Laboratory activity 2: Digital position control of a DC servomotor

Riccardo Antonello* Francesco Ticozzi*

April 19, 2021

1 Activity goal

The goal of this laboratory activity is to design a digital position controller for the DC servomotor available in the laboratory. Two different design approaches are considered: in the *design by emulation*, the digital controller is obtained by discretization of a controller that is originally designed in the continuous–time domain; vice versa, in the *direct digital design* (or *discrete design*), the control design is performed directly in the discrete–time domain, using a discrete–time model of the plant to be controlled.

LAB 2 CHALLENGE: the only things you need to write down and send over are described in the last subsection of this document. The DEADLINE for this challenge (that counts as a presence test for remote attendance and potential extra point in the final grade) is MAY 3, 2021 before midnight.

*Dept. of Information Engineering (DEI), University of Padova; email: {antonello, ticozzi}@dei.unipd.it

2 Laboratory assignments: numerical simulations

Note: for a correct simulation of a sampled–data control system, observe the following recommendations:

a) Use only discrete–time blocks to implement the discrete–time controller. For each block, the sampling time should be explicitly specified in the *Sample time* field of its *Block parameters* dialog window. Consider to parametrise the sampling time with a variable defined in the workspace, so that it becomes easy to test a same control scheme with different sampling times.

b) There is no specific block for the ideal sampler in Simulink. Although counterintuitive, the conventional way of implementing an ideal sampler in Simulink consists of using a *Discrete* $\rightarrow$ *Zero–Order Hold* block, which is also used (as its name states) to implement the zero–order discrete–to–continuous time interpolator.

c) Simulations can be performed with either a *variable* or *fixed* step solver. However, when using a fixed step solver, note that:

- the step size of the numerical solver should be smaller than the controller sampling time, in order to simulate how the continuous–time plant dynamics evolves between consecutive sampling times (*inter–sample behaviour*). Since the minimum sampling time considered in this activity is equal to $1\,\mathrm{ms}$, a reasonable choice for the maximum step size of the numerical solver is $0.1\,\mathrm{ms}$, which implies that the plant dynamics is evaluated at least on ten different time instants within each sampling period.
- a "rate transition" error could be issued by Simulink when attempting to simulate the sampled–data control system with a step size smaller than the controller sampling time. The error message can be avoided by changing the *Configuration parameters* $\rightarrow$ *Diagnostics* $\rightarrow$ *Sample Time* $\rightarrow$ *Multitask rate transition* option from *error* to *warning*.

2.1 Design by emulation

2.1.1 Position PID–controller

1. Reconsider the position PID–controller for the Quanser SRV–02 servomotor designed in the first laboratory activity, using the motor nominal parameters.

   Obtain a discrete equivalent of the original continuous–time controller, using the **backward Euler** discretization method. Verify that the transfer function of the discrete–time PID controller (with "real derivative") is

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

   Note: for the time constant T_L of the "real derivative" in the PID-controller implementation, use the value $T_L = 1/(2\omega_{gc})$, where ω_{gc} is the gain crossover frequency of the continuous–time control design.

2. Implement the discrete–time PID controller of point 1 on an accurate Simulink model of the experimental system, that includes the friction torque estimated in the first laboratory activity.

Use a *Discrete* $\rightarrow$ *Discrete–Time Integrator* block to implement the PID integrator, making sure to select "*Integration: Backward Euler*" in the *Integrator method* field of the *Block Parameters* dialog window. For the "real derivative" term in (1), use a *Discrete* $\rightarrow$ *Discrete Transfer Fcn* block.

Then, validate the design on simulation, with the following three choices of the sampling time

$$T_1 = 1\,\mathrm{ms}\,, \qquad T_2 = 10\,\mathrm{ms}\,, \qquad T_3 = 50\,\mathrm{ms} \tag{2}$$

For the tests, use a 50° step reference input.

3. On the discrete–time PID controller designed in point 1, implement the integrator anti–windup mechanism described in the previous laboratory activity. Use the same anti–windup gain K_W determined in the previous laboratory activity for the implementation of the discrete–time version of the anti–windup scheme.

   Then, validate the effectiveness of the anti–windup scheme on simulation, using an accurate Simulink model of the experimental setup. Do the tests with a sampling time equal to $T = T_2 = 10\,\mathrm{ms}$, and a step reference input of amplitude equal to 360°. Compare the overshoot in the step response obtained by enabling or disabling the anti–windup scheme.

   Note: the implementation of the discrete–time integrator anti–windup scheme could yield an "algebraic loop". Depending on the version of Simulink used, the presence of an algebraic loop could trigger either a *warning* or an *error* message. One method to remove the algebraic loop consists of inserting a *Discrete* $\rightarrow$ *Unit Delay* in the feedback path of the integrator anti–windup scheme. The introduction of the unit delay breaks the algebraic loop, but at the same time modifies the dynamics of the integrator desaturation. However, this modification is negligible when the sampling time is sufficiently smaller than the dominant time constant of the control loop.

   Alternatively, to avoid the issuing of the algebraic loop error, open the *Simulation* $\rightarrow$ *Model Configuration Parameters* window associated with the Simulink model, and in the *Diagnostic* pane, set *Algebraic loop* to *warning* instead of *error*.

4. (**Optional**) On the discrete–time PID controller designed in point 1, implement the feedforward compensation described in the previous laboratory activity. Use the same feedforward gains determined in the previous laboratory activity (continuos–time design) for the implementation of the discrete–time version of the feedforward compensation scheme.

   Then, validate the effectiveness of the feedforward compensation scheme on simulation, using an accurate Simulink model of the experimental setup. For the tests, set the parameters J_{eq}, B_{eq} and τ_{sf} of the motor model equal to the values estimated in the first laboratory activity. Use a sampling time equal to $T = T_2 = 10\,\mathrm{ms}$; for the acceleration, velocity and position reference signals, consider the discretised versions of the reference signals used for the

continuous–time tests of the previous laboratory activity, namely

$$a_l^*(kT) = \begin{cases} 900\,\mathrm{rpm/s} & \text{if} \quad 0\,\mathrm{s} \le kT < 0.5\,\mathrm{s} \\ 0\,\mathrm{rpm/s} & \text{if} \quad 0.5\,\mathrm{s} \le kT < 1\,\mathrm{s} \\ -900\,\mathrm{rpm/s} & \text{if} \quad 1\,\mathrm{s} \le kT < 2\,\mathrm{s} \\ 0\,\mathrm{rpm/s} & \text{if} \quad 2\,\mathrm{s} \le kT < 2.5\,\mathrm{s} \\ 900\,\mathrm{rpm/s} & \text{if} \quad 2.5\,\mathrm{s} \le kT < 3\,\mathrm{s} \end{cases} \qquad \text{(acceleration reference)} \qquad (3)$$

$$\omega_l^*(kT) = T \sum_{n=0}^{k} a_l^*(nT) \qquad \text{(velocity reference)} \qquad (4)$$

$$\vartheta_l^*(kT) = T \sum_{n=0}^{k} \omega_l^*(nT) \qquad \text{(position reference)} \qquad (5)$$

Compare the tracking error obtained with or without the feedforward compensation scheme.

5. Repeat the points 1 and 2 by changing the discretization method. Consider the following alternative discretization methods:

   a) **Forward Euler** discretization method

   b) **Tustin**'s (also referred as bilinear, or trapezoidal) discretization method

   If you have time, it is also interesting to test how the method used to translate $P(s)$ to an exact discrete equivalent performs in emulating $C(s)$. The latter is referred to as exact or zero–order hold (ZOH) method).

   For obtaining the transfer function $C(z)$ of the discrete equivalents, consult the tables in the Handout. Differently from the forward/backward Euler methods, note that the discretization with the Tustin's and exact methods can be performed by using the `c2d` routine of the Control System Toolbox (CST), making sure to specify either `'tustin'` or `'zoh'` as the discretization method.

   For example, if `sysC` is the LTI object of the continuous–time controller, then `sysCd = c2d(sysC, T, 'tustin')` computes the Tustin's discrete equivalent `sysCd`, and `[numCd, denCd] = tfdata(sysCd, 'v')` extracts the numerator and denominator of the discretised transfer function. These polynomials can be next used to implement the discrete–time controller in Simulink, by using a *Discrete* → *Discrete Transfer Fcn* block.

2.1.2 Position state–space controller

1. Reconsider the position state–space controller for nominal perfect tracking of constant set–points designed in the second laboratory activity. Differently from the second laboratory activity, suppose that the plant state is not fully accessible, so that a state observer is required to get an estimate of all the unmeasurable state variables.

   Since the motor speed is the only unmeasurable state variable, consider to design a *reduced–order* state observer for estimating it. Do the design in the continuous–time domain, by placing the single observer eigenvalue λ_o at a frequency which is 5 times larger than the frequency of the controller eigenvalues.

For the design of the reduced–order state observer, follow the procedure reported in the Handout. Note that the state–space representation of the DC gearmotor introduced in the laboratory activity 2, namely $\Sigma = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ with

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 0 & -\dfrac{1}{T_m} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\dfrac{R_{eq}B_{eq} + k_t k_e}{R_{eq}\, J_{eq}} \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} 0 \\ \dfrac{k_m}{N\, T_m} \end{bmatrix} = \begin{bmatrix} 0 \\ \dfrac{k_{drv}\, k_t}{N\, R_{eq}\, J_{eq}} \end{bmatrix} \tag{6}$$

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

and state vector $\boldsymbol{x} = [\vartheta_l,\ \omega_l]^T$, is already partitioned according to (54)-(55) in the Handout, so that the equations of the reduced–order state observer can be immediately derived by setting $\boldsymbol{T} = \boldsymbol{I}_{n\times n}$ in equations (64) and (68) of the Handout. Verify that the resulting observer is a dynamical system with the following state–space representation:

$$\hat{\Sigma} : \begin{cases} \dot{z} = A_o\, z + \boldsymbol{B}_o\, [u,\, y]^T \\ \hat{\boldsymbol{x}} = \boldsymbol{C}_o\, z + \boldsymbol{D}_o\, [u,\, y]^T \end{cases} \tag{7}$$

where

$$\begin{aligned} A_o &= \left(-\frac{1}{T_m} - L\right) & \boldsymbol{B}_o &= \left[\frac{k_m}{N\, T_m},\ \left(-\frac{1}{T_m} - L\right) L\right] \\ \boldsymbol{C}_o &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \boldsymbol{D}_o &= \begin{bmatrix} 0 & 1 \\ 0 & L \end{bmatrix} \end{aligned} \tag{8}$$

with $L \in \mathbb{R}$ being the observer gain.

The state estimate $\hat{\boldsymbol{x}}$ can be used in place of the actual plant state $\boldsymbol{x}$ to perform the state feedback; the resulting control law becomes

$$u = N_u\, r - \boldsymbol{K}\,(\hat{\boldsymbol{x}} - \boldsymbol{N}_x r) = -\boldsymbol{K}\hat{\boldsymbol{x}} + \underbrace{(N_u + \boldsymbol{K}\boldsymbol{N}_x)}_{=\boldsymbol{N}_r}\, r \tag{9}$$

which can be regarded as a *dynamic output feedback* (instead of a *static state feedback*, as in the case of a fully accessible state vector), since the estimated state $\hat{\boldsymbol{x}}$ is obtained from the measured output with the aid of a dynamical system, namely the state observer. The series connection of a state observer (of whatever type) and a state feedback controller is called *regulator*.

2. Test the **continuous–time**[1] design of point 1 in simulation, using an accurate Simulink model of the experimental system. Verify that the overall control system has a satisfactory response to step reference inputs of different amplitudes, e.g. $40°$, $70°$ and $120°$.

   In Simulink, the reduced–order observer (7) can be implemented by using a *Continuous* → *State–Space* block, with the *A*, *B*, *C* and *D* fields of the *Block Parameters* dialog window set according to the values reported in (8). Use the *Signal Routing* → *Mux* to stack the two observer inputs u and y into a single column vector; if necessary, use a *Signal Routing* → *Demux* to separate the components of the state estimate $\hat{\boldsymbol{x}}$ into single scalar signals.

[1] Note that in this point and in the previous one the controller is assumed to be still in continuous-time.

3. Obtain a discrete equivalent of the continuous–time regulator designed in point 1, using the **forward Euler** discretization method.

   The discretised regulator simply consists of the discretization of (7), combined with the same control law (9) of the continuous–time case (which is obviously evaluated only at the sampling instants of the discrete–time controller).

   Verify that the forward Euler discretization of (7) is equal to[2]

$$\hat{\Sigma}_d \; : \; \begin{cases} z[k+1] \; = \; \Phi_o \, z[k] \; + \; \boldsymbol{\Gamma}_o \; [u[k], \, y[k]]^T \\ \quad \hat{\boldsymbol{x}}[k] \; = \; \boldsymbol{H}_o \, z[k] \; + \; \boldsymbol{J}_o \; [u[k], \, y[k]]^T \end{cases} \tag{10}$$

   with

$$\begin{aligned} \Phi_o \; &= \; 1 \; + \; A_o \, T & \qquad \boldsymbol{\Gamma}_o \; &= \; \boldsymbol{B}_o \, T \\ \boldsymbol{H}_o \; &= \; \boldsymbol{C}_o & \qquad \boldsymbol{J}_o \; &= \; \boldsymbol{D}_o \end{aligned} \tag{11}$$

4. Test the discrete–time regulator of point 3 in simulation, using an accurate Simulink model of the experimental system. Validate the design with the three choices of the sampling time reported in (2). For the tests, use a 50° step reference input.

   In Simulink, the discrete–time reduced–order observer (10) can be implemented by using a *Discrete* → *Discrete State–Space* block, with the *A*, *B*, *C* and *D* fields of the *Block Parameters* dialog window set according to the values reported in (11). Use the *Signal Routing* → *Mux* to stack the two observer inputs u and y into a single column vector; if necessary, use a *Signal Routing* → *Demux* to separate the components of the state estimate $\hat{\boldsymbol{x}}$ into single scalar signals.

5. Reconsider the position state–space controller with integral action designed in the second laboratory activity, for achieving robust perfect tracking of constant set–points. Modify the control scheme to include the reduced–order state observer designed in point 1. The modified control law becomes

$$\begin{cases} \dot{x}_I \; = \; y \; - \; r \\ u \; = \; -K_I \, x_I \; - \; \boldsymbol{K} \, \hat{\boldsymbol{x}} \; + \; \boldsymbol{N}_r \, r \end{cases} \tag{12}$$

   where $\hat{x}$ is the output of the observer (7).

   Then, test the modified continuous–time control scheme in simulation, using an accurate model of the experimental system. Verify that the overall control scheme has a satisfactory response to step reference inputs of different amplitudes, e.g. 40°, 70° and 120°.

6. Obtain a discrete equivalent of the continuous–time regulator designed in point 5, using the forward Euler discretization method.

   The discretised regulator consists of the discrete–time observer (10), combined with the discretised version of (12), which is equal to

$$\begin{cases} x_I[k] \; = \; x_I[k-1] \; + \; T \, (y[k-1] \; - \; r[k-1]) \\ u[k] \; = \; -K_I \, x_I[k] \; - \; \boldsymbol{K} \, \hat{\boldsymbol{x}}[k] \; + \; \boldsymbol{N}_r \, r[k] \end{cases} \tag{13}$$

[2] The notation $x[k]$ will be used throughout the whole handout to denote the evaluation of the signal x at the sampling instant $t = kT$, namely $x[k] \triangleq x(kT)$.

7. Test the discrete–time regulator of point 6 in simulation, using an accurate Simulink model of the experimental system. Validate the design with the three choices of the sampling time reported in (2). For the tests, use a 50° step reference input.

   For the Simulink implementation of the discrete–time reduced–order observer (10), follow the indications given in point 4. For the implementation of the discrete–time integral action in (13), use a *Discrete* $\rightarrow$ *Discrete–Time Integrator* block, making sure to select "*Integration: Forward Euler*" in the *Integrator method* field of the *Block Parameters* dialog window.

8. Repeat the points 3–4 and 6–7 by using the **exact discretization** method.

   For obtaining the state–space model $\Sigma_d = (\boldsymbol{\Phi}, \boldsymbol{\Gamma}, \boldsymbol{H}, \boldsymbol{J})$ of the discrete equivalents, consult Tab. 3 of the Handout. Differently from the forward/backward Euler methods, note that the discretization with the Tustin's and exact methods can be performed by using the `c2d` routine of the Control System Toolbox (CST), making sure to specify either `'tustin'` or `'zoh'` as the discretization method.

   For example, if `sysC` is the LTI object of the continuous–time controller, then `sysCd = c2d(sysC, T, 'zoh')` computes the exact discrete equivalent `sysCd`, and `[Phi, Gam, H, J] = ssdata(sysCd)` extracts the state–space matrices $\boldsymbol{\Phi}$, $\boldsymbol{\Gamma}$, $\boldsymbol{H}$ and $\boldsymbol{J}$ of the discretised state–space model. These matrices can be next used to implement the discrete–time controller in Simulink, by using a *Discrete* $\rightarrow$ *Discrete State-Space* block.

9. (**Optional**) Repeat the points 3–4 and 6–7 by changing the discretization method. Consider the following alternative discretization methods:

   a) **Backward Euler** discretization method

   b) **Tustin**'s (also referred as bilinear, or trapezoidal) discretization method

   Verify that the discretised models obtained with these two methods cannot be actually implemented in practice. In fact, when using the backward Euler or the trapezoidal method, the $\boldsymbol{J}$ matrix of the discretised model has a non–null column associated with the input u, so that the observer output $\hat{\boldsymbol{x}}[k]$ can be computed at a certain sampling instant k only if the control input $u[k]$ is known at the same instant. But since the control input $u[k]$ is determined as a static feedback from the estimated state $\hat{\boldsymbol{x}}[k]$, an algebraic loop is formed between the observer output and input, and this finally prevents the implementation of the discrete–time observer.

2.2 Direct digital design

1. Discretise the continuous–time state–space model $\Sigma = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, D)$ of the DC gearmotor using the *exact method*, to obtain the "ZOH discrete equivalent" $\Sigma_d = (\boldsymbol{\Phi}, \boldsymbol{\Gamma}, \boldsymbol{H}, J)$. Note that since $D = 0$, then also $J = 0$. This condition will be considered throughout all this section.

   The discretization can be performed by using the `c2d` routine of the Control System Toolbox (CST), making sure to specify `'zoh'` as the discretization method.

2. Design a discrete–time position state–space controller for *nominal* perfect tracking of constant position set–points, using the discretised plant model Σ_d obtained in point 1.

Since the motor speed is not directly measurable, consider to design a discrete–time reduced–order observer for estimating it. The speed estimate, together with the motor position provided by the encoder, will be used to perform the state feedback.

For the design of the reduced–order state observer, follow the indications reported in Sec. **??**. Verify that the observer matrices are computed according to the following expressions

$$\begin{aligned} \Phi_o &= \Phi_{22} - L\,\Phi_{12} \qquad & \boldsymbol{\Gamma}_o &= \Big[\Gamma_2 - L\,\Gamma_1\,, \quad (\Phi_{22} - L\,\Phi_{12})\,L + \Phi_{21} - L\,\Phi_{11}\Big] \\ \boldsymbol{H}_o &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \boldsymbol{J}_o &= \begin{bmatrix} 0 & 1 \\ 0 & L \end{bmatrix} \end{aligned} \tag{14}$$

where

$$\boldsymbol{\Phi} = \begin{bmatrix} \Phi_{11} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{bmatrix} \qquad \boldsymbol{\Gamma} = \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \end{bmatrix} \tag{15}$$

under the assumption that $\boldsymbol{x} = [\vartheta_l,\, \omega_l]^T$ is the plant state, with the first state component being the measured variable (i.e. the load position ϑ_l).

In (14), L is the observer gain, which is designed to place the eigenvalues of Φ_o (which governs the dynamics of the estimation error) to certain desired locations on the complex plane. Note that for the specific example under consideration, the matrix Φ_o is indeed a real scalar, since the reduced–order observer has only one state variable (i.e. the variable z – see (10)). Moreover, note that the gain L in (14) is in general different from that appearing in (8).

The structure of the control law remains identical to (9). However, note that the feedforward gains $\boldsymbol{N}_x$ and N_u must be computed by considering the conditions for an equilibrium state of a discrete–time system. In practice, denote with $\boldsymbol{x}_\infty$, u_∞ and y_∞ the steady state values of the plant state, input and output variables. Then, the perfect steady state tracking condition $y_\infty = r_\infty$ imposes that $\boldsymbol{x}_\infty$ and u_∞ satisfy the conditions

$$\begin{cases} \boldsymbol{\Phi}\,\boldsymbol{x}_\infty + \boldsymbol{\Gamma}\,u_\infty = \boldsymbol{x}_\infty & (16) \\ \boldsymbol{H}\,\boldsymbol{x}_\infty = y_\infty = r_\infty & (17) \end{cases}$$

where (16) is the condition to impose for specifying that $\boldsymbol{x}_\infty$ is an equilibrium state for the discrete–time model $\Sigma_d = (\boldsymbol{\Phi},\, \boldsymbol{\Gamma},\, \boldsymbol{H},\, 0)$ with input u_∞. From (16) and (17) it follows that both $\boldsymbol{x}_\infty$ and u_∞ depend linearly on r_∞, namely:

$$\boldsymbol{x}_\infty = \boldsymbol{N}_x\,r_\infty\,, \qquad u_\infty = N_u\,r_\infty \tag{18}$$

After replacing (18) within (16) and (17), it is immediate to verify that the two gains $\boldsymbol{N}_x \in \mathbb{R}^{n\times 1}$ and $N_u \in \mathbb{R}$ can be determined by solving the following system of linear equations

$$\begin{bmatrix} \boldsymbol{\Phi} - \boldsymbol{I} & \boldsymbol{\Gamma} \\ \boldsymbol{H} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ N_u \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix} \tag{19}$$

Regarding the design of the feedback gain $\boldsymbol{K}$, it is desired to design a discrete–time regulator that has, at the sampling instants, the same response of the continuous–time regulator designed in point 1 of Sec. 2.1.2. For such purpose, the discrete–time controller and observer eigenvalues

must be chosen as the z–plane images, according to the transformation $z = e^{s\,T}$, of the corresponding continuous–time eigenvalues (located on the s–plane).

3. Test the discrete–time regulator of point 2 in simulation, using an accurate Simulink model of the experimental system. Validate the design with the three choices of the sampling time reported in (2). For the tests, use a 50° step reference input. Compare the responses with those obtained in point 4 of point Sec. 2.1.2.

   Regarding the Simulink implementation of the discrete–time reduced–order observer, use a *Discrete* $\rightarrow$ *Discrete State–Space* block, with the *A*, *B*, *C* and *D* fields of the *Block Parameters* dialog window set according to the values reported in (14). Use the *Signal Routing* $\rightarrow$ *Mux* to stack the two observer inputs u and y into a single column vector; if necessary, use a *Signal Routing* $\rightarrow$ *Demux* to separate the components of the state estimate $\hat{\boldsymbol{x}}$ into single scalar signals.

4. Design a discrete–time position state–space controller with integral action to achieve *robust* perfect tracking of constant position set–points, and perfect rejection of constant load disturbances. For the design, use the discretised plant model Σ_d obtained in point 1, and a discrete–time reduced–order observer similar to that obtained in point 2.

   The discrete–time control law with integral action (and dynamic output feedback, i.e. feedback from the *estimated* state) can be written as follows

$$\begin{cases} x_I[k+1] \;=\; x_I[k] \,+\, (\, y[k] \,-\, r[k]\,) \\ \qquad u[k] \;=\; -K_I\, x_I[k] \,-\, \boldsymbol{K}\, \hat{\boldsymbol{x}}[k] \,+\, \boldsymbol{N}_r\, r[k] \end{cases} \tag{20}$$

   where the reference feedforward gain $\boldsymbol{N}_r = N_u + \boldsymbol{K}\boldsymbol{N}_x$ is obviously computed with the feedforward gains obtained in point 2 of Sec. 2.2. As done in the continuous–time case, the integrator state x_I can be combined with the plant state $\boldsymbol{x}$ to form the state $\boldsymbol{x}_e = [x_I,\, \boldsymbol{x}]^T$ of the augmented–state system

$$\Sigma_e \;:\; \begin{bmatrix} x_I[k+1] \\ \boldsymbol{x}[k+1] \end{bmatrix} \;=\; \underbrace{\begin{bmatrix} 1 & \boldsymbol{H} \\ \boldsymbol{0} & \boldsymbol{\Phi} \end{bmatrix}}_{\triangleq\, \boldsymbol{\Phi}_e} \underbrace{\begin{bmatrix} x_I[k] \\ \boldsymbol{x}[k] \end{bmatrix}}_{\triangleq\, \boldsymbol{x}_e} + \underbrace{\begin{bmatrix} 0 \\ \boldsymbol{\Gamma} \end{bmatrix}}_{\triangleq\, \boldsymbol{\Gamma}_e} u[k] \,-\, \begin{bmatrix} 1 \\ \boldsymbol{0} \end{bmatrix} r[k] \tag{21}$$

   Thanks to the *separation principle*, the designs of the state–feedback controller and the state observer can be carried out separately. The controller is designed as the state $\boldsymbol{x}_e$ were fully accessible, so that the second of (20) can be replaced by $u[k] = -\boldsymbol{K}_e\, \boldsymbol{x}_e[k]$, with $\boldsymbol{K}_e = [K_I,\, \boldsymbol{K}]$. Then, the state feedback gain $\boldsymbol{K}_e$ is chosen so that eigenvalues of closed–loop system, namely the eigenvalues of

$$\boldsymbol{\Phi}_{cl} \;\triangleq\; \boldsymbol{\Phi}_e \,-\, \boldsymbol{\Gamma}_e\, \boldsymbol{K}_e \;=\; \begin{bmatrix} 1 & \boldsymbol{H} \\ -\boldsymbol{\Gamma}\, K_I & \boldsymbol{\Phi} - \boldsymbol{\Gamma}\, \boldsymbol{K} \end{bmatrix} \tag{22}$$

   are located to certain desired locations. For what regards the observer design, the gain L is selected so that the matrix $\Phi_o = \Phi_{22} - L\,\Phi_{12}$ in (14) (which governs the dynamics of the estimation error) has the eigenvalues to certain desired locations. Note that for the specific example under consideration, the matrix Φ_o is indeed a real scalar, since the reduced–order observer has only one state variable (i.e. the variable z – see (10)).

As done in point 2, place the controller and observer eigenvalues to the positions obtained by mapping the eigenvalues of the continuous–time design of point 5 of Sec. 2.1.2 on the z–plane, using the transformation $z = e^{sT}$.

5. Test the discrete–time regulator of point 4 in simulation, using an accurate Simulink model of the experimental system. Validate the design with the three choices of the sampling time reported in (2). For the tests, use a 50° step reference input. Compare the responses with those obtained in point 7 of point Sec. 2.1.2.

   For the Simulink implementation of the discrete–time reduced–order observer (10), follow the indications given in point 3. For the implementation of the discrete–time integral action in (20), use a *Discrete* $\rightarrow$ *Discrete–Time Integrator* block, making sure to select "*Accumulation: Forward Euler*" in the *Integrator method* field of the *Block Parameters* dialog window.

3 Laboratory assignments

In the following, you will be required to validate your designs with either the actual motor in the laboratory (LAB1 activity) or using the black-box motor model if you chose the remote attendance mode. As in the previous assignment, you will need to modify and adapt your simulink files:

– **For in-presence attendance only:** replace the model of the DC servomotor with the blocks of the Simulink Desktop Real–Time (SLDRT) toolbox that allow to communicate with the experimental device, as you did in the previous experience.

  In addition to send the voltage command to the DAC (using the *Analog Output* block) and read the pulse count from the encoder (using the *Encoder Input* block), consider also to implement a block for sensing the motor current. Follow the instructions provided in the introductory guide to the experimental setup to implement such block.

– configure the simulation parameters of the new Simulink model to perform a "real–time simulation". For the purpose, consult the instructions provided in the introductory guide to the experimental setup.

– **For remote attendance only:** replace the model of the DC servomotor with the motor simulink block we provide in the `Quanser_SRV02_block.zip` files - select the build that matches your operating system (Win, OSX or UNIX), use the `.slx` file for the simulink model, and keep the `.slxp` file in the same directory. From now on this will be considered you experimental device[3]. Thus, it has one input (the voltage input to the driver, `u`, expected in volts V, and two outputs: thl_meas that corresponds to the output of the encoder and hence it is in pulses, and uRs_meas that corresponds to the voltage at the shunt resistor of the motor armature. The latter allows you to implement a block for sensing the motor current. Follow the instructions provided in the introductory guide to the experimental setup to implement such block.

[3] It has been constructed to match the behaviour of one of the motors in the lab so do not expect a perfect match between simulations of the model we derived in class and this one, even when you use the estimated friction parameters.

Note: for a correct execution of the experimental tests, all the Simulink models used for the "real–time simulation" of a *discrete–time controller* must be prepared according to the following recommendations:

a) the Simulink model must contain only discrete–time blocks. For each block, the sampling time should be explicitly specified in the *Sample time* field of its *Block Parameters* dialog window. Consider to parametrise the sampling time with a variable defined in the workspace, so that it becomes easy to test the same control scheme with different sampling times.

b) in the *Model Configuration Parameters* window, in the *Solver options* section, set *Type* to *Fixed–step* and *Solver* to *discrete (no continuous state)*. The latter choice corresponds to specify that no numerical integration is required to evaluate the Simulink model, since it does not contain any continuous–time block.

   The *Fixed–step size* field specifies the controller sampling time: set it to the appropriate value, depending on the test under consideration.

3.1 Design by emulation

3.1.1 Position PID–controller

Validate the control designs listed below on the experimental setup:

1. the discrete–time PID controller obtained in point 1 of Sec. 2.1.1 by using the backward Euler discretization method.
2. the discrete–time anti–windup scheme designed in point 3 of Sec. 2.1.1.
3. (**Optional**) the discrete–time feedforward compensation scheme designed in point 4 of Sec. 2.1.1.
4. the alternative discrete–time PID controllers obtained in point 5 of Sec. 2.1.1 by using the forward, trapezoidal and exact discretization methods.

Perform each experimental test by adopting the same methodology used for the corresponding numerical simulation.

3.1.2 Position state–space controller

Validate the control designs listed below on the experimental setup:

1. the discrete–time position state–space controller for nominal tracking of constant set–points obtained in point 3 of Sec. 2.1.2 by using the forward Euler discretization method.
2. the discrete–time position state–space controller with integral action obtained in point 6 of Sec. 2.1.2 by using the forward Euler discretization method.
3. the alternative discrete–time position state–space controllers obtained in point 8 of Sec. 2.1.2 using the exact discretization method.

Perform each experimental test by adopting the same methodology used for the corresponding numerical simulation. If necessary, modify the observer design (i.e. change the location of the observer

eigenvalue) to get a satisfactory control response from the two experimental tests of points 1 and 2.

Note: in the real–time simulations, an algebraic loop error is issued by Simulink if the reduced–order observer is implemented by using a *Discrete State–Space* block with a $\boldsymbol{D}$ matrix (the $\boldsymbol{J}_o$ matrix in (14)) different from zero. To avoid such error, consider to implement the reduced–order observer as illustrated in Fig. 1, where the matrices of the Discrete State–Space block are $(\boldsymbol{\Phi}_o, \boldsymbol{\Gamma}_o, \boldsymbol{H}_o, \boldsymbol{0})$, while the external gain block coincides with the second column of the matrix $\boldsymbol{J}_o$ defined in (14).

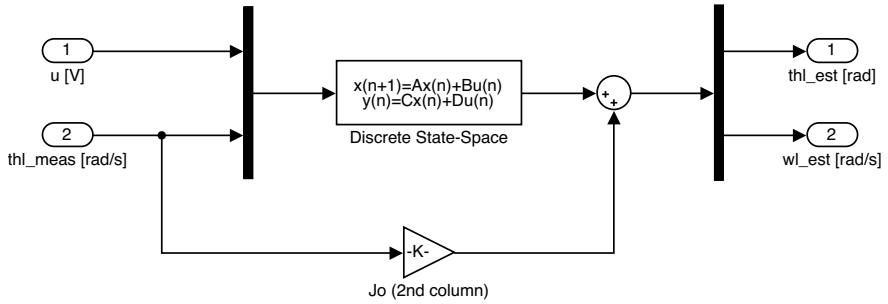

Figure 1: Possible implementation of the reduced–order observer for the real–time simulations.

3.2 Direct digital design

Validate the control designs listed below on the experimental setup:

1. the discrete–time position state–space controller designed in point 2 of Sec. 2.2 to achieve nominal perfect tracking of constant set–points.

2. the discrete–time position state–space controller designed in point 4 of Sec. 2.2 to achieve robust perfect tracking of constant set–points, by using the integral action.

Perform each experimental test by adopting the same methodology used for the corresponding numerical simulation. If necessary, modify the observer design (i.e. change the location of the observer eigenvalue) to get a satisfactory control response from the two experimental tests of points 1 and 2.

3.3 LAB 2 CHALLENGE

Design, using one or a combinations of the techniques presented in the course, a digital control system for the QUANSER SRV-02 MODEL such that:

- **It ensures asymptotic tracking of step references;**
- **It ensures an overshoot $M_p \leq 10\%$ for a 70deg step reference;**
- **It attains a settling time $t_{s,5\%} \leq 0.15$s for the same reference;**
- **It employs the longest possible sampling time T_S.**

A pdf file (2 pages max.) including:

- **the description of the control structure and strategy;**
- **the parameters and values needed to implement the controller;**
- **a plot of the response to the 70 degree step;**
- **the corresponding largest value of T_s you attained;**

should be sent to `ticozzi@dei.unipd.it` **with subject** `[CONTROL LAB] LAB2 RESULTS GROUP "x,d"` **by May 3, 2021 (before midnight).**