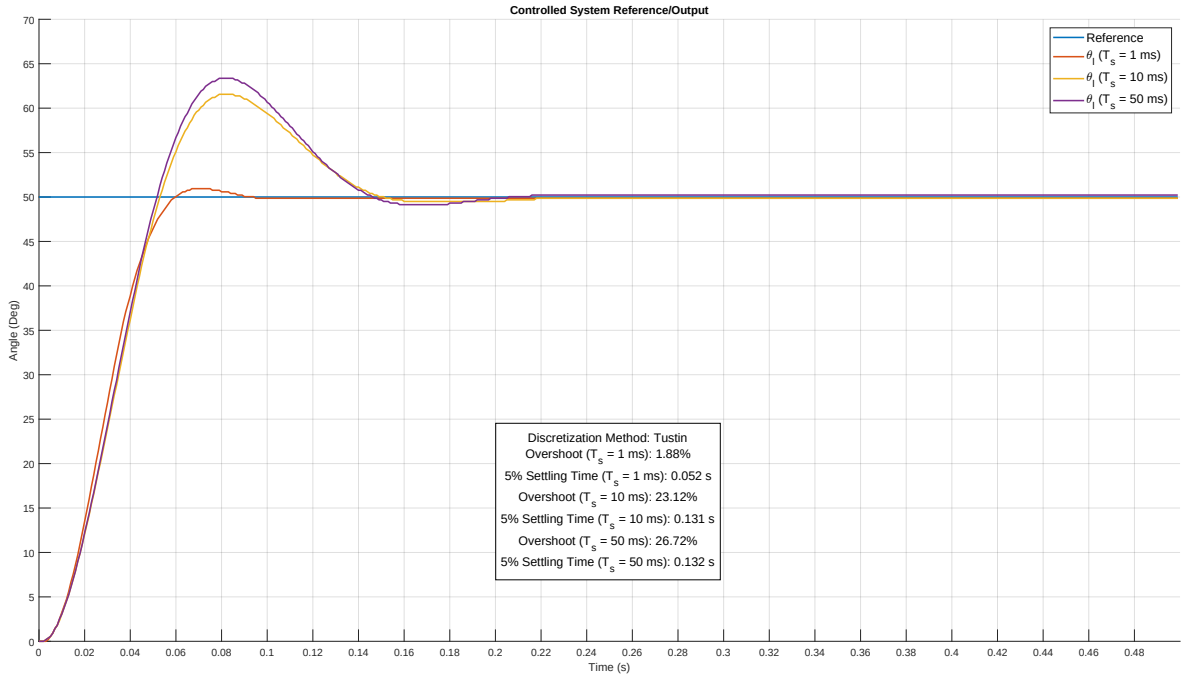

Figure 4: Response of the Controlled System (Tustin)

2.1.2 PID controller with Anti-Windup mechanism

The actuator outputs have a limited dynamic range in real systems, and for the system at hand, this was modeled in simulation by using a saturation block, as seen in Figure 1. When the actuator output is saturated, the feedback loop turns out to be effectively disabled.

A problem with the previously described regular PID design was that the integrator unnecessarily accumulates a large tracking error in case of saturated actuator output. To solve this problem, an anti-windup mechanism seen in Figure 5 was implemented.

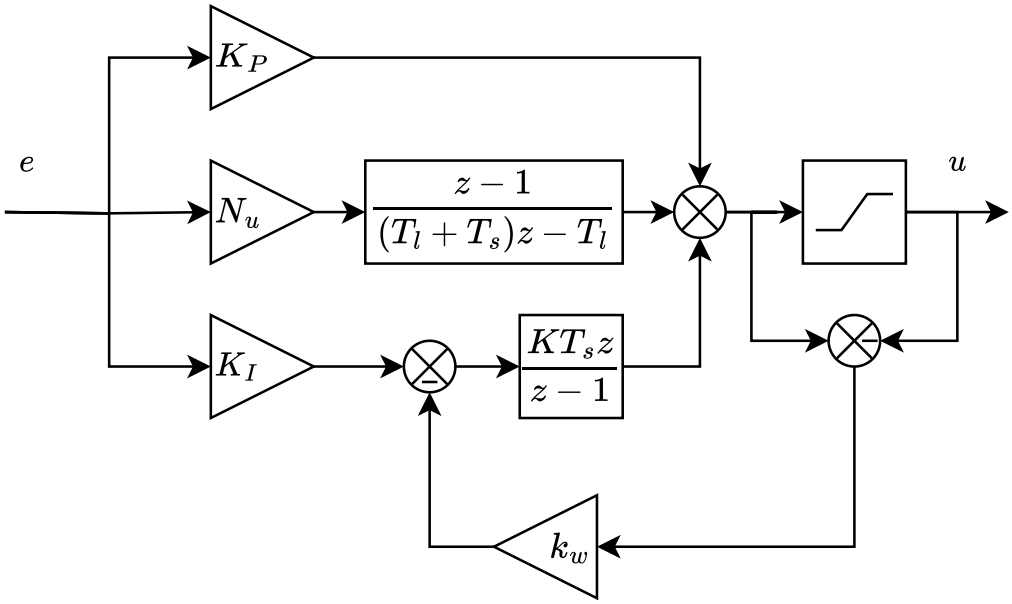

Figure 5: Implementation of the PID controller with anti-windup

The anti-windup gain K_W was chosen using the relations $K_W = \frac{1}{T_W}$, and $T_W = \frac{t_{s,5\%}}{5}$, and its value was calculated as $K_W = 41.6667$.

The response of the system with and without this new anti-windup mechanism can be seen in Figures 6, 7, and 8.

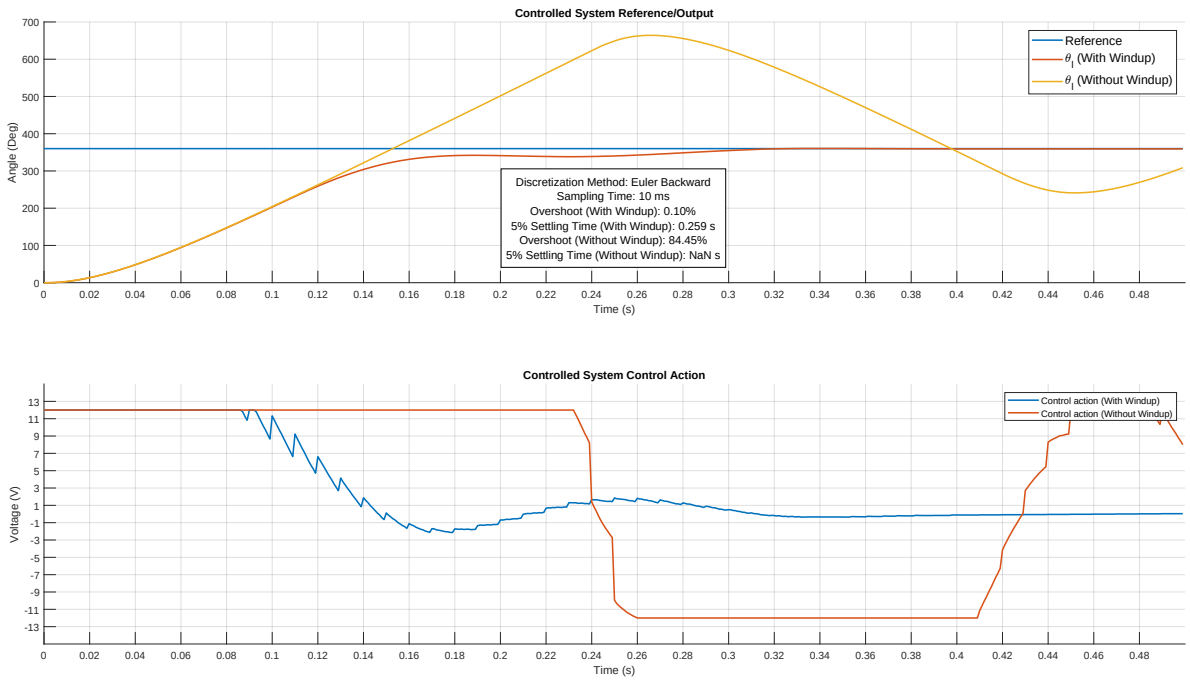

Figure 6: Response of the Controlled System (Euler Backward)

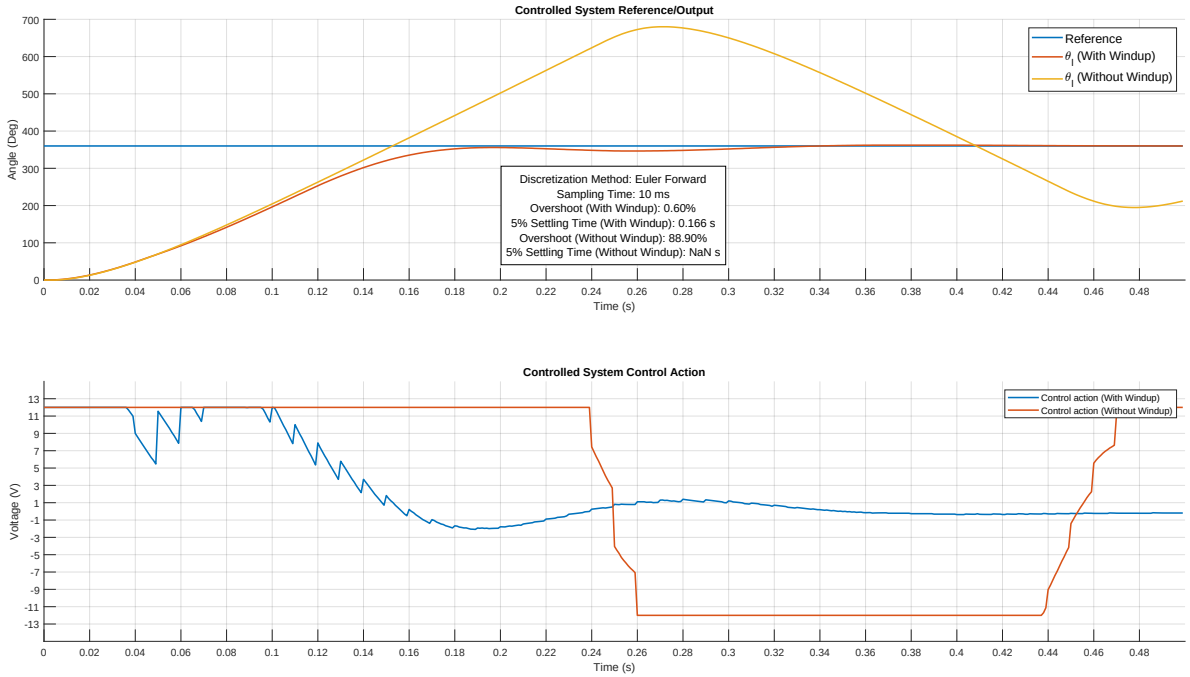

Figure 7: Response of the Controlled System (Euler Forward)

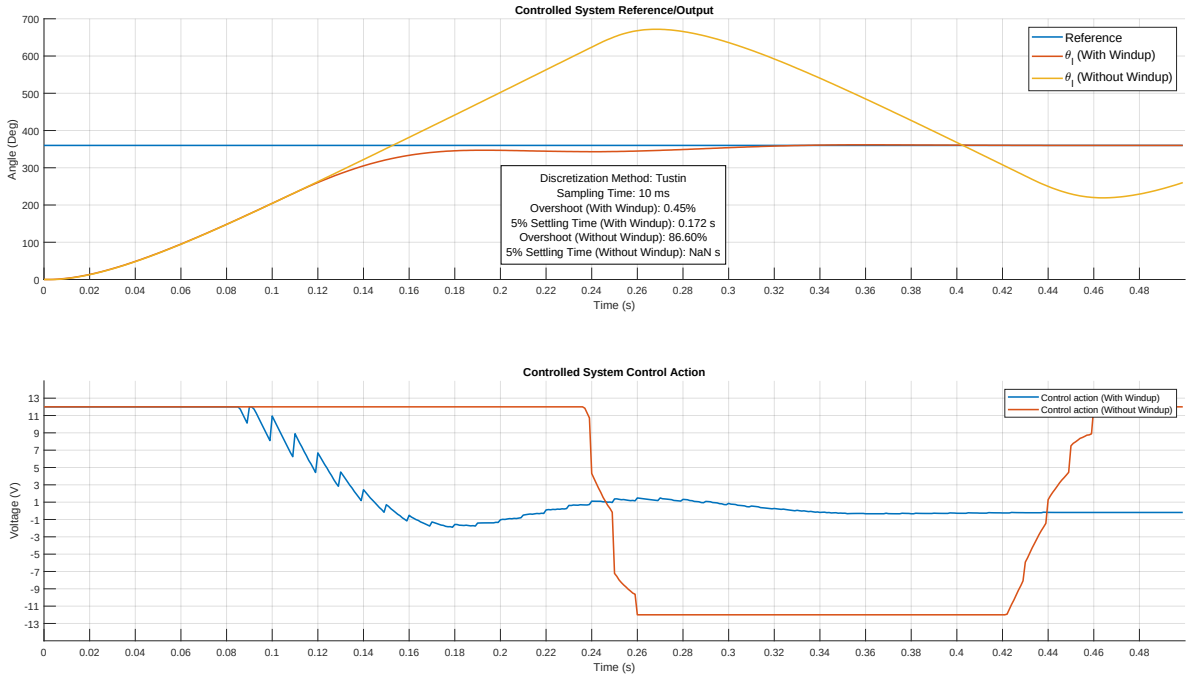

Figure 8: Response of the Controlled System (Tustin)

The large overshoot and "zig-zag" pattern of the responses without the anti-windup mechanism was caused by the error build-up by the integrator in the controller. As seen in above figures, the anti-windup mechanism was successful in eliminating this problem.

2.1.3 PID controller with feedforward compensation

To increase the effectiveness of the feedback control system, feedforward control actions can be incorporated. The structure of the feedforward compensation combined with the previously designed

regular PID controller can be seen in Figure 9.

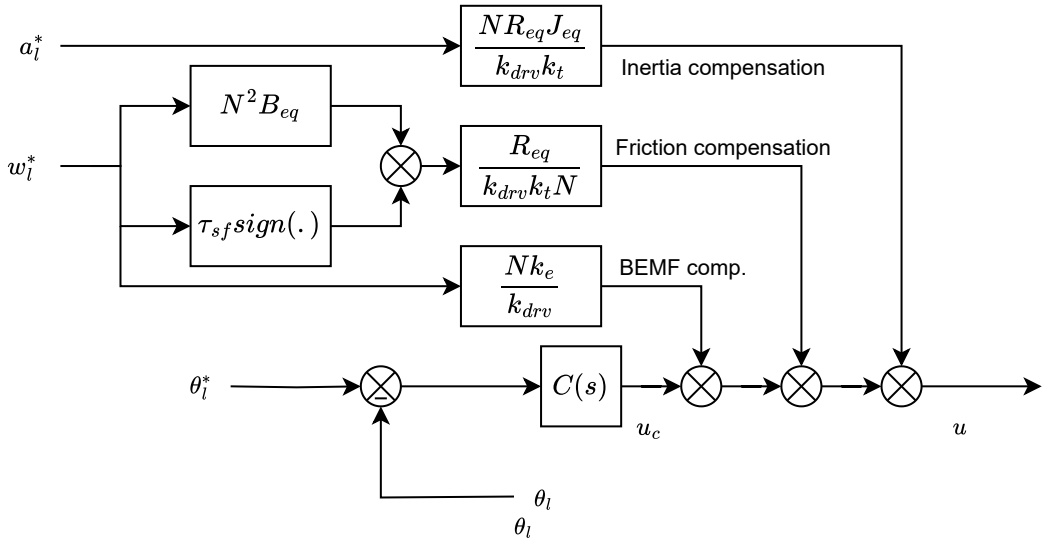

Figure 9: Implementation of the PID controller with feedforward compensation

The components in different compensation paths seen in Figure 9 were analytically found using the differential equations describing the plant. They were then modeled in simulation environment to accept inputs, the derivatives of the reference signal θ_l^*, the load angular position.

One complication with this feedforward scheme is that the first and second derivatives of the reference signals can not always be known in general. If the reference signal is known beforehand, it is possible to manually specify ("hard-code") the derivative signals.

For testing purposes, the reference signal and its derivatives were hand crafted using the following relations:

$$a_i^*(t) \triangleq \frac{d\omega_i^*(t)}{dt} = \begin{cases} 900\,\text{rpm/s} & \text{if } 0\,\text{s} \leqslant t < 0.5\,\text{s} \\ 0\,\text{rpm/s} & \text{if } 0.5\,\text{s} \leqslant t < 1\,\text{s} \\ -900\,\text{rpm/s} & \text{if } 1\,\text{s} \leqslant t < 2\,\text{s} \\ 0\,\text{rpm/s} & \text{if } 2\,\text{s} \leqslant t < 2.5\,\text{s} \\ 900\,\text{rpm/s} & \text{if } 2.5\,\text{s} < t < 3\,\text{s} \end{cases} \tag{7}$$

and

$$\omega_i^*(t) = \int_0^t a_i^*(\tau)\,d\tau, \quad \vartheta_i^*(t) = \int_0^t \omega_i^*(\tau)\,d\tau \tag{8}$$

The controller with this new feedforward compensation scheme was tested with different discretization methods. Following figures show the results.

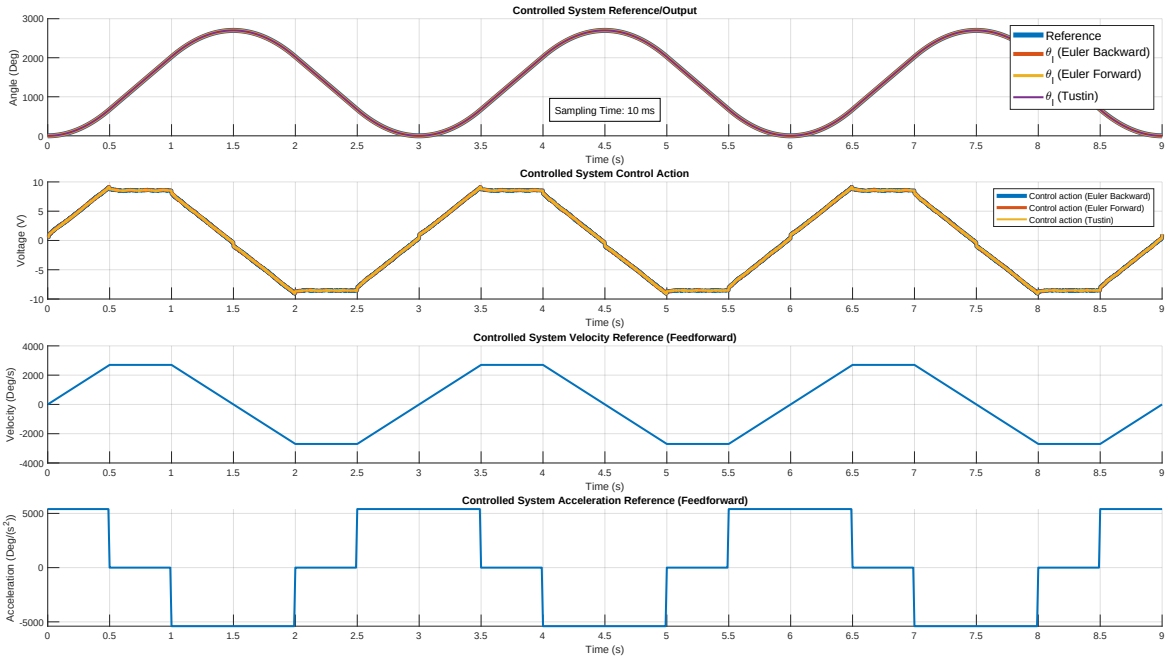

Figure 10: Response of the Controlled System

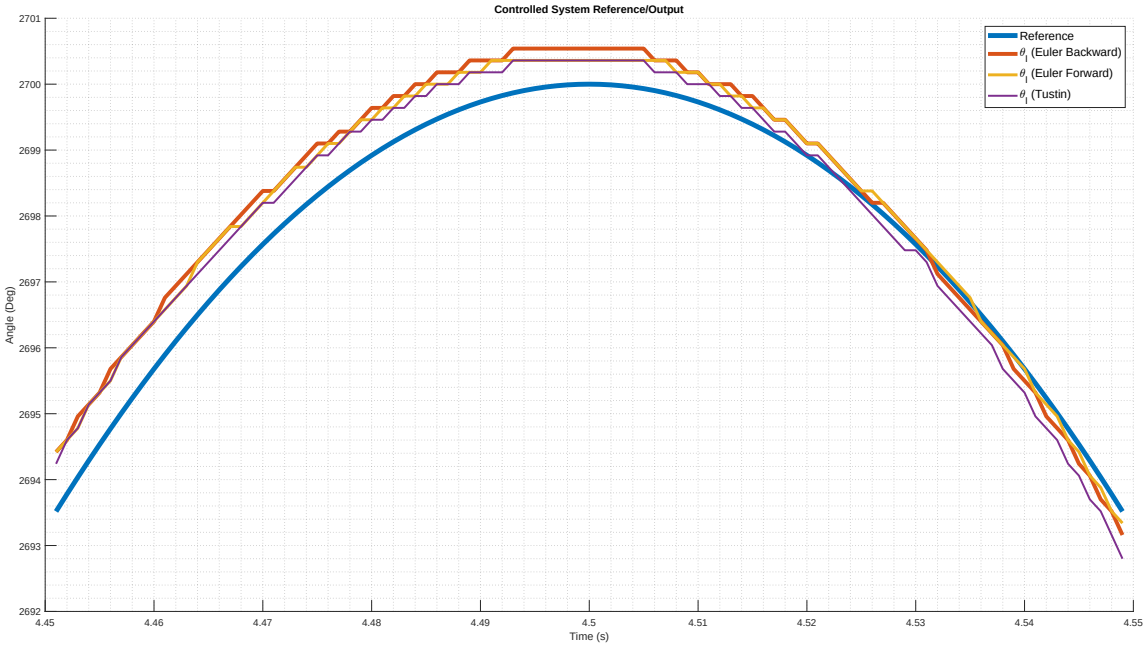

Figure 11: Response of the Controlled System (Zoomed)

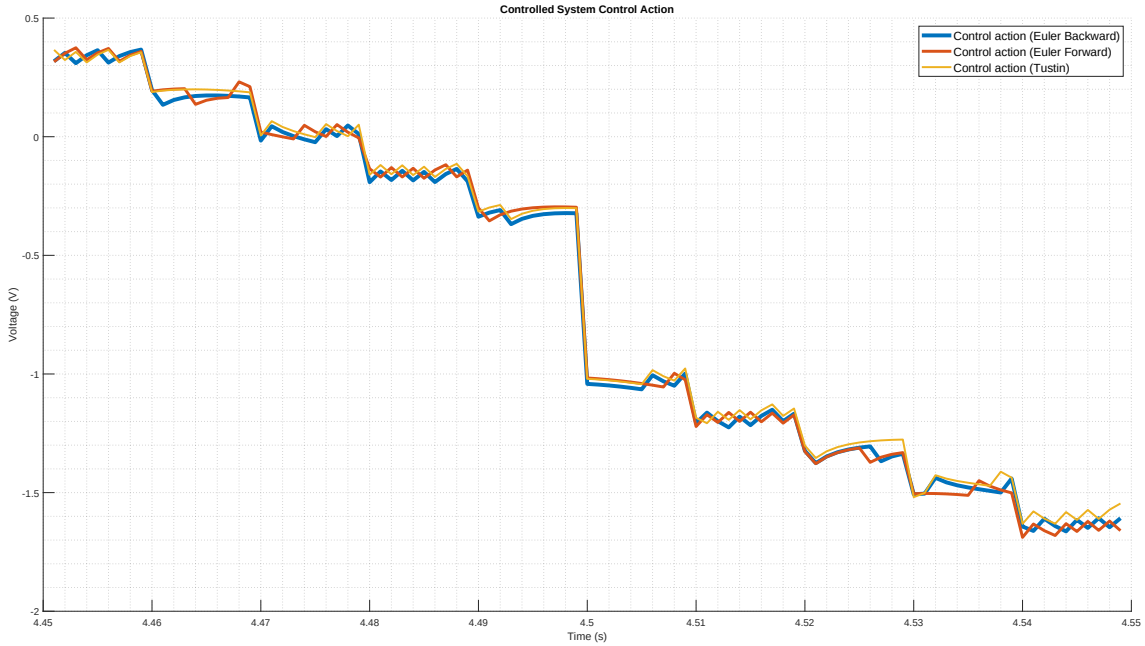

Figure 12: Control Action (Zoomed)

2.2 State space controller design by Emulation

The goal of this section is to validate the performance of two discrete-time position controllers implemented on the experimental setup:

A nominal state-space controller for constant reference tracking without integral action.

A state-space controller with integral action to compensate for steady-state errors.

Both controllers were obtained via forward Euler discretization of the corresponding continuous-time designs, following the design by emulation methodology. In both cases, a reduced-order observer was employed to estimate the unmeasured component of the system state, enabling the implementation of a regulator. Since not all components of the state vector are directly measurable, observer-based estimation was necessary. Specifically, the angular displacement of the load is directly measured through an optical encoder, while the angular velocity of the load is not directly accessible and must be reconstructed through state estimation. This reduced order observer structure was integrated into both controllers to allow closed-loop regulation . According to realization theory, it is possible to derive a state-space representation corresponding to the transfer function P(s) in section 1.2. The state vector is defined as $x = [\theta_l,\ \omega_l]^T$, where θ_l denotes the angular displacement and ω_l the angular velocity of the load-side shaft. The corresponding continuous-time state-space model, obtained via realization theory, is given by equation 9.

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{1}{T_m} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{k_m}{NT_m} \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad D = 0 \tag{9}$$

This representation is consistent with the second-order transfer function from which it is derived, as confirmed by the relationship:

$$P(s) = C(sI - A)^{-1} B + D \tag{10}$$

The availability of a state-space model enables the design of controllers using state feedback

techniques. In this section, this model will be exploited to synthesize both nominal and integral-action state-space controllers, relying on the placement of closed-loop poles to meet the following dynamic performances :

- Step response with overshoot $M_p \leqslant 10\%$
- Step response with 5% settling time $t_{s,5\%} \leqslant 0.15s$

2.2.1 Reduced order observer

To estimate the internal state of the DC gearmotor system, we designed a continuous-time reduced-order state observer, leveraging the fact that only the angular velocity ω_l of the load-side shaft needs to be estimated, since the angular position θ_l is directly measured by an encoder. Because of the structure of the C matrix, $C = [1\ 0]$, we do not need a change of basis. By selecting the state transformation matrix $T = I$, the design simplifies to the direct application of equations :

$$A_o = A'_{22} - LA'_{12} \qquad B_o = \left[B'_2 - LB'_1,\ (A'_{22} - LA'_{12})L + A'_{21} - LA'_{11}\right]$$

$$C_o = T \begin{bmatrix} 0_{p\times(n-p)} \\ I_{(n-p)\times(n-p)} \end{bmatrix} \quad D_o = T \begin{bmatrix} 0_{p\times m} & I_{p\times p} \\ 0_{(n-p)\times m} & L \end{bmatrix}$$

Following this procedure, the observer dynamics are expressed in terms of the new variable $z = \hat{v} - Ly$, leading to a dynamic system of the form:

$$\Sigma_{\text{obs}} : \begin{cases} \dot{z} = A_o z + B_o \begin{bmatrix} u \\ y \end{bmatrix} \\ \hat{x} = C_o z + D_o \begin{bmatrix} u \\ y \end{bmatrix} \end{cases}$$

The matrices are computed as follows:

$$A_o = -\frac{1}{T_m} - L, \quad B_o = \left[\tfrac{k_m}{NT_m},\ \ -\tfrac{1}{T_m} - L,\ \ L\right], \quad C_o = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad D_o = \begin{bmatrix} 0 & 1 \\ 0 & L \end{bmatrix}$$

The observer gain $L \in \mathbb{R}$ is chosen to ensure the asymptotic stability of the estimation error dynamics by placing the eigenvalue of the matrix A_o to $\lambda_o = -5 \times w_n$. This guarantees that the estimation of the unmeasured state converges rapidly and does not interfere with the controller dynamics. Once the observer is synthesized, the estimated state $\hat{x}$ can be substituted into the feedback control law, enabling the implementation of a dynamic output feedback regulator. This structure allows for effective control of the system even when full state measurement is not available.

2.2.2 Discretization of the observer

Once the reduced-order observer has been designed in continuous time, it is discretized using the forward Euler method, as required. According to the assignment instructions, the continuous-time observer is approximated in discrete time by the following state-space system:

$$\hat{\Sigma}_d : \begin{cases} z[k+1] = \Phi_o z[k] + \Gamma_o \begin{bmatrix} u[k] \\ y[k] \end{bmatrix} \\ \hat{x}[k] = H_o z[k] + J_o \begin{bmatrix} u[k] \\ y[k] \end{bmatrix} \end{cases}$$

Where the discrete-time matrices are computed as:

$$\Phi_o = I + A_o T_s, \quad \Gamma_o = B_o T_s, \quad H_o = C_o, \quad J_o = D_o$$

with T_s denoting the sampling time of the digital controller. This forward Euler discretization is straightforward to implement and preserves the observer structure by approximating the continuous-time dynamics over each sampling interval. The matrices A_o, B_o, C_o, and D_o are inherited from the continuous-time design and substituted into the expressions above to yield the numerical values for Φ_o, Γ_o, H_o, and J_o.

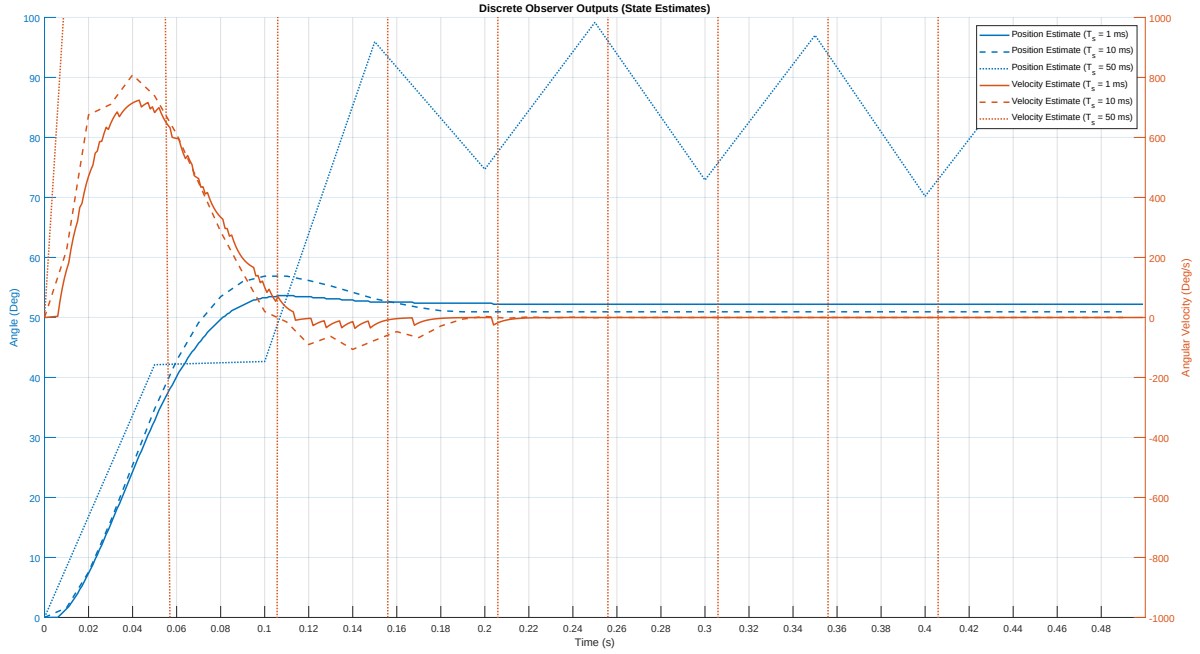

Figure 13: Nominal forward observer output

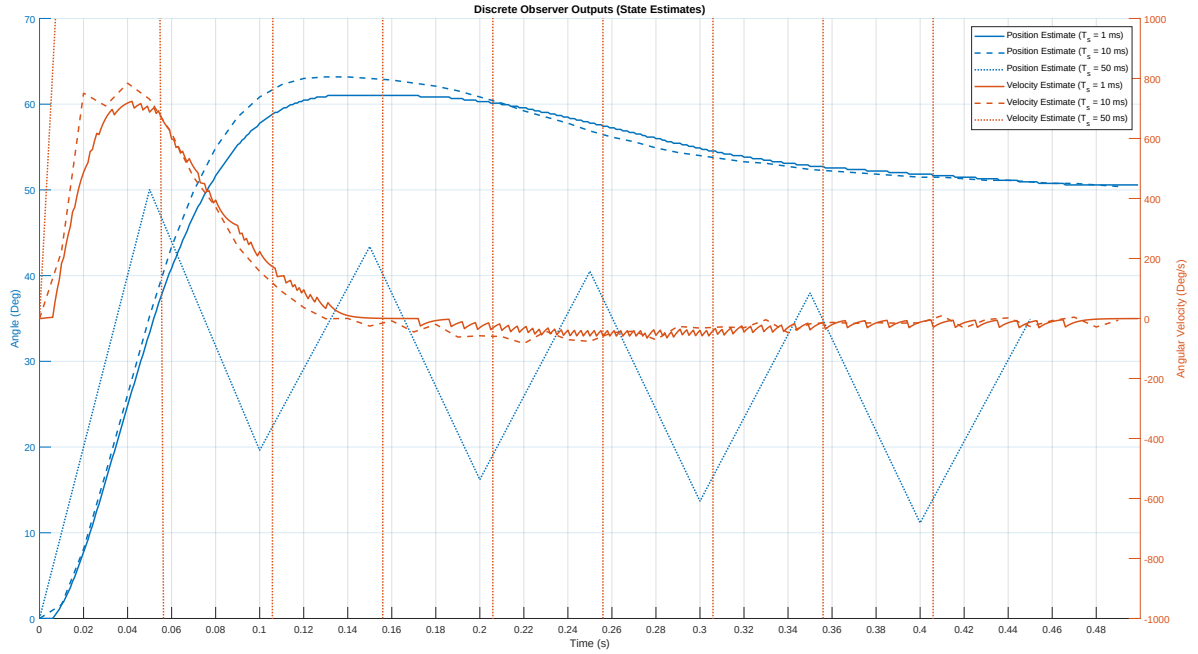

Figure 14: Integral forward observer output

2.2.3 Nominal Position State-Space Controller

The implementation of the nominal state-space controller begins with the design of a continuous-time state-feedback law that ensures the desired transient performance in terms of settling time and overshoot. To achieve this, we placed the poles accordingly:

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

$$\omega_d = \omega_n\sqrt{1-\delta^2}$$

$$\lambda_1 = -w_n\delta + i\omega_d$$

$$\lambda_2 = -w_n\delta - i\omega_d$$

$$\lambda = \begin{bmatrix} -20.0 + 27.2i \\ -20.0 - 27.2i \end{bmatrix}$$

With the Matlab command `K=place(A,B,`λ`)` finds the state feedback matrix K that places in poles in λ .

To achieve perfect asymptotic tracking of constant reference signals, two feedforward components were included. The steady-state values of the system state and input, x_∞ and u_∞, were expressed as linear functions of the reference r_∞. These gains were computed by solving the linear system:

$$\begin{bmatrix} A & B \\ C & 0 \end{bmatrix} \begin{bmatrix} N_x \\ N_u \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

where $N_x \in \mathbb{R}^{2\times 1}$ and $N_u \in \mathbb{R}$ denote the state and input feedforward gains, respectively. The

resulting control law is:

$$u = u_\infty - K(x - x_\infty) = N_u r_\infty - K(x - N_x r_\infty)$$

This structure ensures that the system output tracks the constant reference with zero steady-state error under nominal conditions. Since the full system state is not directly measurable, the control law is implemented using the estimated state provided by the previously designed reduced-order observer. This allows the controller to operate in a *regulator* configuration, combining state estimation with feedback and feedforward components. Given that the control law is purely algebraic, the discrete-time version was obtained by simply evaluating the same expression at discrete time steps. The forward Euler discretization method was used to discretize the plant dynamics for implementation on the experimental setup.

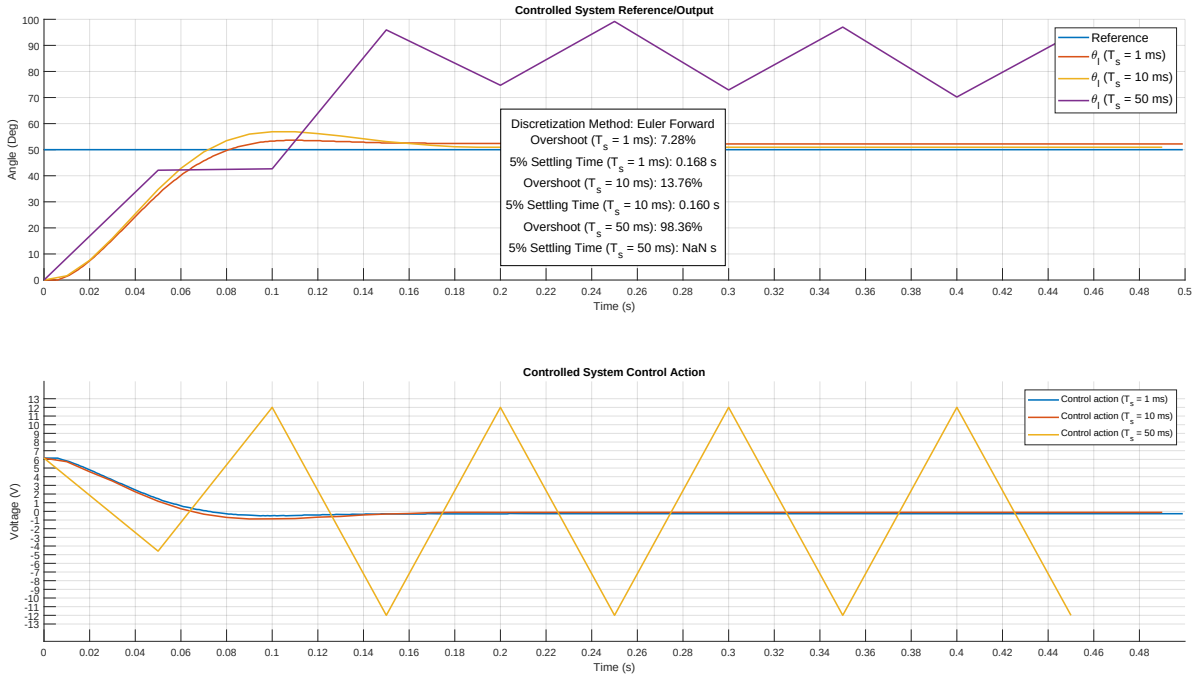

Figure 15: Nominal forward

2.2.4 Robust Position State-Space Controller

To enhance the tracking capabilities of the system in the presence of constant disturbances or model uncertainties, an integral action was incorporated into the control architecture. This addition ensures asymptotic rejection of constant disturbances and robustness of the closed-loop system, particularly with respect to steady-state tracking performance. To implement this, the original state-space model was augmented by introducing a new integrator state $x_I(t)$, defined as the integral of the tracking error:

$$x_I(t) = \int_0^t [y(\tau) - r(\tau)]\, d\tau$$

The resulting augmented state vector becomes $x_e = \begin{bmatrix} x_I & x^\top \end{bmatrix}^\top$, and the extended state-space model is expressed as:

$$\begin{bmatrix} \dot{x}_I(t) \\ \dot{x}(t) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & C \\ 0 & A \end{bmatrix}}_{A_e} \begin{bmatrix} x_I(t) \\ x(t) \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ B \end{bmatrix}}_{B_e} u(t) - \underbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{E_e} r(t)$$

$$y(t) = \underbrace{\begin{bmatrix} 0 & C \end{bmatrix}}_{C_e} \begin{bmatrix} x_I(t) \\ x(t) \end{bmatrix}$$

A state-feedback control law was then designed based on this extended model, with the general form:

$$u(t) = -K_e x_e(t) + u_\infty + K x_\infty$$

where $K_e = \begin{bmatrix} K_I & K \end{bmatrix}$ is the augmented feedback gain composed of the integral gain K_I and the original state feedback gain K. These gains were computed via pole placement applied to the closed-loop system:

$$A_{cl} = A_e - B_e K_e = \begin{bmatrix} 0 & C \\ -BK_I & A - BK \end{bmatrix}$$

The poles were selected to maintain the desired stability margin and convergence speed while accounting for the added integrator dynamics. Their locations were chosen based on engineering trade-offs between speed of response and overshoot, generally placing the integrator-associated pole on the real axis with a sufficient negative real part to ensure fast error correction without inducing instability. The best result were obtained with the allocation choice $\lambda_{c,\{1,2,3\}} = -\sigma = -w_n\delta = -20$. The integrator state x_I was updated in real time using the measured output y to compute the error $y - r$. For digital implementation, the integral term was discretized using the forward Euler method, in accordance with the discretization scheme used for the rest of the system. Unlike the nominal case, the inclusion of integral action requires careful tuning of the sampling time to avoid numerical instability and performance degradation, due to the higher order of the closed-loop dynamics. When implemented with sufficiently small sampling times, the robust controller demonstrated improved steady-state accuracy and reduced sensitivity to disturbances, at the cost of slightly degraded transient performance, as expected from the integrator dynamics.

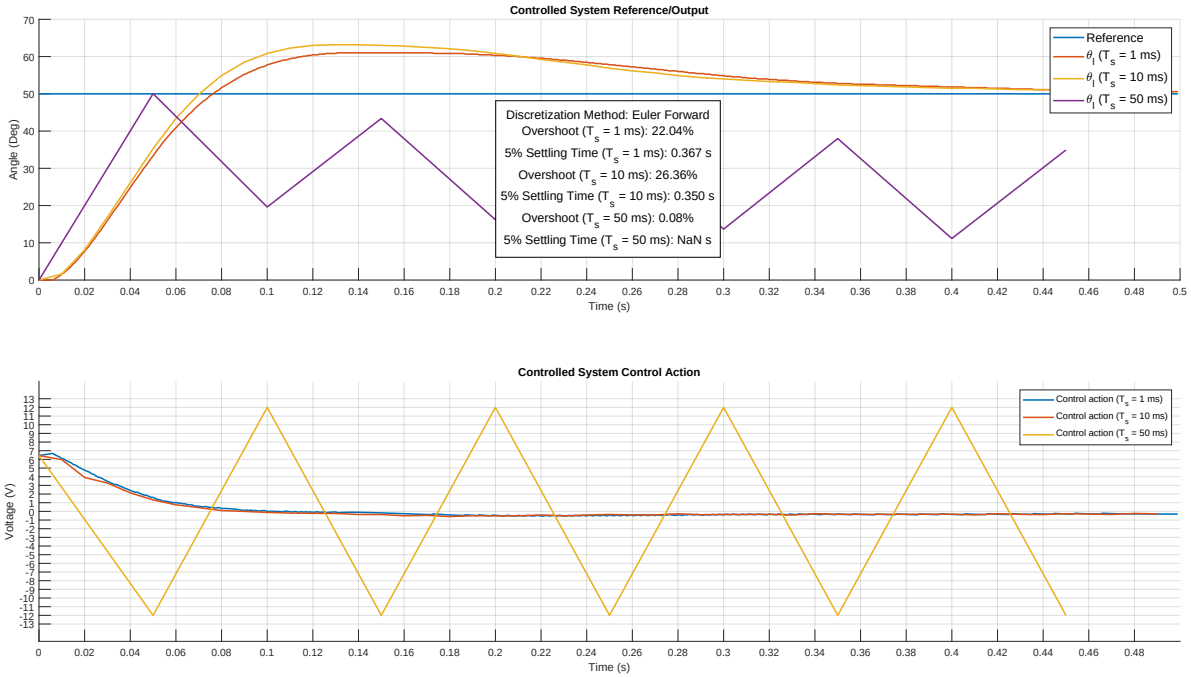

Figure 16: Integral forward

2.3 State space controller design by Direct Digital Design

In this section, the controller is designed directly in the discrete-time domain by applying an exact discretization of the continuous-time plant model using the zero-order hold (ZOH) method. This operation is carried out through MATLAB's c2d function, resulting in the discrete-time state-space model:

$$\Sigma_d = (\Phi, \Gamma, H, J)$$

where:

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

2.3.1 Nominal Position State-Space Controller

Starting from the discretized model $\Sigma_d = (\Phi, \Gamma, H, J)$, a discrete-time state-feedback controller is synthesized by assigning the closed-loop poles via standard pole placement. The desired discrete-time poles are obtained from the continuous-time ones through the exponential mapping:

$$\lambda_d = e^{\lambda_c T_s}.$$

To ensure reference tracking, the feedforward gains are computed as:

$$\begin{bmatrix} N_x \\ N_u \end{bmatrix} = \begin{bmatrix} \Phi - I & \Gamma \\ H & J \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

where N_x and N_u denote the state and input feedforward gains, respectively. These gains remain the same across different sampling times, since they depend only on the plant model.

A reduced-order observer is designed based on the discrete-time model, using the same methodology as in the emulation approach but now applied directly to (Φ, Γ, H, J).

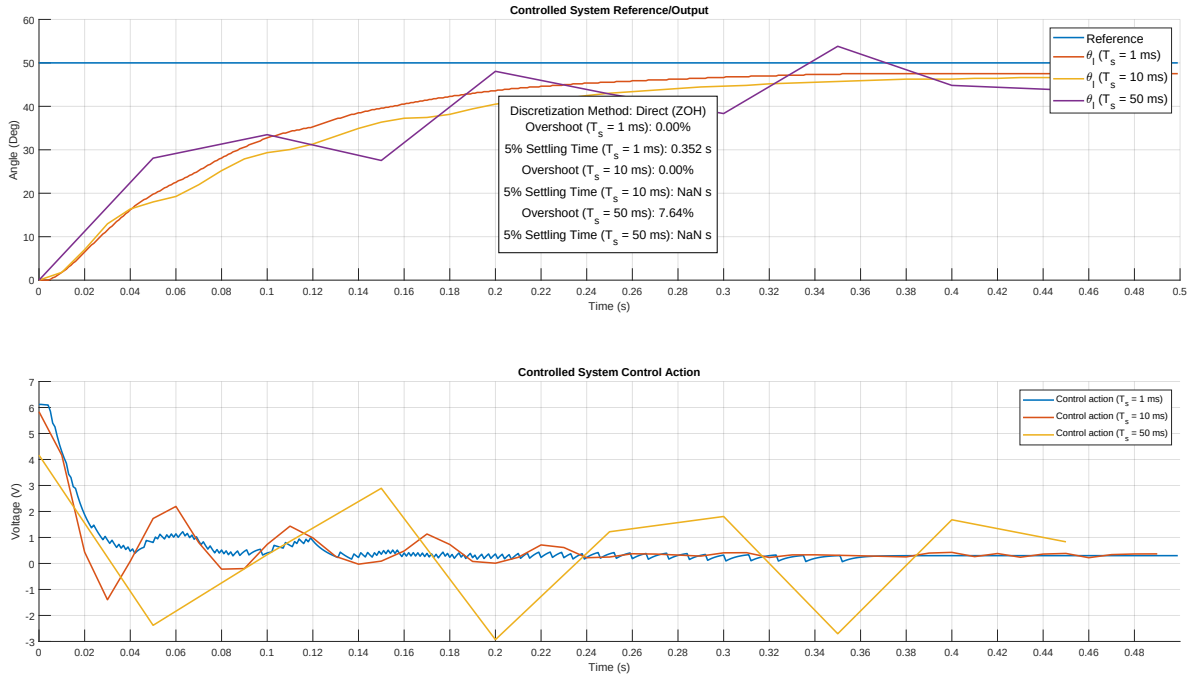

Figure 17: Direct digital nominal

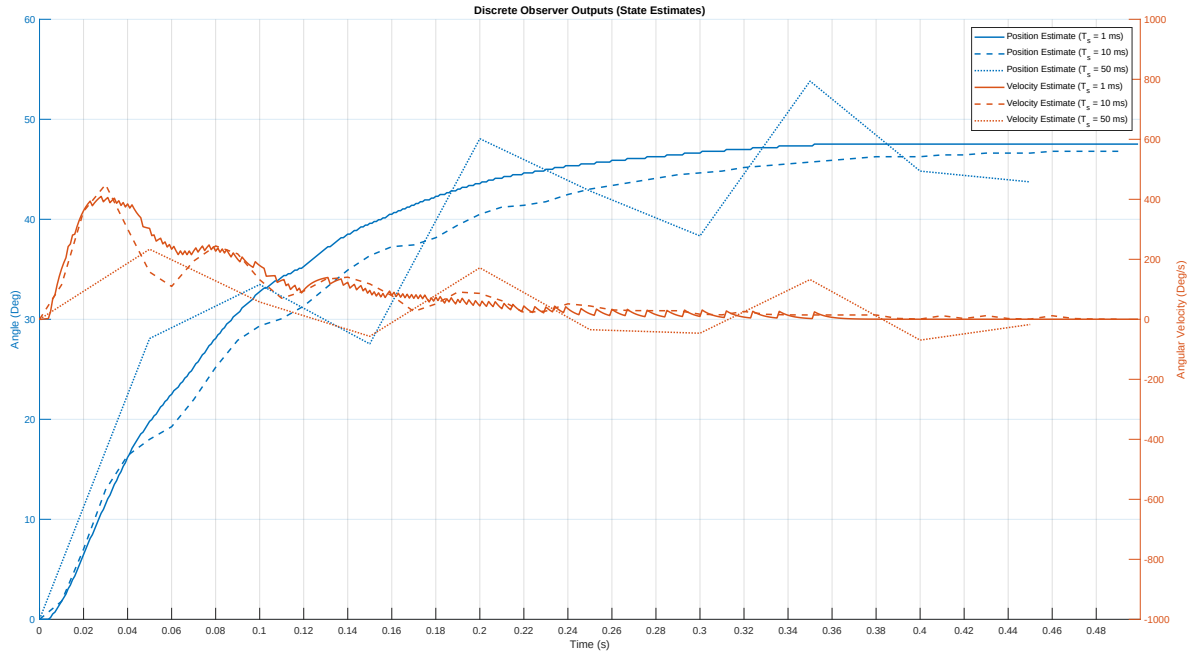

Figure 18: Direct digital nominal observer output

Figures 17 and 18 illustrate the closed-loop response and observer estimates. At $T_s = 1$ ms, the system shows fast, well-damped behavior with accurate estimation. As T_s increases, the performance gradually degrades, leading to slower responses, increased oscillations, and noisier estimates. Nevertheless, the ZOH-based design retains satisfactory performance for moderate sampling times, outperforming emulation with forward Euler, which exhibits instability or significant overshoot in such conditions.

2.3.2 Robust Position State-Space Controller

To improve reference tracking and reject disturbances, integral action is introduced by augmenting the state-space model. The extended discrete-time system becomes:

$$\Phi_e = \begin{bmatrix} 1 & H \\ 0 & \Phi \end{bmatrix}, \quad \Gamma_e = \begin{bmatrix} 0 \\ \Gamma \end{bmatrix}, \quad H_e = \begin{bmatrix} 0 & H \end{bmatrix}, \quad J_e = J.$$

The state-feedback and observer gains are again computed via pole placement, with poles mapped to the discrete domain using the same exponential transformation, as discussed earlier. The feedforward gains N_x and N_u remain unchanged.

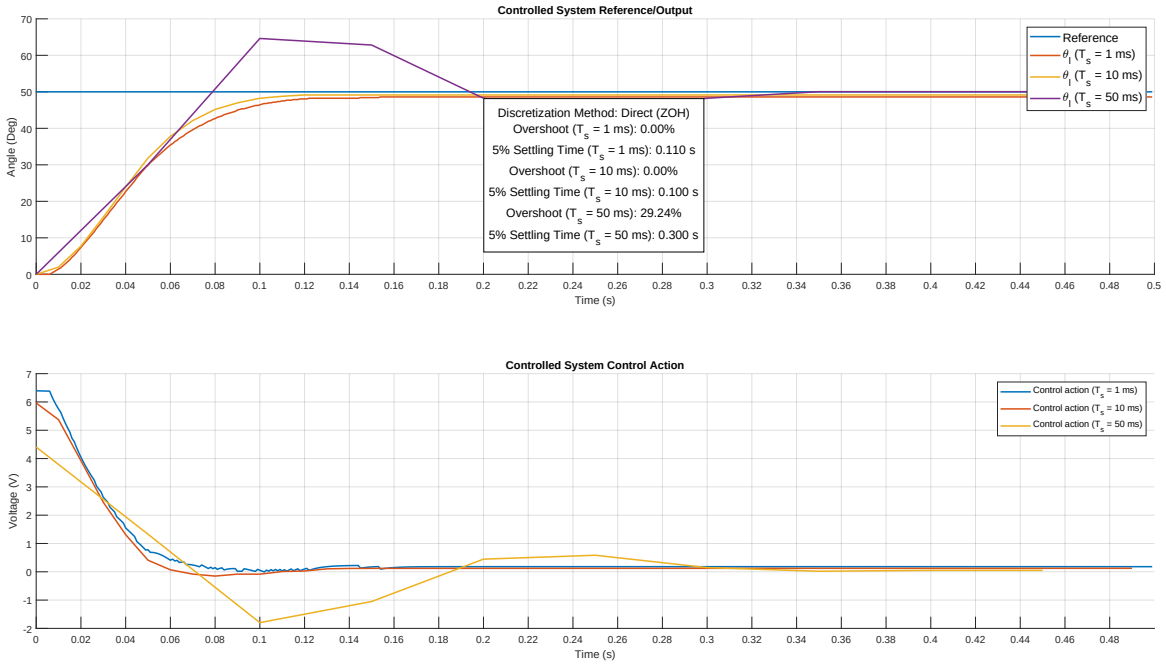

Figure 19: Direct digital robust

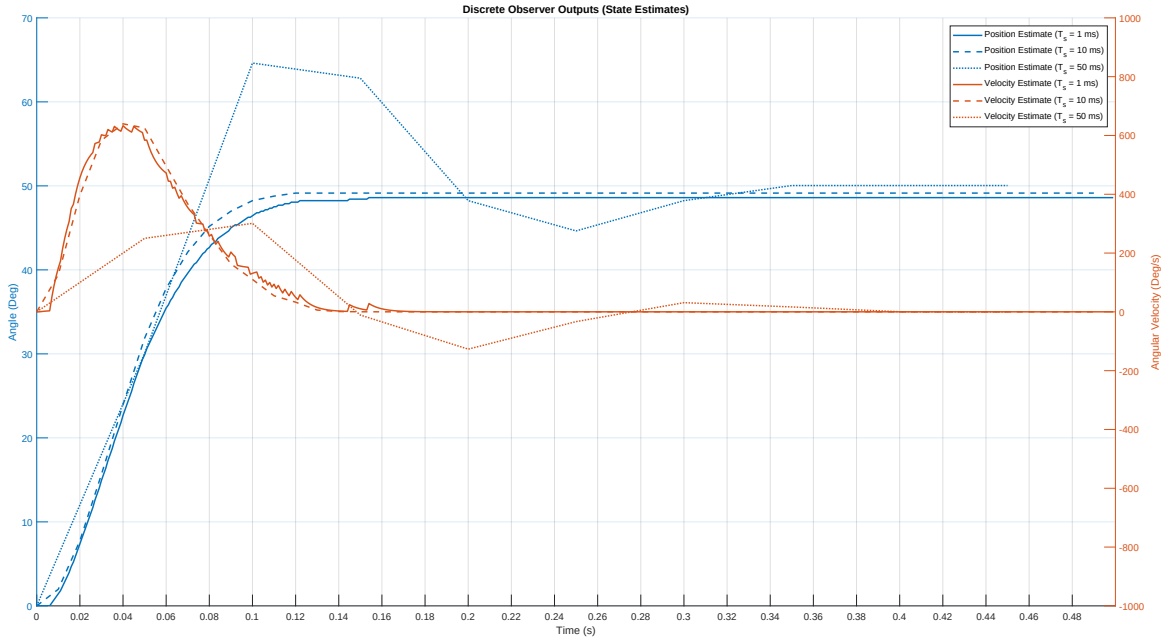

Figure 20: Direct digital robust observer output

As shown in Figures 19 and 20, the robust controller reduces the steady-state error across all sampling times, thanks to the integral action. The system exhibits excellent performance for $T_s = 1$ ms, while larger sampling intervals (e.g., 10 ms and 50 ms) lead to slower convergence and moderate oscillations. Estimation quality also degrades slightly at higher T_s, particularly for the velocity component.

Overall, the direct digital design using exact discretization proves to be more resilient to variations in sampling time compared to the emulation approach, particularly when forward Euler is used. The ZOH method ensures a more faithful representation of the original dynamics, making it a reliable choice for digital controller implementation.

A Appendix

A.1 Data sheet

Symbol	Description	Matlab Variable	Value
V_{mot}	Motor nominal voltage	–	6.0 V
R_m	Motor armature resistance	Rm	2.6 Ω
L_m	Motor armature inductance	Lm	0.18 mH
k_t	Motor current-torque constant	kt	7.68×10^{-3} Nm/A
k_e	Motor back-EMF constant	km	7.68×10^{-3} V $\cdot$ s/rad
	Low-gear total gear ratio	Kg	14
η_m	Motor efficiency	eta.m	0.69
η_g	Gearbox efficiency	eta.g	0.80
$J_{m,\text{rotor}}$	Rotor moment of inertia	Jm.rotor	3.90×10^{-7} kg $\cdot$ m^2
J_{eq}	High-gear equivalent moment of inertia (no load)	Jeq	6.51×10^{-7} kg $\cdot$ m^2
B_{eq}	High-gear equivalent viscous damping coefficient	Beq	1.22×10^{-7} N $\cdot$ m/(rad/s)
m_b	Mass of bar load	m.b	0.038 kg
L_b	Length of bar load	L.b	0.1525 m
m_d	Mass of disc load	m.d	0.04 kg
r_d	Radius of disc load	r.d	0.05 m
m_{max}	Maximum load mass	–	5 kg
f_{max}	Max input voltage frequency	–	50 Hz
I_{max}	Maximum input current	–	1 A
ω_{max}	Maximum motor speed	–	628.3 rad/s

Table 2: Main SRV02 specifications

Symbol	Description	Matlab Variable	Value
K_{gi}	Internal gearbox ratio	Kgi	14
$K_{ge,\text{low}}$	Internal gearbox ratio (low-gear)	Kge	1
$K_{ge,\text{high}}$	Internal gearbox ratio (high-gear)	Kge	5
m_{24}	Mass of 24-tooth gear	m24	0.005 kg
m_{72}	Mass of 72-tooth gear	m72	0.030 kg
m_{120}	Mass of 120-tooth gear	m120	0.083 kg
r_{24}	Radius of 24-tooth gear	r24	6.35×10^{-3} m
r_{72}	Radius of 72-tooth gear	r72	0.019 m
r_{120}	Radius of 120-tooth gear	r120	0.032 m

Table 3: SRV02 gearhead specifications

Symbol	Description	Matlab Variable	Value
K_{pot}	Potentiometer sensitivity	K_POT	35.2 deg/V
K_{enc}	SRV02-E encoder sensitivity	K_ENC	4096 counts/rev
K_{enc}	SRV02-EHR encoder sensitivity	K_ENC	8192 counts/rev
K_{tach}	Tachometer sensitivity	K_TACH	1.50 V/kRPM

Table 4: SRV02 sensor specifications

A.2 Detailed calculation for PID design

```
close all;

encoder_pulse_count = 500; % for motor 8 and 10, it is 1024, otherwise 500
encoder_quantization = 360 / (encoder_pulse_count * 4); % this is used in the simulink model
run('load_params.m')

B_eq = 8.1298e-07;
tau_sf = 0.0056039;
J_eq = 6.0731e-07;

%% construct the simplified plant transfer function (end of page 5 in the handout)
R_eq = (mot.R + sens.curr.Rs);
k_m = (drv.dcgain * mot.Kt) / (R_eq*B_eq + mot.Kt*mot.Ke);
T_m = (R_eq * J_eq) / (R_eq*B_eq + mot.Kt*mot.Ke);
simplified_P_num = k_m;
simplified_P_den = [T_m * gbox.N, gbox.N, 0];
P_TF = tf(simplified_P_num, simplified_P_den);

% settling time (limit is 0.15)
t_s = 0.12; % seconds
% overshoot (limit is 10%)
M_p = 0.07;

% sampling time
T_list = [0.001, 0.01, 0.05];

%% FOR REGULAR PID
% design the controller
save("ws_before_controller.mat");
[PID_K_P, PID_K_I, PID_K_D, PID_T_I, PID_T_D, PID_T_L, C_TF, OL_TF, CL_TF, OL_2TF, CL_2TF, w_n, w_gc, delta, phi_m] =
     controller_designer(t_s, M_p, P_TF);
PID_T_W = t_s / 5;
PID_K_W = 1 / PID_T_W;

% nice values
PID_K_I = 5;
PID_T_L = 1 / (2 * w_gc); % for PID derivative
SAMPLING_T = T_list(1); % USE 1, 2, 3 FOR TESTS (50 deg step)

%% FOR PID WITH WINDUP
% % design the controller
% save("ws_before_controller.mat");
% [PID_K_P, PID_K_I, PID_K_D, PID_T_I, PID_T_D, PID_T_L, C_TF, OL_TF, CL_TF, OL_2TF, CL_2TF, w_n, w_gc, delta, phi_m] =
      controller_designer(t_s, M_p, P_TF);
% PID_T_W = t_s / 5;
% PID_K_W = 1 / PID_T_W;
%
% % nice values
% PID_T_L = 1 / (2 * w_gc); % for PID derivative
% SAMPLING_T = T_list(2); % USE 2 FOR TESTS (360 deg step)

%% FOR PID WITH FEEDFORWARD
% % design the controller
% save("ws_before_controller.mat");
% [PID_K_P, PID_K_I, PID_K_D, PID_T_I, PID_T_D, PID_T_L, C_TF, OL_TF, CL_TF, OL_2TF, CL_2TF, w_n, w_gc, delta, phi_m] =
      controller_designer(t_s, M_p, P_TF);
% PID_T_W = t_s / 5;
% PID_K_W = 1 / PID_T_W;
%
% % nice values
% PID_K_P = 8;
% PID_K_I = 4;
% PID_K_D = 0.3;
%
% PID_T_L = 1 / (2 * w_gc); % for PID derivative
% SAMPLING_T = T_list(2); % USE 2 FOR TESTS
```

```
    function [PID_K_P, PID_K_I, PID_K_D, PID_T_I, PID_T_D, PID_T_L, controller_tf, ol_tf, cl_tf, ol_2tf, cl_2tf, w_n,
        w_gc, delta, phi_m] = controller_designer(t_s, M_p, P_TF)
load('ws_before_controller.mat'); % load parameters
%% 2.2 - Design and numerical validation of the position PID controller
% Formulation of control design specifications for the loop transfer function
delta = log(1 / M_p) / sqrt(pi^2 + (log(1 / M_p))^2);
w_n = 3 / (delta * t_s);
delta_2 = delta^2;
delta_4 = delta^4;
sqrt_1_delta_4 = sqrt(1 + 4 * delta_4);
phi_m = atan(2 * delta / (sqrt(sqrt_1_delta_4 - 2 * delta_2))); % phase margin
% NOTE: this approximation is super bad, the dominant poles are not even
% matching the resultant closed loop poles
cl_2tf = tf(w_n^2, [1, 2 * delta * w_n, w_n^2]); % approximation
ol_2tf = cl_2tf / (1 - cl_2tf); % approximation

w_gc = 3 / (delta * t_s); % approximation (open loop gain crossover freq.)
% % find the gain crossover freq. (the approximation does not give a nice result, empirically finding it here)
% [mag_OL, ~, w_OL] = bode(ol_2tf, logspace(-2, 2, 100000)); % WARN: freq range here will not work for every system
% mag_db_OL = 20 * log10(squeeze(mag_OL)); % unnecessary but whatever
% idx_P = find(mag_db_OL <= 0, 1, 'first');
% w_gc = w_OL(idx_P);

% disp("WE WANT");
% disp("nat. freq. w_n: " + num2str(w_n));
% disp("damping coef. delta: " + num2str(delta));
% disp("phase margin phi_m: " + num2str(phi_m));
% disp("ol gain cross. freq. w_gc: " + num2str(w_gc));
% disp("poles of the cl:");
% disp(pole(cl_2tf));

% freq response of P at gc
[mag, phase, ~] = bode(P_TF, w_gc);
phase = phase*pi/180; % gc phase (convert to rad)
delta_K = 1 / mag;
delta_phi = -pi + phi_m - phase;

% PID parameters
alpha = 4; % we're taking T_I / T_D 4 for now (alpha >= 4)
PID_K_P = delta_K * cos(delta_phi);
PID_T_D = (tan(delta_phi) + sqrt((tan(delta_phi))^2 + 4 / alpha)) / (2 * w_gc);
PID_T_I = alpha * PID_T_D;
PID_K_D = PID_K_P * PID_T_D;
PID_K_I = PID_K_P / PID_T_I;
% 1 / T_L = w_gc * 2 / 5
PID_T_L = 1 / (5 * w_gc); % 5 / 2 / w_gc

controller_tf = tf([PID_K_P*PID_T_D + PID_K_P*PID_T_L, PID_K_P + PID_K_I*PID_T_L, PID_K_I], [PID_T_L, 1, 0]);
ol_tf = controller_tf * P_TF;
cl_tf = feedback(ol_tf, 1);

% disp("AFTER DESIGN");
% disp("poles of the cl:");
% disp(pole(cl_tf));
% disp("zeros of the cl:");
% disp(zero(cl_tf));
% disp("P: " + num2str(PID_K_P));
% disp("I: " + num2str(PID_K_I));
% disp("D: " + num2str(PID_K_D));
end
```

A.3 Detailed calculations for state space design with emulation

```
%% Nominal control

% State space model
J_eq = 6.51e-7;
B_eq = 1.22e-6;
R_eq = (mot.R + sens.curr.Rs);

A = [0, 1; 0, -(R_eq*B_eq+mot.Kt*mot.Ke)/(R_eq*J_eq)];
B = [0; (drv.dcgain*mot.Kt)/(gbox.N*R_eq*J_eq)];
C = [1, 0];
D = 0;

% Specification
Mp = 0.1;
ts_5 = 0.15;

```

```
% Second order approximation
delta = log(1/Mp)/sqrt(pi^2+(log(1/Mp))^2);% damping coefficient
wn = 3 / (delta * ts_5);                    % natural frequency
lambda1 = -delta * wn + 1i * wn * sqrt(1 - delta^2);
lambda2 = -delta * wn - 1i * wn * sqrt(1 - delta^2);
poles = [lambda1, lambda2];
K = place(A, B, poles);

% Feedforward gains calculation
M = [A B; C 0];
rhs = [zeros(size(A,1),1); 1];
sol = M \ rhs;
Nx = sol(1:end-1);
Nu = sol(end);

%% Reduced order observer
lambda_o = 5 * (-wn);
A22 = A(2,2);
A12 = A(1,2);
L = place(A22, A12, lambda_o);

Ao = (A22 - L);
Bo = [B(2) , Ao * L ];
C_o = [0; 1];
D_o = [0, 1; 0, L];

%% Discretization

% Choise of sampling time T_s = 1e-3 / 10e-3 / 50e-3
T_s = 1e-3;

% Forward euler method
Phi_o = 1 + Ao * T_s;
Gamma_o = Bo * T_s;
H_o = C_o;
J_o = D_o;
```

```
%% Robust control

% State space model
J_eq = 6.51e-7;
B_eq = 1.22e-6;
R_eq = (mot.R + sens.curr.Rs);

A = [0, 1; 0, -(R_eq*B_eq+mot.Kt*mot.Ke)/(R_eq*J_eq)];
B = [0; (drv.dcgain*mot.Kt)/(gbox.N*R_eq*J_eq)];
C = [1, 0];
D = 0;

% Augmented state space model
A_e = [0 C; zeros(size(A,1),1), A];
B_e = [0;B];
C_e = [0,C];
D_e = 0;

% Specification
Mp = 0.1;
ts_5 = 0.15;

% Second order approximation
delta = log(1/Mp)/sqrt(pi^2+(log(1/Mp))^2);% damping coefficient
wn = 3 / (delta * ts_5);                    % natural frequency
sigma = delta * wn;
omega_d = wn * sqrt(1 - delta^2);

% Pole allocation choise
poli1 = [-sigma + 1i*omega_d, -sigma - 1i*omega_d, -sigma];
poli2 = [-sigma, -sigma, -sigma];
poli3 = [-2*sigma + 1i*omega_d, -2*sigma - 1i*omega_d, -2*sigma];
poli4 = [-2*sigma + 1i*omega_d, -2*sigma - 1i*omega_d, -3*sigma];

poli = poli2; % our choise
Ke = acker(A_e, B_e, poli);
K_I = Ke(1);
K = Ke(2:end);

% Feedforward gains calculation
M = [A B; C 0];
rhs = [zeros(size(A,1),1); 1];
sol = M \ rhs;
Nx = sol(1:end-1);
Nu = sol(end);
```

```
46
47 %% Reduced order observer
48 lambda_o = 5 * (-wn);
49 A22 = A(2,2);
50 A12 = A(1,2);
51 L = place(A22', A12', lambda_o)';
52 Ao = A22 - L;
53 Bo = [B(2), (A22 - L) * L];
54 Co = [0; 1];
55 Do = [0, 1; 0, L];
56
57 %% Discretization
58
59 % Choise of sampling time T_s = 1e-3 / 10e-3 / 50e-3
60 T_s = 1e-3;
61
62 % Forward euler method
63 Phi_o = 1 + Ao * T_s;
64 Gamma_o = Bo * T_s;
65 H_o = Co;
66 J_o = Do;
```

A.4 Detailed calculations for state space design with direct method

```
1  %% Direct nominal state spece design
2
3  % State space model
4  J_eq = 6.51e-7;
5  B_eq = 1.22e-6;
6  R_eq = (mot.R + sens.curr.Rs);
7
8  A = [0, 1; 0, -(R_eq*B_eq+mot.Kt*mot.Ke)/(R_eq*J_eq)];
9  B = [0; (drv.dcgain*mot.Kt)/(gbox.N*R_eq*J_eq)];
10 C = [1, 0];
11 D = 0;
12
13 sys_c = ss(A, B, C, D);
14
15 % Specification
16 Mp = 0.1;
17 ts_5 = 0.15;
18
19 % Second order approximation
20 delta = log(1/Mp)/sqrt(pi^2+(log(1/Mp))^2); % damping coefficient
21 wn = 3 / (delta * ts_5);                    % natural frequency
22
23 % Controller eigenvalues
24 lambda1 = -delta * wn + 1i * wn * sqrt(1 - delta^2);
25 lambda2 = -delta * wn - 1i * wn * sqrt(1 - delta^2);
26
27 % Observer eigenvalues
28 lambda_o_cont = 5*(-wn);
29
30 % Choise of sampling time T_s = 1e-3 / 10e-3 / 50e-3
31 T_s = 1e-3;
32
33 %% Discretization
34 sys_d = c2d(sys_c, T_s, 'zoh');
35 [Phi, Gamma, H, J] = ssdata(sys_d);
36
37 % Nominal control design
38 z1 = exp(lambda1 * T_s);
39 z2 = exp(lambda2 * T_s);
40 z_poli = [z1, z2];
41 K = place(Phi, Gamma, z_poli);
42
43 % Feedforward gains calculation
44 M = [Phi - eye(size(Phi)), Gamma; H, J];
45 rhs = [zeros(size(Phi,1),1); 1];
46 sol = M \ rhs;
47 Nx = sol(1:end-1);
48 Nu = sol(end);
49
50 % Reduced order observer design
51 Phi11 = Phi(1,1); Phi12 = Phi(1,2);
52 Phi21 = Phi(2,1); Phi22 = Phi(2,2);
53 Gamma1 = Gamma(1); Gamma2 = Gamma(2);
54 z_o = exp(lambda_o_cont * T_s);
55
```

```
Phi_o = Phi22 - L * Phi12;
Gamma_o = [Gamma2 - L * Gamma1, (Phi22 - L * Phi12) * L + Phi21 - L * Phi11];
H_o = [0; 1];
J_o = [0, 1; 0, L];
```

```
%% Direct robust state space design

% State space model
J_eq = 6.51e-7;
B_eq = 1.22e-6;
R_eq = (mot.R + sens.curr.Rs);

A = [0, 1; 0, -(R_eq*B_eq+mot.Kt*mot.Ke)/(R_eq*J_eq)];
B = [0; (drv.dcgain*mot.Kt)/(gbox.N*R_eq*J_eq)];
C = [1, 0];
D = 0;

sys_c = ss(A, B, C, D);

% Specification
Mp = 0.1;
ts_5 = 0.15;

% Second order approximation
delta = log(1/Mp)/sqrt(pi^2+(log(1/Mp))^2); % damping coefficient
wn = 3 / (delta * ts_5);                    % natural frequency

% Controller eigenvalues
sigma = delta * wn;
omega_d = wn * sqrt(1 - delta^2);
pole1 = [-sigma + 1i*omega_d, -sigma - 1i*omega_d, -sigma];
pole2 = [-sigma, -sigma, -sigma];
pole3 = [-2*sigma + 1i*omega_d, -2*sigma - 1i*omega_d, -2*sigma];
pole4 = [-2*sigma + 1i*omega_d, -2*sigma - 1i*omega_d, -3*sigma];

poli = poli2; % our choise

% Observer eigenvalues
lambda_o_s = 5*(-wn);

% Choise of sampling time T_s = 1e-3 / 10e-3 / 50e-3
T_s = 1e-3;

%% Discretization
sys_d = c2d(sys_c, T_s, 'zoh');
[Phi, Gamma, H, J] = ssdata(sys_d);

% Augmented state space model
Phi_e = [1, H; zeros(2,1), Phi];
Gam_e = [0; Gamma];
H_e = [0, H];
J_e = 0;

% Robust control design
pole_z = exp(pole2 * T_s);
Ke = acker(Phi_e, Gam_e, pole_z);
K_I = Ke(1);
K = Ke(2:end)

% Feedforward gains calculation
M = [Phi - eye(2), Gamma; H, J];
rhs = [0; 0; 1];
sol = M \ rhs;
Nx = sol(1:2);
Nu = sol(3);

% Reduced order observer design
Phi11 = Phi(1,1); Phi12 = Phi(1,2);
Phi21 = Phi(2,1); Phi22 = Phi(2,2);
Gam1 = Gamma(1);   Gam2 = Gamma(2);
lambda_o_z = exp(lambda_o_s * T_s);
L = place(Phi22', Phi12', lambda_o_z)';

Phi_o = Phi22 - L * Phi12;
Gamma_o = [Gam2 - L * Gam1, (Phi22 - L * Phi12) * L + Phi21 - L * Phi11];
H_o = [0; 1];
J_o = [0, 1; 0, L];
```

A.5 Example of Simulink model for numerical simulations

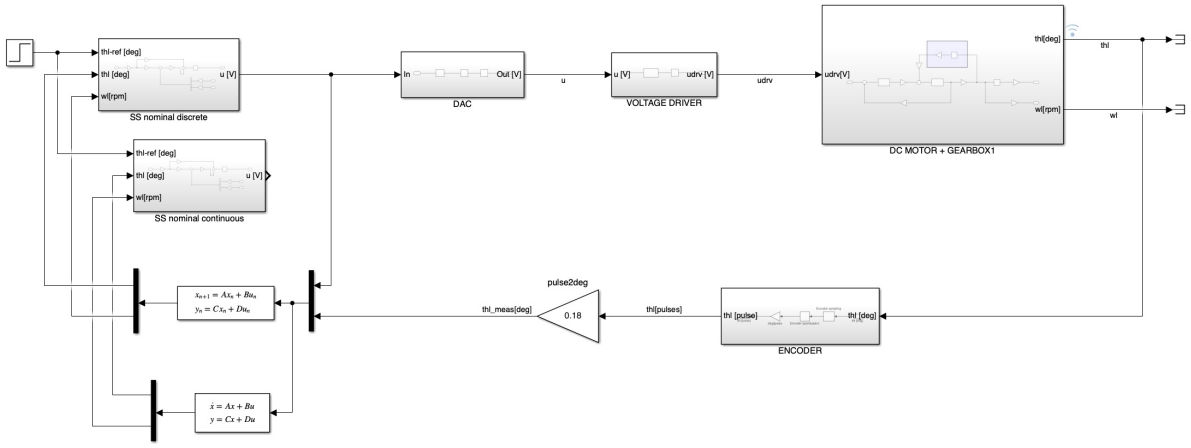

Figure 21: State-space nominal controller with emulation method

A.6 Example of Simulink model for experimental simulations

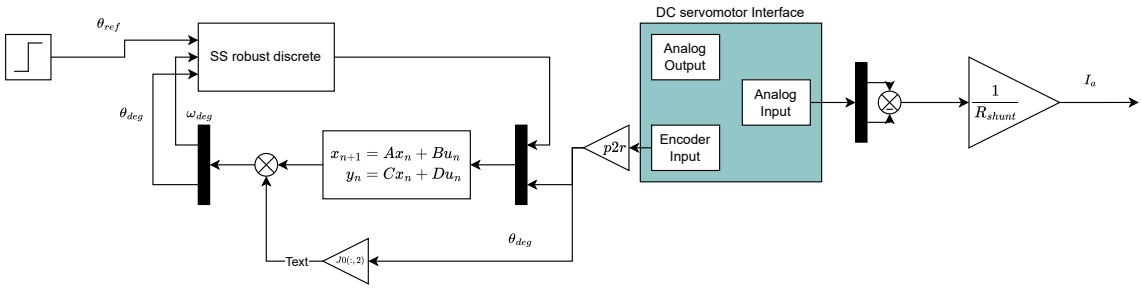

Figure 22: Experimental tests setup