Laboratory activity 4: Longitudinal state–space control of the balancing robot

Riccardo Antonello* Francesco Ticozzi*

May 17, 2021

1 Activity goal

The purpose of this laboratory activity is to design and test a longitudinal state–space controller for the two–wheeled balancing robot (also referred as "two–wheeled inverted pendulum robot", or "Segway–like robot") available in laboratory. The controller is designed to simultaneously stabilize the robot body to its upward vertical position, and the robot base to a desired longitudinal position set-point. The design is performed by resorting to a simplified model of the robot dynamics, obtained by assuming that the motion occurs along a straight line (i.e. the lateral or heading–angle dynamics is ignored).

There is no challenge for this LAB. Students attending remotely only just need to send a pdf file with the results from experimental assignment in Section 6.2

2 Analytical model of the balancing robot

A mathematical model for the longitudinal dynamics of the balancing robot is derived in this section. The model is valid under the simplifying assumption that the robot moves along a straight line, namely no lateral motion occurs (i.e. the heading–angle is constant).

2.1 Mechanical model

The two–wheeled balancing robot can be represented as a multi–body system that comprises the following rigid bodies (see also Fig. 1):

- *Wheels* (left/right).
- *DC gearmotor rotors* (left/right).
- *Robot body*. It comprises the *robot chassis* (includes all the electronic boards and the motor supporting brackets), the *DC gearmotor stators* (left/right) and the *battery*.

The following reference frames are introduced to describe the robot structure and configuration (see also Fig. 1 ÷ 4):

- *Body frame*. The frame is located on the wheels rotation axis, and is centered in the midpoint

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

between the two wheels. The frame is rigidly attached with the robot chassis. Its y–axis is directed along the wheel rotation axis, and points toward the left wheel. The z–axis passes through the body Center–of–Mass (CoM), and points toward it. The x–axis direction and orientation is determined by the right–hand rule. The body frame is denoted with $\{b\}$.

- *Vehicle frame*. The frame center and y–axis are the same as the body frame. However, the frame is not rigidly attached with the robot body; instead, its z–axis is always oriented oppositely to the acceleration of gravity. The x–axis direction and orientation is determined by the right–hand rule. The vehicle frame is denoted with $\{v\}$.
- *World (or earth) frame*. Its pose (position and orientation) is ground–fixed, and coincides with the initial pose of the vehicle frame. The world frame is denoted with $\{o\}$.
- *Wheel frames* (left/right). Each frame is rigidly attached with the wheel body, and centered on its CoM. The axes are initially aligned with those of the vehicle frame. The left and right wheel frames are denoted with, respectively, $\{w, l\}$ and $\{w, r\}$.
- *Rotor frames* (left/right). Each frame is rigidly attached with the rotor body, and centered on its CoM. The axes are initially aligned with those of the vehicle frame. The left and right rotor frames are denoted with, respectively, $\{rot, l\}$ and $\{rot, r\}$.

The geometrical and inertial parameters of each body (and its possible subparts) are provided in Tab. 1. Note that:

- The masses of the DC gearmotor rotor and stator are estimated as, respectively, the 35% and 65% of the whole motor mass (equal to $m_{mot} = 215\,\mathrm{g}$).
- The total mass of the robot body is computed as the sum of the chassis, battery and motor stators masses, namely

$$m_b \;=\; m_c \;+\; m_{batt} \;+\; 2\,m_{stat} \tag{1}$$

- The stator CoM position has been determined after noting (experimentally) that the motor CoM is displaced by $6\,\mathrm{mm}$ off the geometrical center, in the direction of motor gearbox.
- The body CoM z–axis coordinate z_b^b in the body frame is determined as follows

$$z_b^b \;=\; \frac{1}{m_b}\left(m_c\,z_c^b \;+\; m_{batt}\,z_{batt}^b \;+\; 2\,m_{stat}\,z_{stat}^b\right) \tag{2}$$

  The other two coordinates, are equal to zero because of geometrical symmetry reasons.

- The Moments–of–Inertia (MoI) of both the robot chassis and battery are determined by assuming that the bodies are solid parallelepipeds with uniform mass distribution. They are computed with respect to a frame that is centered on the corresponding body CoM, and has axes aligned with, respectively, the parallelepiped width, height and depth directions, namely:

$$I_{xx} = \frac{m}{12}\left(w^2 + h^2\right)\,, \qquad I_{yy} = \frac{m}{12}\left(d^2 + h^2\right)\,, \qquad I_{zz} = \frac{m}{12}\left(w^2 + d^2\right) \tag{3}$$

  where w, h and d are the parallelepiped dimensions (width, height and depth), and m the body mass.

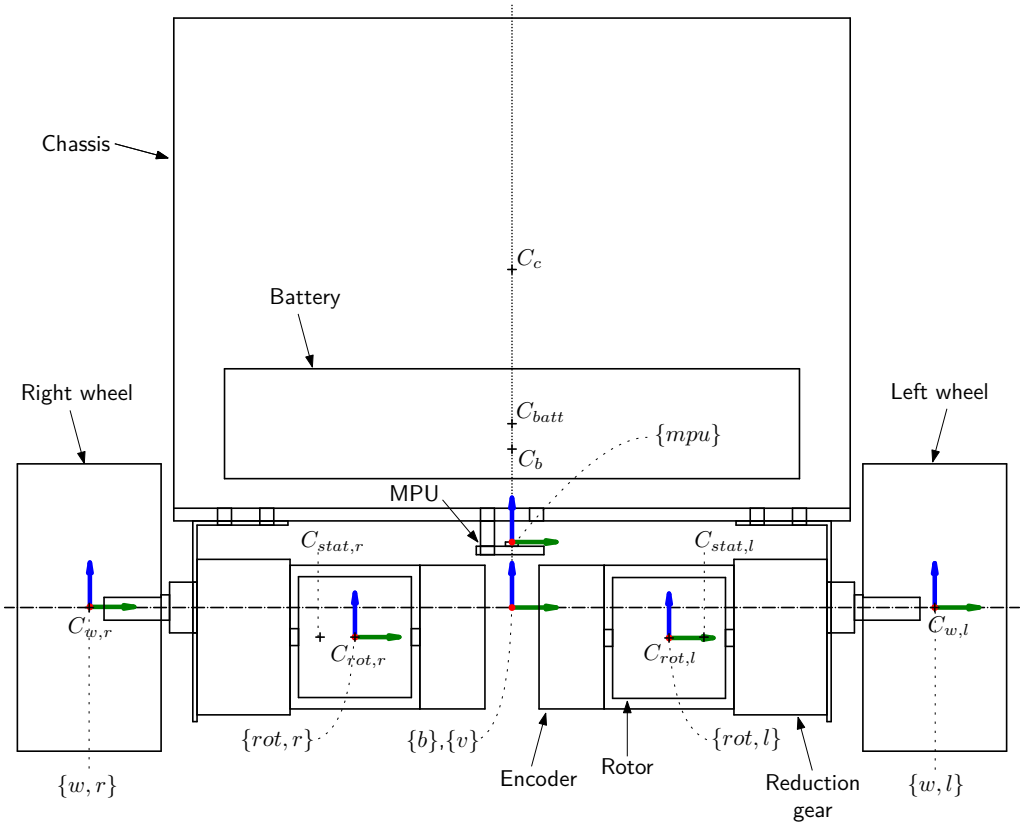
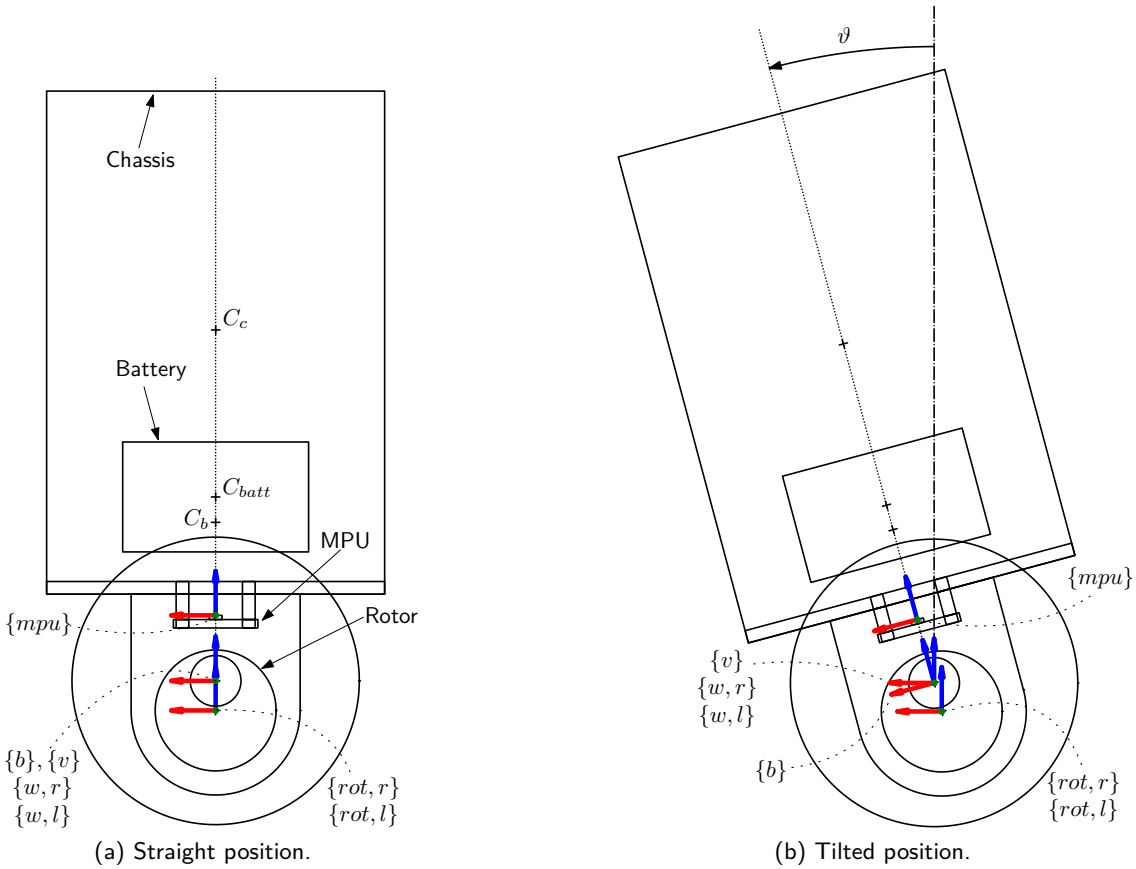


Figure 1: Front view.



(a) Straight position.

(b) Tilted position.

Figure 2: Side view.

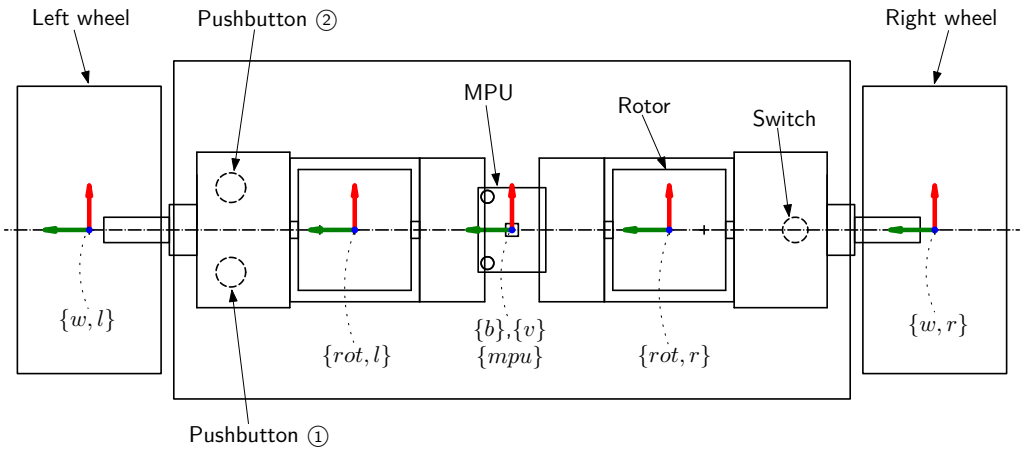


Figure 3: Top view.

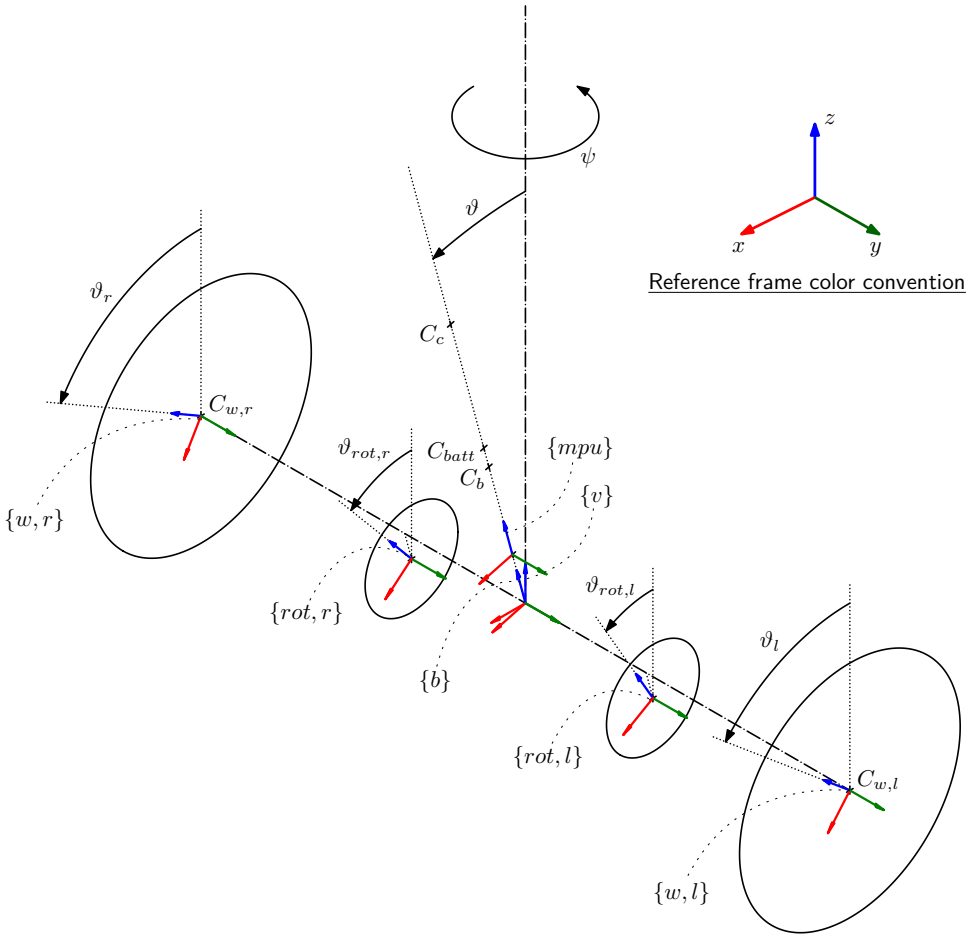


Figure 4: Simplified 3D view.

• **Robot body**			
Center–of–Mass coords wrt body frame $\{b\}$	$x_b^b,\ y_b^b,\ z_b^b$	$0,\ 0,\ 46.05$	[mm]
Mass	m_b	1.06	[kg]
Principal Moments–of–Inertia	$I_{b,xx},\ I_{b,yy},\ I_{b,zz}$	$4.22,\ 2.20,\ 2.65$	[gm^2]
↳ **Robot chassis**			
Dimensions (width, height, depth)	$w_c,\ h_c,\ d_c$	$160,\ 119,\ 80$	[mm]
Center–of–Mass coords wrt body frame $\{b\}$	$x_c^b,\ y_c^b,\ z_c^b$	$0,\ 0,\ 80$	[mm]
Mass	m_c	456	[g]
Principal Moments–of–Inertia	$I_{c,xx},\ I_{c,yy},\ I_{c,zz}$	$1.5,\ 0.78,\ 1.2$	[gm^2]
↳ **Battery**			
Dimensions (width, height, depth)	$w_{batt},\ h_{batt},\ d_{batt}$	$136,\ 26,\ 44$	[mm]
Center–of–Mass coords wrt body frame $\{b\}$	$x_{batt}^b,\ y_{batt}^b,\ z_{batt}^b$	$0,\ 0,\ 44$	[mm]
Mass	m_{batt}	320	[g]
Principal Moments–of–Inertia	$I_{batt,xx},\ I_{batt,yy},\ I_{batt,zz}$	$0.51,\ 0.07,\ 0.06$	[gm^2]
↳ **DC gearmotor stator**			
Dimensions (height, radius)	$h_{stat},\ r_{stat}$	$68.1,\ 17$	[mm]
Center–of–Mass coords wrt body frame $\{b\}$	$x_{stat}^b,\ y_{stat}^b,\ z_{stat}^b$	$0,\ \pm 52.1,\ -7$	[mm]
Mass	m_{stat}	139.75	[g]
Principal Moments–of–Inertia	$I_{stat,xx} = I_{stat,zz},\ I_{stat,yy}$	$0.064,\ 0.02$	[gm^2]

• **DC gearmotor rotor**			
Dimensions (height, radius)	$h_{rot},\ r_{rot}$	$30.7,\ 15.3$	[mm]
Center–of–Mass coords wrt body frame $\{b\}$	$x_{rot}^b,\ y_{rot}^b,\ z_{rot}^b$	$0,\ \pm 42.7,\ -7$	[mm]
Mass	m_{rot}	75.25	[g]
Principal Moments–of–Inertia	$I_{rot,xx} = I_{rot,zz},\ I_{rot,yy}$	$0.01,\ 0.009$	[gm^2]

• **Wheels**			
Dimensions (height, radius)	$h_w,\ r_w$	$26,\ 34$	[mm]
Center–of–Mass coords wrt body frame $\{b\}$	$x_w^b,\ y_w^b,\ z_w^b$	$0,\ \pm 100,\ 0$	[mm]
Mass	m_w	50	[g]
Principal Moments–of–Inertia	$I_{w,xx} = I_{w,zz},\ I_{w,yy}$	$0.017,\ 0.029$	[gm^2]

Table 1: Geometrical and inertial nominal parameters.

- The MoI of both the wheel and DC gearmotor rotor/stator are determined by assuming that the bodies are solid cylinders with uniform mass distribution. They are computed with respect to a frame that is centered on the corresponding body CoM, and has axes aligned with, respectively, the cylinder height and radial directions, namely:

$$I_{xx} = I_{zz} = \frac{m}{12}\left(3r^2 + h^2\right), \qquad I_{yy} = \frac{m\,r^2}{2} \tag{4}$$

  where h and r are the cylinder dimensions (height and radius), and m the body mass.

- The MoI of the robot body are determined with respect to a reference frame ($\{b'\}$) centered on the robot body CoM, and aligned with the body frame $\{b\}$. They are computed by resorting to the *Huygens–Steiner theorem* (or *parallel axis theorem*), which states that the moment of inertia I' of a body rotating about an axis z' displaced from the body CoM by a distance d is equal to $I' = I + md^2$, where I is the body moment of inertia with respect to an axis z parallel to z' and passing through the body CoM. It holds that:

$$\begin{aligned}
I_{b,xx} &= I_{c,xx} + m_c\,(z_b^b - z_c^b)^2 + I_{batt,xx} + m_{batt}\,(z_b^b - z_{batt}^b)^2 + \cdots \\
&\quad \cdots + I_{stat,xx} + m_{stat}\left[(y_b^b - y_{stat,r}^b)^2 + (z_b^b - z_{stat,r}^b)^2\right] + \cdots \\
&\quad \cdots + I_{stat,xx} + m_{stat}\left[(y_b^b - y_{stat,l}^b)^2 + (z_b^b - z_{stat,l}^b)^2\right] \\
I_{b,yy} &= I_{c,yy} + m_c\,(z_b^b - z_c^b)^2 + I_{batt,yy} + m_{batt}(z_b^b - z_{batt}^b)^2 + \cdots \\
&\quad \cdots + 2\,I_{stat,yy} + 2\,m_{stat}\,(z_b^b - z_{stat}^b)^2 \\
I_{b,zz} &= I_{c,zz} + I_{batt,zz} + 2\,I_{stat,zz} + 2m_{stat}\,(y_{stat}^b)^2
\end{aligned} \tag{5}$$

To describe the robot configuration, the following set of *generalized coordinates* is introduced (see also Fig. 4):

- *robot position* $\boldsymbol{p}_v^o = (x_v,\, y_v,\, z_v)$: coordinates of the vehicle frame origin with respect to the world frame.
- *robot tilt angle* ϑ : pitch angle of the body frame with respect to the vehicle frame.
- *wheels angles* ϑ_l and ϑ_r : pitch angle of the wheel frame with respect to the vehicle frame.

Some additional coordinates are introduced to ease the derivation of the dynamical model:

- *robot heading angle* ψ : yaw angle of the vehicle frame with respect to the world frame.
- *rotor angles* $\vartheta_{rot,l}$ and $\vartheta_{rot,r}$: pitch angle of the rotor frames with respect to the vehicle frame.

However, these latter coordinates are not independent variables. In fact, they are directly related to the previously defined generalized coordinates because of the presence of the following kinematic constraints:

- *gearbox mechanical coupling*: let

$$\Delta\vartheta_l = \vartheta_l - \vartheta\,, \qquad \Delta\vartheta_{rot,l} = \vartheta_{rot,l} - \vartheta \tag{6}$$

denote the angular displacements of the left wheel and rotor with respect to the body frame. Since the motor stator is rigidly attached to the body frame, then $\Delta\vartheta_{rot,l}$ and $\Delta\vartheta_l$ are the angles by which the rotor and the output shaft of the gearmotor rotate with respect to the stator. The mechanical coupling due to the gearbox imposes that

$$\Delta\vartheta_{rot,l} \;=\; N\,\Delta\vartheta_l \tag{7}$$

where N denotes the gearbox ratio. By replacing (7) within (6), it follows that:

$$\vartheta_{rot,l} \;=\; \vartheta \;+\; N\,(\vartheta_l \;-\; \vartheta) \tag{8}$$

A similar expression holds for the right rotor angle $\vartheta_{rot,r}$.

- *pure rolling* and *no side–slip wheel constraints*: it is assumed that the velocity of the center of the wheel is parallel to the wheel sagittal plane (*no side–slip condition*) and is proportional to the wheel rotation velocity (*pure rolling condition*). It is immediate to verify that these conditions imply that:

$$v_l \;=\; r\,\dot{\vartheta}_l\,, \qquad v_r \;=\; r\,\dot{\vartheta}_r\,, \qquad w\,\dot{\psi} \;=\; v_r - v_l \tag{9}$$

where $w = 2|y_w^b|$ is the distance between the two wheel CoM, r is the wheel radius, and v_l and v_r the left and right wheel velocities (parallel to the wheel sagittal plane, and hence orthogonal to the wheels axle). From (9) it follows that:

$$\dot{\psi} \;=\; \frac{r}{w}\left(\dot{\vartheta}_r - \dot{\vartheta}_l\right) \qquad \Rightarrow \qquad \psi \;=\; \psi(0) \;+\; \frac{r}{w}\,(\vartheta_r - \vartheta_l) \tag{10}$$

where $\psi(0) = 0$ because the vehicle and world frame are aligned at $t = 0$.

In this handout, the analysis is restricted to the longitudinal dynamics only, provided that the robot moves along a straight line, i.e. the heading angle ψ is a constant. From (10), this is equivalent to state that the two wheels angles ϑ_l and ϑ_r are always identical: in the following, γ will be used to denote the common value of the two wheels angles. Without loss of generality, it can be assumed that $\psi = 0$, so that $y_v = 0$ and the motion remains confined on the xz–plane of the world frame $\{o\}$. On such plane, the robot dynamics is equivalent to that of a planar robot with a single wheel and a single motor. With respect to the two–wheeled configuration, both the wheel/rotor inertial parameters and the motor torque are doubled (note that in order to keep a straight motion, it is required that the left and right motor torques are always identical). This planar model will be adopted in the following for the derivation of the Equations–of–Motion (EoM) of the longitudinal dynamics. To further simplify the analysis, it will be assumed that the robot moves on a horizontal flat surface (i.e. the gravity vector is perpendicular to the ground surface), so that $z_v = 0$ and, because of the pure rolling condition imposed to the wheels, $x_v = r\gamma$.

The Equations–of–Motion are derived by using a Lagrangian approach. This requires first to obtain the kinematics equations for each body in the planar multi–body system. For such purpose, the following notation is introduced:

- $\boldsymbol{p}_a^b = \left[x_a^b,\, z_a^b\right]^T$: position vector of point a, expressed with respect to frame $\{b\}$. If $\{b\}$ is the world frame $\{o\}$, then the superscripts in the position coordinates will be omitted.

- $\boldsymbol{r}_{a,b}^{c} = \left[x_{a,b}^{c},\, z_{a,b}^{c}\right]^{T}$: displacement vector from point a to point b, expressed with respect to frame $\{c\}$. It holds that:

$$\boldsymbol{r}_{a,b}^{c} \;=\; \boldsymbol{p}_{b}^{c} \;-\; \boldsymbol{p}_{a}^{c} \tag{11}$$

- $\boldsymbol{R}_{a}^{b}$: rotation matrix of frame $\{a\}$ with respect to frame $\{b\}$. If ϑ is the angle by which the frame $\{a\}$ is rotated with respect to frame $\{b\}$, then the rotation matrix is equal to

$$\boldsymbol{R}_{a}^{b} \;=\; \begin{bmatrix} \cos\vartheta & \sin\vartheta \\ -\sin\vartheta & \cos\vartheta \end{bmatrix} \tag{12}$$

The derivation of the kinematics equations for each body in the planar multi–body system is reported below:

- *Robot body kinematics*. Let $\boldsymbol{p}_{v}^{o} = [r\,\gamma,\, 0]^{T}$ be the position of the vehicle frame with respect to world frame, $\boldsymbol{p}_{C_b}^{b} = [0,\, l]^{T}$ with $l \triangleq z_{b}^{b}$ the position of the robot body CoM with respect to the body frame, and

$$\boldsymbol{R}_{b}^{o} = \begin{bmatrix} \cos\vartheta & \sin\vartheta \\ -\sin\vartheta & \cos\vartheta \end{bmatrix} \tag{13}$$

  the rotation matrix of body frame with respect to world frame. Then, the position vector $\boldsymbol{p}_{C_b}^{o} = [x_{C_b},\, z_{C_b}]^{T}$ of the robot body CoM C_b with respect to the world frame is equal to:

$$\boldsymbol{p}_{C_b}^{o} \;=\; \boldsymbol{p}_{v}^{o} \;+\; \boldsymbol{r}_{v,C_b}^{o} \;=\; \boldsymbol{p}_{v}^{o} \;+\; \boldsymbol{R}_{b}^{o}\,\boldsymbol{p}_{C_b}^{b} \;=\; \begin{bmatrix} r\,\gamma \;+\; l\,\sin\vartheta \\ l\,\cos\vartheta \end{bmatrix} \tag{14}$$

  The linear velocity vector $\dot{\boldsymbol{p}}_{C_b}^{o} = [\dot{x}_{C_b},\, \dot{z}_{C_b}]^{T}$ of the robot body CoM with respect to world frame is therefore equal to:

$$\dot{\boldsymbol{p}}_{C_b}^{o} \;=\; \dot{\boldsymbol{p}}_{v}^{o} \;+\; \frac{d\boldsymbol{R}_{b}^{o}}{dt}\,\boldsymbol{p}_{C_b}^{b} \;+\; \boldsymbol{R}_{b}^{o}\,\dot{\boldsymbol{p}}_{C_b}^{b} \;=\; \begin{bmatrix} r\,\dot{\gamma} \;+\; l\,\cos\vartheta\,\dot{\vartheta} \\ -\,l\,\sin\vartheta\,\dot{\vartheta} \end{bmatrix} \tag{15}$$

- *Wheel kinematics*. The position vector $\boldsymbol{p}_{C_l}^{o} = [x_{C_w},\, z_{C_w}]^{T}$ of the single–wheel CoM C_w with respect to the world frame is equal to

$$\boldsymbol{p}_{C_w}^{o} \;=\; \boldsymbol{p}_{v}^{o} \;+\; \boldsymbol{r}_{v,C_w}^{o} \;=\; \boldsymbol{p}_{v}^{o} \;=\; \begin{bmatrix} r\,\gamma \\ 0 \end{bmatrix} \tag{16}$$

  since $\boldsymbol{r}_{v,C_w}^{o} = [0,\, 0]^{T}$. The linear velocity vector $\dot{\boldsymbol{p}}_{C_w}^{o} = [\dot{x}_{C_w},\, \dot{z}_{C_w}]^{T}$ is therefore equal to

$$\dot{\boldsymbol{p}}_{C_w}^{o} \;=\; \dot{\boldsymbol{p}}_{v}^{o} \;=\; \begin{bmatrix} r\,\dot{\gamma} \\ 0 \end{bmatrix} \tag{17}$$

- *Rotor kinematics*. Let $\boldsymbol{p}_{C_{rot}}^{b} = [0,\, z_{rot}^{b}]^{T}$ be the position of the single–rotor CoM C_{rot} with respect to the body frame. Then, the position vector $\boldsymbol{p}_{C_{rot}}^{o} = [x_{C_{rot}},\, z_{C_{rot}}]^{T}$ of the same CoM with respect to world frame is equal to:

$$\boldsymbol{p}_{C_{rot}}^{o} \;=\; \boldsymbol{p}_{v}^{o} \;+\; \boldsymbol{r}_{v,C_{rot}}^{o} \;=\; \boldsymbol{p}_{v}^{o} \;+\; \boldsymbol{R}_{b}^{o}\,\boldsymbol{p}_{C_{rot}}^{b} \;=\; \begin{bmatrix} r\,\gamma \;+\; z_{rot}^{b}\,\sin\vartheta \\ z_{rot}^{b}\,\cos\vartheta \end{bmatrix} \tag{18}$$

The linear velocity vector $\dot{\boldsymbol{p}}^o_{C_{rot}} = [\dot{x}^o_{C_{rot}},\, \dot{z}^o_{C_{rot}}]^T$ of the rotor CoM with respect to world frame is therefore equal to:

$$\dot{\boldsymbol{p}}^o_{C_{rot}} \;=\; \dot{\boldsymbol{p}}^o_v \;+\; \frac{d\boldsymbol{R}^o_b}{dt}\,\boldsymbol{p}^b_{C_{rot}} \;+\; \boldsymbol{R}^o_b\,\dot{\boldsymbol{p}}^b_{C_{rot}} \;=\; \begin{bmatrix} r\,\dot{\gamma} \;+\; z^b_{rot}\,\cos\vartheta\,\dot{\vartheta} \\ -\,z^b_{rot}\,\sin\vartheta\,\dot{\vartheta} \end{bmatrix} \tag{19}$$

For the application of the Lagrangian approach, it is necessary to compute the Lagrangian function:

$$\mathcal{L} \;=\; T \;-\; U \tag{20}$$

which depends on the kinetic energy T and potential energy U of the whole multi–body system. These are in turn equal to the sum of the kinetic and potential energies of every single body composing the planar multi–body system, namely

$$T \;=\; T_b \;+\; T_w \;+\; T_{rot} \qquad \text{and} \qquad U \;=\; U_b \;+\; U_w \;+\; U_{rot} \tag{21}$$

The derivation of every single kinetic and potential energy contributions is reported below:

- *Robot body kinetic and potential energies*. The kinetic energy of the robot body is equal to:

$$\begin{aligned} T_b \;&=\; \frac{1}{2}\,m_b\,\left(\dot{\boldsymbol{p}}^o_{C_b}\right)^T\left(\dot{\boldsymbol{p}}^o_{C_b}\right) \;+\; \frac{1}{2}\,I_{b,yy}\,\dot{\vartheta}^2 \\ &=\; \frac{1}{2}\,m_b\,r^2\,\dot{\gamma}^2 \;+\; \frac{1}{2}\left(I_{b,yy} + m_b\,l^2\right)\dot{\vartheta}^2 \;+\; m_b\,r\,l\cos\vartheta\,\dot{\gamma}\,\dot{\vartheta} \end{aligned} \tag{22}$$

  where both the translational kinetic energy of the CoM and the rotational kinetic energy of the body with respect to its CoM have been taken into account. The potential energy is equal to:

$$U_b \;=\; m_b\,g\,z_{C_b} \;=\; m_b\,g\,l\cos\vartheta \tag{23}$$

- *Wheel kinetic and potential energies*. The kinetic energy of the single–wheel is equal to:

$$\begin{aligned} T_w \;&=\; \frac{1}{2}\,(2m_w)\,\left(\dot{\boldsymbol{p}}^o_{C_w}\right)^T\left(\dot{\boldsymbol{p}}^o_{C_w}\right) \;+\; \frac{1}{2}\,\left(2I_{w,yy}\right)\dot{\gamma}^2 \\ &=\; \left(I_{w,yy} + m_w r^2\right)\dot{\gamma}^2 \end{aligned} \tag{24}$$

  The potential energy is instead equal to $U_w = (2m_w)\,g\,z_{C_w} = 0$.

- *Rotor kinetic and potential energies*. Let $\vartheta_{rot} = \vartheta + N\,(\gamma - \vartheta)$ denote the angular position of the single–rotor. Then, the kinetic energy is equal to:

$$\begin{aligned} T_{rot} \;&=\; \frac{1}{2}\,(2m_{rot})\,\left(\dot{\boldsymbol{p}}^o_{C_{rot}}\right)^T\left(\dot{\boldsymbol{p}}^o_{C_{rot}}\right) \;+\; \frac{1}{2}\,(2I_{rot,yy})\,\dot{\vartheta}^2_{rot} \\ &=\; \left(N^2 I_{rot,yy} + m_{rot}\,r^2\right)\dot{\gamma}^2 \;+\; \left[(1-N)^2 I_{rot,yy} + m_{rot}(z^b_{rot})^2\right]\dot{\vartheta}^2 \;+\; \cdots \\ &\quad \cdots + 2\left[N(1-N)I_{rot,yy} \;+\; m_{rot}\,r\,z^b_{rot}\cos\vartheta\right]\dot{\gamma}\,\dot{\vartheta} \end{aligned} \tag{25}$$

  The potential energy is equal to:

$$U_{rot} \;=\; (2m_{rot})\,g\,z_{C_{rot}} \;=\; (2m_{rot})\,g\,z^b_{rot}\cos\vartheta \tag{26}$$

Gearbox viscous friction coefficient (at output shaft)	B	$25 \times 10^{-3}\,\mathrm{Nm/(rad/s)}$
Wheel viscous friction coefficient	B_w	$1.5 \times 10^{-3}\,\mathrm{Nm/(rad/s)}$

Table 2: Viscous friction nominal parameters.

The Lagrangian function is therefore equal to:

$$\begin{aligned}\mathcal{L} \;=\; & \left[I_{w,yy} \,+\, N^2\, I_{rot,yy} \,+\, \left(\frac{1}{2}\, m_b \,+\, m_w \,+\, m_{rot}\right) r^2\right] \dot{\gamma}^2 \,+\, \cdots \\ & \cdots \,+\, \left[\frac{1}{2}\, I_{b,yy} \,+\, (1-N)^2\, I_{rot,yy} \,+\, \frac{1}{2}\, m_b\, l^2 \,+\, m_{rot} (z_{rot}^b)^2\right] \dot{\vartheta}^2 \,+\, \cdots \\ & \cdots \,+\, \left[2\, N\, (1-N)\, I_{rot,yy} \,+\, \left(m_b\, l \,+\, 2\, m_{rot}\, z_{rot}^b\right) r \cos\vartheta\right] \dot{\gamma}\, \dot{\vartheta} \,-\, \cdots \\ & \cdots \,-\, \left(m_b\, l \,+\, 2\, m_{rot}\, z_{rot}^b\right) g\, \cos\vartheta\end{aligned} \tag{27}$$

The Equations-of–Motion describing the longitudinal dynamics of the balancing robot are obtained by evaluating the following *Lagrange equations*:

$$\begin{aligned}\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\gamma}} \,-\, \frac{\partial \mathcal{L}}{\partial \gamma} \;&=\; \xi_\gamma \\ \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\vartheta}} \,-\, \frac{\partial \mathcal{L}}{\partial \vartheta} \;&=\; \xi_\vartheta\end{aligned} \tag{28}$$

where ξ_γ and ξ_ϑ are the *generalized forces (torques)* associated with the generalized coordinates γ and ϑ. The expression for computing the generic generalized force ξ_j associated with the generic generalized coordinate q_j is:

$$\xi_j \;=\; \sum_{i=1}^{n} \boldsymbol{F}_i^T\, \frac{\partial \boldsymbol{p}_i}{\partial q_j} \,+\, \sum_{i=1}^{n} \boldsymbol{\tau}_i^T\, \frac{\partial \boldsymbol{\vartheta}_i}{\partial q_j} \tag{29}$$

where $\boldsymbol{F}_i$ and $\boldsymbol{\tau}_i$ are the generic force and torque acting on the system, while $\boldsymbol{p}_i$ and $\boldsymbol{\vartheta}_i$ are their point and axis of application (expressed in world frame). The forces/torques are either external (e.g. actuation forces/torques, external disturbances, etc.), or nonconservative (e.g. frictional force/torque). In the balancing robot example, the contributions to be considered are the motor torque τ generated at the output shaft, and the two dominant viscous friction torques

$$\tau_f' \;=\; B\,(\dot{\gamma} \,-\, \dot{\vartheta}) \qquad \text{and} \qquad \tau_f'' = B_w \dot{\gamma} \tag{30}$$

The former is due to the gearmotor internal friction, and is proportional to the rotation speed of the motor output shaft with respect to the stator. The latter is instead the viscous friction torque acting on the rolling wheel, and is proportional to the wheel speed. By considering that these contributions act identically on both the left and right sides, from (29) it follows that:

$$\begin{aligned}\xi_\gamma \;&=\; 2\,\tau \,-\, 2\,\tau_f' \,-\, 2\,\tau_f'' \;=\; 2\tau \,-\, 2\,(B + B_w)\,\dot{\gamma} \,+\, 2B\dot{\vartheta} \\ \xi_\vartheta \;&=\; -2\tau + 2\tau_f' \;=\; -2\tau \,+\, 2B\dot{\gamma} \,-\, 2B\dot{\vartheta}\end{aligned} \tag{31}$$

The estimated values of the viscous friction coefficients B and B_w are reported in Tab. 2.
After replacing (27) and (31) within (28), the following equations–of–motion finally result for the

longitudinal dynamics:

$$\begin{aligned}
&\left[2\,I_{w,yy} \,+\, 2\,N^2\,I_{rot,yy} \,+\, (m_b \,+\, 2\,m_w \,+\, 2\,m_{rot})\,r^2\right]\ddot{\gamma} \,+\, 2\,(B + B_w)\,\dot{\gamma} \,+\, \cdots \\
&\quad \cdots + \left[2\,N\,(1-N)\,I_{rot,yy} \,+\, \left(m_b\,l \,+\, 2\,m_{rot}\,z_{rot}^b\right) r\cos\vartheta\right]\ddot{\vartheta} \,-\, 2B\dot{\vartheta} \,-\, \cdots \\
&\quad \cdots - \left(m_b\,l \,+\, 2\,m_{rot}\,z_{rot}^b\right) r\,\sin\vartheta\,\dot{\vartheta}^2 - 2\tau \;=\; 0
\end{aligned} \tag{32}$$

$$\begin{aligned}
&\left[2\,N\,(1-N)\,I_{rot,yy} \,+\, \left(m_b\,l \,+\, 2\,m_{rot}\,z_{rot}^b\right) r\cos\vartheta\right]\ddot{\gamma} \,-\, 2B\dot{\gamma} \,+\, \cdots \\
&\quad \cdots + \left[I_{b,yy} \,+\, 2\,(1-N)^2\,I_{rot,yy} \,+\, m_b\,l^2 \,+\, 2\,m_{rot}(z_{rot}^b)^2\right]\ddot{\vartheta} \,+\, 2B\dot{\vartheta} \,-\, \cdots \\
&\quad \cdots - \left(m_b\,l \,+\, 2\,m_{rot}\,z_{rot}^b\right) g\,\sin\vartheta \,+\, 2\tau \;=\; 0
\end{aligned} \tag{33}$$

Let $\boldsymbol{q} = [\gamma,\, \vartheta]^T$ be the vector of the generalized variables. A compact matrix formulation of the dynamical model (32)–(33) is:

$$\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} \,+\, \boldsymbol{C}(\boldsymbol{q},\,\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} \,+\, \boldsymbol{F}_v\,\dot{\boldsymbol{q}} \,+\, \boldsymbol{g}(\boldsymbol{q}) \;=\; \boldsymbol{\tau} \tag{34}$$

where

$$\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix} M_{11}(\boldsymbol{q}) & M_{12}(\boldsymbol{q}) \\ M_{21}(\boldsymbol{q}) & M_{22}(\boldsymbol{q}) \end{bmatrix}, \quad \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} C_{11}(\boldsymbol{q},\dot{\boldsymbol{q}}) & C_{12}(\boldsymbol{q},\dot{\boldsymbol{q}}) \\ C_{21}(\boldsymbol{q},\dot{\boldsymbol{q}}) & C_{22}(\boldsymbol{q},\dot{\boldsymbol{q}}) \end{bmatrix}, \quad \boldsymbol{F}_v = \begin{bmatrix} F_{v,11} & F_{v,12} \\ F_{v,21} & F_{v,22} \end{bmatrix} \tag{35}$$

with

$$\begin{aligned}
M_{11}(\boldsymbol{q}) \;&=\; 2\,I_{w,yy} \,+\, 2\,N^2\,I_{rot,yy} \,+\, (m_b \,+\, 2\,m_w \,+\, 2\,m_{rot})\,r^2 \\
M_{12}(\boldsymbol{q}) \;&=\; M_{21}(\boldsymbol{q}) \;=\; 2\,N\,(1-N)\,I_{rot,yy} \,+\, \left(m_b\,l \,+\, 2\,m_{rot}\,z_{rot}^b\right) r\cos\vartheta \\
M_{22}(\boldsymbol{q}) \;&=\; I_{b,yy} \,+\, 2\,(1-N)^2\,I_{rot,yy} \,+\, m_b\,l^2 \,+\, 2\,m_{rot}(z_{rot}^b)^2
\end{aligned} \tag{36}$$

$$\begin{aligned}
C_{11}(\boldsymbol{q},\dot{\boldsymbol{q}}) \;&=\; C_{21}(\boldsymbol{q},\dot{\boldsymbol{q}}) \;=\; C_{22}(\boldsymbol{q},\dot{\boldsymbol{q}}) \;=\; 0 \\
C_{12}(\boldsymbol{q},\dot{\boldsymbol{q}}) \;&=\; -\left(m_b\,l \,+\, 2\,m_{rot}\,z_{rot}^b\right) r\,\sin\vartheta\,\dot{\vartheta}
\end{aligned} \tag{37}$$

$$\begin{aligned}
F_{v,11} \;&=\; 2(B + B_w) \\
F_{v,12} \;&=\; F_{v,21} \;=\; -2B \\
F_{v,22} \;&=\; 2B
\end{aligned} \tag{38}$$

are, respectively, the inertia matrix, the matrix of centrifugal and Coriolis–related coefficients (so that $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}}$ is the torque contribution due to the centrifugal and Coriolis accelerations), and the matrix of viscous friction coefficients $\boldsymbol{F}_v$, while

$$\boldsymbol{g}(\boldsymbol{q}) \;=\; \left[0\,,\quad -\left(m_b\,l \,+\, 2\,m_{rot}\,z_{rot}^b\right) g\sin\vartheta\right]^T \tag{39}$$

is the torque contribution due to gravity. The motor torque input is equal to $\boldsymbol{\tau} = [2\tau,\, -2\tau]^T$.
The dynamical model (32)–(33) is nonlinear; for the design of the linear state–space balancing controller, it has to be linearized about the unstable equilibrium configuration with the robot body in steady upward vertical position. Consider the unstable equilibrium point $P_0 = (\boldsymbol{q}_0,\, \dot{\boldsymbol{q}}_0,\, \ddot{\boldsymbol{q}}_0,\, \boldsymbol{\tau}_0)$ with $\boldsymbol{q}_0 = [\gamma_0,\, 0]^T$, $\dot{\boldsymbol{q}}_0 = \ddot{\boldsymbol{q}}_0 = [0,\, 0]^T$ and $\tau_0 = 0$ (note that P_0 is an equilibrium point for any choice of $\gamma_0 \in \mathbb{R}$). The linearization of (34) around P_0 is given by:

$$\boldsymbol{f}(P_0) \,+\, \frac{\partial \boldsymbol{f}\,(P_0)}{\partial \boldsymbol{q}}\,\delta\boldsymbol{q} \,+\, \frac{\partial \boldsymbol{f}\,(P_0)}{\partial \dot{\boldsymbol{q}}}\,\delta\dot{\boldsymbol{q}} \,+\, \frac{\partial \boldsymbol{f}\,(P_0)}{\partial \ddot{\boldsymbol{q}}}\,\delta\ddot{\boldsymbol{q}} \,+\, \frac{\partial \boldsymbol{f}\,(P_0)}{\partial \tau}\delta\tau \;=\; \boldsymbol{0} \tag{40}$$

where

$$\boldsymbol{f}\,(\boldsymbol{q},\, \dot{\boldsymbol{q}},\, \ddot{\boldsymbol{q}},\, \tau) \;=\; \boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} \,+\, \boldsymbol{C}(\boldsymbol{q},\, \dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} \,+\, \boldsymbol{F}_v\,\dot{\boldsymbol{q}} \,+\, \boldsymbol{g}(\boldsymbol{q}) \,-\, \boldsymbol{\tau} \tag{41}$$

and $\delta\boldsymbol{q}$, $\delta\dot{\boldsymbol{q}}$, $\delta\ddot{\boldsymbol{q}}$ and $\delta\tau$ denote small deviations around the equilibrium values specified in P_0 (e.g. $\delta\boldsymbol{q} = \boldsymbol{q} - \boldsymbol{q}_0$). Note that $\boldsymbol{f}\,(P_0) = \boldsymbol{0}$ in the Taylor's expansion of $\boldsymbol{f}$, since P_0 is an equilibrium point. The evaluation of (40) yields the following linearized model:

$$\begin{aligned}
&\left[2\,I_{w,yy} \,+\, 2\,N^2\,I_{rot,yy} \,+\, (m_b \,+\, 2\,m_w \,+\, 2\,m_{rot})\,r^2\right]\ddot{\gamma} \,+\, 2\,(B + B_w)\,\dot{\gamma} \,+\, \cdots \\
&\cdots \,+\, \left[2\,N\,(1-N)\,I_{rot,yy} \,+\, \left(m_b\,l \,+\, 2\,m_{rot}\,z^b_{rot}\right)r\right]\ddot{\vartheta} \,-\, 2B\dot{\vartheta} \,-\, 2\tau \;=\; 0
\end{aligned} \tag{42}$$

$$\begin{aligned}
&\left[2\,N\,(1-N)\,I_{rot,yy} \,+\, \left(m_b\,l \,+\, 2\,m_{rot}\,z^b_{rot}\right)r\right]\ddot{\gamma} \,-\, 2B\dot{\gamma} \,+\, \cdots \\
&\cdots \,+\, \left[I_{b,yy} \,+\, 2\,(1-N)^2\,I_{rot,yy} \,+\, m_b\,l^2 \,+\, 2\,m_{rot}(z^b_{rot})^2\right]\ddot{\vartheta} \,+\, 2B\dot{\vartheta} \,-\, \cdots \\
&\qquad \cdots \,-\, \left(m_b\,l \,+\, 2\,m_{rot}\,z^b_{rot}\right)g\,\vartheta \,+\, 2\tau \;=\; 0
\end{aligned} \tag{43}$$

where, with some abuse of notation, the variables γ, ϑ and τ have been reused (in place of $\delta\gamma$, $\delta\vartheta$ and $\delta\tau$) to denote the small deviations around the equilibrium values. A compact matrix formulation of (42)–(43) is:

$$\boldsymbol{M}\,\ddot{\boldsymbol{q}} \,+\, \boldsymbol{F}_v\,\dot{\boldsymbol{q}} \,+\, \boldsymbol{G}\,\boldsymbol{q} \;=\; \boldsymbol{\tau} \tag{44}$$

where

$$\boldsymbol{M} \;=\; \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \qquad \boldsymbol{G} \;=\; \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} \tag{45}$$

with

$$\begin{aligned}
M_{11} \;&=\; 2\,I_{w,yy} \,+\, 2\,N^2\,I_{rot,yy} \,+\, (m_b \,+\, 2\,m_w \,+\, 2\,m_{rot})\,r^2 \\
M_{12} \;&=\; M_{21} \;=\; 2\,N\,(1-N)\,I_{rot,yy} \,+\, \left(m_b\,l \,+\, 2\,m_{rot}\,z^b_{rot}\right)r \\
M_{22} \;&=\; I_{b,yy} \,+\, 2\,(1-N)^2\,I_{rot,yy} \,+\, m_b\,l^2 \,+\, 2\,m_{rot}(z^b_{rot})^2
\end{aligned} \tag{46}$$

$$\begin{aligned}
G_{11} \;&=\; G_{12} \;=\; G_{21} \;=\; 0 \\
G_{22} \;&=\; -\left(m_b\,l \,+\, 2\,m_{rot}\,z^b_{rot}\right)g
\end{aligned} \tag{47}$$

are the linearized versions around the equilibrium point P_0 of the inertia matrix $\boldsymbol{M}(\boldsymbol{q})$ and the gravity

torque contribution $\boldsymbol{g}(\boldsymbol{q})$.

The linear model (44) can be alternatively rewritten in state–space form as follows. Define the vector of state variables as $\boldsymbol{x} = [\boldsymbol{q},\, \dot{\boldsymbol{q}}]^T = [\gamma,\, \vartheta,\, \dot{\gamma},\, \dot{\vartheta}]^T$; then, the resulting state–space model is:

$$\dot{\boldsymbol{x}} \;=\; \boldsymbol{A}\,\boldsymbol{x} \;+\; \boldsymbol{B}\,u \tag{48}$$

where

$$\boldsymbol{A} \;=\; \begin{bmatrix} \boldsymbol{0}_{2\times 2} & \boldsymbol{I}_{2\times 2} \\ -\boldsymbol{M}^{-1}\boldsymbol{G} & -\boldsymbol{M}^{-1}\boldsymbol{F}_v \end{bmatrix}, \qquad \boldsymbol{B} \;=\; \begin{bmatrix} \boldsymbol{0}_{2\times 2} \\ \boldsymbol{M}^{-1} \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{49}$$

and $u = 2\,\tau$. Note that (48) has a scalar input; for the computation of the input matrix $\boldsymbol{B}$ in (49), it has been used the fact that $\boldsymbol{\tau} = [1,\, -1]^T(2\tau)$.

2.2 Model of the actuation system

The electromechanical torque generated by each DC gearmotor at the output shaft is equal to:

$$\tau \;=\; N\,k_t\,i_a \tag{50}$$

where i_a is the armature current, and k_t is the torque constant. The current i_a satisfies the electrical equation:

$$L_a\,\frac{di_a}{dt} \;+\; R_a\,i_a \;+\; k_e\,\frac{d\Delta\vartheta_{rot}}{dt} \;=\; u_a \tag{51}$$

where R_a and L_a are the resistance and inductance of the armature circuit, k_e is the electric (back electromotive force – BEMF) constant, $\Delta\vartheta_{rot} = \vartheta_{rot} - \vartheta$ is the angular displacement of the rotor with respect to the stator, and u_a is the supply voltage to the armature circuit. Note that the BEMF is proportional to how fast the rotor spins with respect to the stator. Suppose that the electrical time constant L_a/R_a is negligible (compared to the smallest mechanical time constant of the system): then, the differential equation (51) reduces to the following algebraic relation

$$i_a \;=\; \frac{1}{R_a}\Big[u_a \;-\; k_e\,N\,(\dot{\gamma} - \dot{\vartheta})\Big] \tag{52}$$

where it has been used the fact that $d(\Delta\vartheta_{rot})/dt = N(\dot{\gamma} - \dot{\vartheta})$. Correspondingly, the torque expression (50) becomes:

$$\tau \;=\; \frac{Nk_t}{R_a}\,u_a \;-\; \frac{N^2 k_t k_e}{R_a}\,\Big(\dot{\gamma} - \dot{\vartheta}\Big) \tag{53}$$

Once the expression (53) is replaced within (34), the following nonlinear dynamical model results:

$$\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} \;+\; \boldsymbol{C}(\boldsymbol{q},\,\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} \;+\; \boldsymbol{F}'_v\,\dot{\boldsymbol{q}} \;+\; \boldsymbol{g}(\boldsymbol{q}) \;=\; \boldsymbol{\tau}' \tag{54}$$

where

$$\boldsymbol{F}'_v \;=\; \boldsymbol{F}_v \;+\; \frac{2\,N^2\,k_t\,k_e}{R_a}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \qquad \boldsymbol{\tau}' \;=\; \frac{2\,N\,k_t}{R_a}\begin{bmatrix} 1 \\ -1 \end{bmatrix} u_a \tag{55}$$

Similarly, by replacing (53) within (44), the following linear dynamical model results:

$$\boldsymbol{M}\,\ddot{\boldsymbol{q}} \;+\; \boldsymbol{F}'_v\,\dot{\boldsymbol{q}} \;+\; \boldsymbol{G}\,\boldsymbol{q} \;=\; \boldsymbol{\tau}' \tag{56}$$

Armature resistance	R_a	$2.4\,\Omega$
Armature inductance	L_a	n.a. (neglected)
Electric (BEMF) constant	k_e	$10.3 \times 10^{-3}\,\mathrm{V\,s/rad}$
Torque constant	k_m	$5.2 \times 10^{-3}\,\mathrm{Nm/A}$
Gearbox ratio	N	30

Table 3: DC gearmotor nominal parameters.

where $\boldsymbol{F}'_v$ and $\boldsymbol{\tau}'$ are defined as in (55). The corresponding state–space model is:

$$\dot{\boldsymbol{x}} \;=\; \boldsymbol{A}\,\boldsymbol{x} \;+\; \boldsymbol{B}\,u \tag{57}$$

with

$$\boldsymbol{A} \;=\; \begin{bmatrix} \boldsymbol{0}_{2\times 2} & \boldsymbol{I}_{2\times 2} \\ -\boldsymbol{M}^{-1}\boldsymbol{G} & -\boldsymbol{M}^{-1}\boldsymbol{F}'_v \end{bmatrix}, \qquad \boldsymbol{B} \;=\; \frac{2\,N\,k_t}{R_a} \begin{bmatrix} \boldsymbol{0}_{2\times 2} \\ \boldsymbol{M}^{-1} \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{58}$$

and $u = u_a$. The nominal values of the DC gearmotor parameters are reported in Tab. 3.

2.3 Model of the inertial measurement system

The linear acceleration and angular velocity measurements provided by the inertial measurement sensor (also referred as Motion Processing Unit – MPU) are referred to a reference frame $\{mpu\}$ whose origin is located on the sensor geometrical center C_{mpu}, and whose axes are aligned with those of the body frame $\{b\}$. The coordinates of the sensor center with respect to the body frame are reported in Tab. 4. The acceleration measurement is provided by the onboard accelerometer, and is equal to the sum of the actual linear acceleration experienced by the robot body at the location of the sensor center, and the gravity acceleration vector. Instead, the angular velocity measurement is provided by the gyroscope, and it coincides with the actual angular velocity of the robot body.
To determine the expression of the accelerometer output as a function of the robot generalized coordinates (and their derivatives), it is first necessary to determine the position vector of the sensor center with respect to the world frame, and derive it twice to get the acceleration. Then, after adding the gravity contribution, the result has to be reprojected on the sensor frame to get the actual sensor output. Let $\boldsymbol{p}^b_{C_{mpu}} = [0,\, z^b_{mpu}]^T$ be the position vector of the sensor center with respect to the body frame. Then, the position vector $\boldsymbol{p}^o_{C_{mpu}} = [x_{C_{mpu}},\, z_{C_{mpu}}]^T$ of the sensor center with respect to the world frame is equal to:

$$\boldsymbol{p}^o_{C_{mpu}} \;=\; \boldsymbol{p}^o_v \;+\; \boldsymbol{R}^o_b\,\boldsymbol{p}^b_{C_{mpu}} \;=\; \begin{bmatrix} r\,\gamma \;+\; z^b_{mpu}\,\sin\vartheta \\ z^b_{mpu}\,\cos\vartheta \end{bmatrix} \tag{59}$$

The linear velocity and acceleration vectors are obtained by differentiation of the position vector, and are equal to:

$$\dot{\boldsymbol{p}}^o_{C_{mpu}} \;=\; \dot{\boldsymbol{p}}^o_v \;+\; \frac{d\boldsymbol{R}^o_b}{dt}\,\boldsymbol{p}^b_{C_{mpu}} \;+\; \boldsymbol{R}^o_b\,\dot{\boldsymbol{p}}^b_{C_{mpu}} \;=\; \begin{bmatrix} r\,\dot{\gamma} \;+\; z^b_{mpu}\,\cos\vartheta\,\dot{\vartheta} \\ -z^b_{mpu}\,\sin\vartheta\,\dot{\vartheta} \end{bmatrix} \tag{60}$$

Sensor center wrt body frame $\{b\}$	$x^b_{mpu},\ y^b_{mpu},\ z^b_{mpu}$	$0,\ 0,\ 13.5$	$[\mathrm{mm}]$

Table 4: Motion processing unit (MPU) geometrical parameters.

$$\ddot{\boldsymbol{p}}^o_{C_{mpu}} = \ddot{\boldsymbol{p}}^o_v + \frac{d^2\boldsymbol{R}^o_b}{dt^2}\,\boldsymbol{p}^b_{C_{mpu}} + 2\frac{d\,\boldsymbol{R}^o_b}{dt}\,\dot{\boldsymbol{p}}_{C_{mpu}} + \boldsymbol{R}^o_b\,\ddot{\boldsymbol{p}}^b_{C_{mpu}} = \begin{bmatrix} r\,\ddot{\gamma} + z^b_{mpu}\,(-\sin\vartheta\,\dot{\vartheta}^2 + \cos\vartheta\,\ddot{\vartheta}) \\ -z^b_{mpu}\,(\cos\vartheta\,\dot{\vartheta}^2 + \sin\vartheta\,\ddot{\vartheta}) \end{bmatrix} \tag{61}$$

The accelerometer output $\boldsymbol{y}_a = [x^{mpu}_a,\ z^{mpu}_a]^T$ is therefore equal to:

$$\boldsymbol{y}_a = \boldsymbol{R}^b_o\left(\ddot{\boldsymbol{p}}^o_{C_{mpu}} + \boldsymbol{g}^o\right) \tag{62}$$

where $\boldsymbol{g}^o = [0,\ -g]^T$ is the gravity acceleration vector with respect to the world frame, and

$$\boldsymbol{R}^b_o = \begin{bmatrix} \cos\vartheta & -\sin\vartheta \\ \sin\vartheta & \cos\vartheta \end{bmatrix} \tag{63}$$

is the rotation matrix of the world frame $\{o\}$ with respect to the body frame $\{b\}$ (note that $\boldsymbol{R}^b_o = (\boldsymbol{R}^o_b)^{-1}$, and $(\boldsymbol{R}^o_b)^{-1} = (\boldsymbol{R}^o_b)^T$ because $\boldsymbol{R}^o_b$ is a rotation matrix). It finally holds that:

$$\boldsymbol{y}_a = \begin{bmatrix} r\,\ddot{\gamma}\cos\vartheta + z^b_{mpu}\,\ddot{\vartheta} + g\sin\vartheta \\ r\,\ddot{\gamma}\sin\vartheta - z^b_{mpu}\,\dot{\vartheta}^2 - g\cos\vartheta \end{bmatrix} \tag{64}$$

The gyroscope output is simply the rate of change of the robot body tilt angle, namely:

$$y_g = \dot{\vartheta} \tag{65}$$

3 Tilt estimation

The robot tilt angle ϑ can be estimated by using the data provided by the MPU. For such purpose, consider first the expression of the accelerometer output (64). Assume that the robot body motion is slow, so that the linear acceleration $r\ddot{\gamma}$, the tangential acceleration $z^b_{mpu}\ddot{\vartheta}$ and the centripetal acceleration $z^b_{mpu}\dot{\vartheta}^2$ can all be regarded as negligible quantities. Under this hypothesis, the measurement provided by the accelerometer is:

$$\boldsymbol{y}_a = \begin{bmatrix} x^{mpu}_a \\ z^{mpu}_a \end{bmatrix} \approx \begin{bmatrix} g\,\sin\vartheta \\ -g\cos\vartheta \end{bmatrix} \tag{66}$$

which consists of the projection of the gravity acceleration vector onto the two sensor axes. Therefore, the robot tilt angle can be estimated as follows:

$$\hat{\vartheta}_a = \mathrm{atan2}\,(x^{mpu}_a,\ -z^{mpu}_a) \tag{67}$$

where $\mathrm{atan2}$ is the four–quadrant arctangent function, namely:

$$\mathrm{atan2}(y,\, x) \;=\; \begin{cases} \mathrm{atan}\,(y/x) & \text{if} \quad x > 0 \\ \mathrm{atan}\,(y/x) \,+\, \pi & \text{if} \quad x < 0 \;\text{ and }\; y \geqslant 0 \\ \mathrm{atan}\,(y/x) \,-\, \pi & \text{if} \quad x < 0 \;\text{ and }\; y < 0 \\ +\pi/2 & \text{if} \quad x = 0 \;\text{ and }\; y > 0 \\ -\pi/2 & \text{if} \quad x = 0 \;\text{ and }\; y < 0 \\ \text{undefined} & \text{if} \quad x = 0 \;\text{ and }\; y = 0 \end{cases} \tag{68}$$

There are two drawbacks of using the estimate (67). Firstly, the estimate is sensitive to the robot body accelerations, and could be unreliable whenever the robot motion is not slow enough. Secondly, the estimate is corrupted by the accelerometer output noise, which is rather large, especially at high frequency. To overcome these issues, a different estimation method can be potentially exploited, which simply consists of integrating the gyroscope output (65) to obtain the desired tilt angle estimate, namely:

$$\hat{\vartheta}_g \;=\; \hat{\vartheta}_g(0) \,+\, \int_0^t y_g(\tau)\, d\tau \tag{69}$$

This estimate is certainly not affected by the robot body accelerations, and is also less noisy, since the gyroscope output noise is filtered by the operation of integration. Unfortunately, to be effective, the estimation procedure based on (69) requires to be properly initialized with the actual value of the tilt angle, i.e. the initial condition should be chosen as $\hat{\vartheta}_g(0) = \vartheta(0)$. Moreover, the gyroscope output is typically affected by both a bias (constant) and a drift (linear ramp) measurement error, which cause the integral in (69) to grow unbounded over time.

In practice, by noting that the accelerometer–based estimate $\hat{\vartheta}_a$ is more reliable at low frequency, while the gyroscope–based estimate $\hat{\vartheta}_g$ is more reliable at high frequency, one can resort to a *complementary filtering* approach for combining ("fusing") them together, thus yielding a final estimate $\hat{\vartheta}$ that is sufficiently accurate at every frequency. The approach can be briefly described as follows. Let $H(s)$ be a low–pass filter with cut–off frequency ω_c and unit DC gain, and $H'(s) = 1 - H(s)$ its complementary filter, namely the high–pass filter such that $H(s) + H'(s) = 1$. The low–pass filter can be used to remove most of the high–frequency noise affecting the accelerometer–based estimate $\hat{\vartheta}_a$, while the high–pass filter can be used to remove the low–frequency drift affecting the gyroscope–based estimate $\hat{\vartheta}_g$. In this sense, the filtered estimates:

$$\hat{\vartheta}_{a,f} \;=\; H(s)\, \hat{\vartheta}_a \,, \qquad \hat{\vartheta}_{g,f} \;=\; [1 - H(s)]\, \hat{\vartheta}_g \tag{70}$$

can be regarded as two reliable estimates of the tilt angle ϑ, which are however valid on two disjoint and complementary frequency ranges, namely the range $[0,\, \omega_c)$ for the former, and $[\omega_c,\, +\infty)$ for the latter. Consequently, their sum

$$\hat{\vartheta} \;=\; \hat{\vartheta}_{a,f} \,+\, \hat{\vartheta}_{g,f} \;=\; H(s)\, \hat{\vartheta}_a \,+\, [1 - H(s)]\, \hat{\vartheta}_g \tag{71}$$

can be regarded as a reliable estimate of the tilt angle ϑ over the whole frequency range $[0,\, +\infty)$. The choice of the order and cut–off frequency of the pair of complementary filters is necessarily obtained as a trade–off between two conflicting requirements, namely the attenuation of the accelerometer output noise at high frequency, and the rejection of the low frequency bias/drift due to the gyroscope.

In practice, such choice is performed by trial and error. The typical choices are:

- first–order complementary filters pair

$$H(s) \,=\, \frac{1}{T_c\, s + 1}\,, \qquad 1 - H(s) \,=\, \frac{T_c\, s}{T_c\, s + 1} \tag{72}$$

This is the simplest choice, and is mostly motivated by the desired to reduce the implementation complexity of the complementary filters pair. However, note that the high–pass filter $1 - H(s)$ is unable to effectively reject a bias or drift error that possibly affects the gyroscope output. In fact, suppose that the gyroscope output is affected by the error

$$\tilde{y}_g \,=\, d_g\, t \,+\, b_g \tag{73}$$

where $d_g\, t$ and b_g are, respectively, the drift and bias error components. Once integrated in (69), it gives rise to the position estimation error

$$\tilde{\vartheta}_g \,=\, \frac{1}{2}\, d_g\, t^2 \,+\, b_g\, t \tag{74}$$

The Laplace transform is

$$\tilde{\Theta}_g(s) \,=\, \frac{d_g}{s^3} \,+\, \frac{b_g}{s^2} \tag{75}$$

At the output of the high–pass filter it holds that

$$\begin{aligned} \tilde{\Theta}_{g,f}(s) \,&=\, [1 \,-\, H(s)]\, \tilde{\Theta}_g(s) \,=\, \frac{T_c\, d_g}{(T_c\, s + 1)\, s^2} \,+\, \frac{T_c\, b_g}{(T_c\, s + 1)\, s} \,=\, \cdots \\ \cdots \,&=\, -\, \frac{T_c^2\, d_g}{s} \,+\, \frac{T_c\, d_g}{s^2} \,+\, \frac{T_c^3 d_g}{T_c\, s + 1} \,+\, \frac{T_c\, b_g}{s} \,-\, \frac{T_c^2\, b_g}{T_c\, s + 1} \end{aligned} \tag{76}$$

where the last identity has been obtained by partial fraction expansion. For large values of the time t, it is immediate to verify that:

$$\tilde{\theta}_{g,f}(t) \,\approx\, (T_c\, d_g)\, t \,+\, \left(T_c\, b_g - T_c^2\, d_g\right) \tag{77}$$

namely the filtered position estimation error contains both a drift and a bias component. Therefore, an implementation based on (72) can be effectively used only when the gyroscope bias and drift are both negligible.

- second–order complementary filters pair

$$H(s) \,=\, \frac{2\, T_c\, s + 1}{(T_c\, s + 1)^2}\,, \qquad 1 - H(s) \,=\, \frac{T_c^2\, s^2}{(T_c\, s + 1)^2} \tag{78}$$

In this case, the filtered version of the position estimation error (74)–(75) is

$$\begin{aligned} \tilde{\Theta}_{g,f}(s) \,&=\, [1 \,-\, H(s)]\, \tilde{\Theta}_g(s) \,=\, \frac{T_c^2\, d_g}{(T_c\, s + 1)^2\, s} \,+\, \frac{T_c^2\, b_g}{(T_c\, s + 1)^2} \,=\, \cdots \\ \cdots \,&=\, \frac{T_c^2\, d_g}{s} \,-\, \frac{T_c^3 d_g}{T_c\, s + 1} \,-\, \frac{T_c^3 d_g}{(T_c\, s + 1)^2} \,+\, \frac{T_c^2\, b_g}{(T_c\, s + 1)^2} \end{aligned} \tag{79}$$

where the last identity has been obtained by partial fraction expansion. For large values of the

time t, it is immediate to verify that the filtered position estimation error is a constant:

$$\tilde{\theta}_{g,f}(t) \approx T_c^2 d_g \tag{80}$$

In particular, the error is insensitive to the gyroscope bias. Therefore, the implementation based on (72) is suitable for the cases where the gyroscope drift is negligible.

- third–order complementary filters pair

$$H(s) = \frac{3\,T_c^2\,s^2 + 3\,T_c\,s + 1}{(T_c\,s+1)^3}\,, \qquad 1 - H(s) = \frac{T_c^3\,s^3}{(T_c\,s+1)^3} \tag{81}$$

In this case, the filtered version of the position estimation error (74)–(75) is

$$\tilde{\Theta}_{g,f}(s) = [1 - H(s)]\,\tilde{\Theta}_g(s) = \frac{T_c^3\,d_g}{(T_c\,s+1)^3} + \frac{T_c^3\,b_g\,s}{(T_c\,s+1)^3} \tag{82}$$

For large values of the time t, it is immediate to verify that the filtered position estimation error is equal to zero, i.e. $\tilde{\theta}_{g,f}(t) \approx 0$. In practice, with a third–order complementary filters pair, the position estimation becomes insensitive to both the gyroscope bias and drift.

It is worth to point out here that complementary filtering is not the only method available for obtaining a reasonably accurate estimate of the robot tilt angle from the measurements provided by the MPU (accelerometer & gyroscope combo). An alternative approach, which is often used in practice, consists of resorting to a state observer, such as a Kalman filter[1]. Although appealing, this approach is however not considered in these notes.

4 State–space balance–and–position control

The balance–and–position control of the balancing robot is performed by resorting to conventional state–space methods. In particular:

- both the balance–and–position controllers are designed in the discrete–time domain, after discretizing the plant dynamics (57)–(58) with the exact discretization method (*direct digital design* – see handout of laboratory activity 2).
- the nominal or robust tracking of constant longitudinal position set–points is achieved with the state–space control schemes introduced in the handout of the laboratory activity 1 (obviously adapted to the discrete–time case).
- the state feedback matrix (state feedback controller gain) of the balance/position state–space controller is designed with the linear quadratic (LQ) optimal design techniques introduced in the handout of laboratory activity 3 (obviously adapted to the discrete–time case).

[1] A nice tutorial paper on complementary and Kalman filtering techniques is: W. T. Higgins, "A Comparison of Complementary and Kalman Filtering", in IEEE Transactions on Aerospace and Electronic Systems, vol. AES-11, no. 3, pp. 321-325, May 1975.

5 Laboratory assignments: numerical simulations

5.1 Simulink model of the balancing robot (longitudinal dynamics only)

1. Implement a Simulink model of the nonlinear electromechanical dynamics of the balancing robot derived in Sec. 2.1 and 2.2 – see (54)–(55). A possible Simulink implementation is shown in Fig. 5. It basically consists of rewriting (54) as follows:

$$\ddot{\boldsymbol{q}} \;=\; \boldsymbol{M}^{-1}(\boldsymbol{q}) \left[-\boldsymbol{C}(\boldsymbol{q},\, \dot{\boldsymbol{q}})\, \dot{\boldsymbol{q}} \;-\; \boldsymbol{F}_v'\, \dot{\boldsymbol{q}} \;-\; \boldsymbol{g}(\boldsymbol{q}) \;+\; \boldsymbol{\tau}' \right] \tag{83}$$

   and then using the "chain of integrators" approach to derive an equivalent block diagram representation of the (nonlinear) differential equation. The state–dependent matrices $\boldsymbol{M}(\boldsymbol{q})$, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $\boldsymbol{g}(\boldsymbol{q})$ can be implemented as shown in Fig. 5b–5d. In particular:

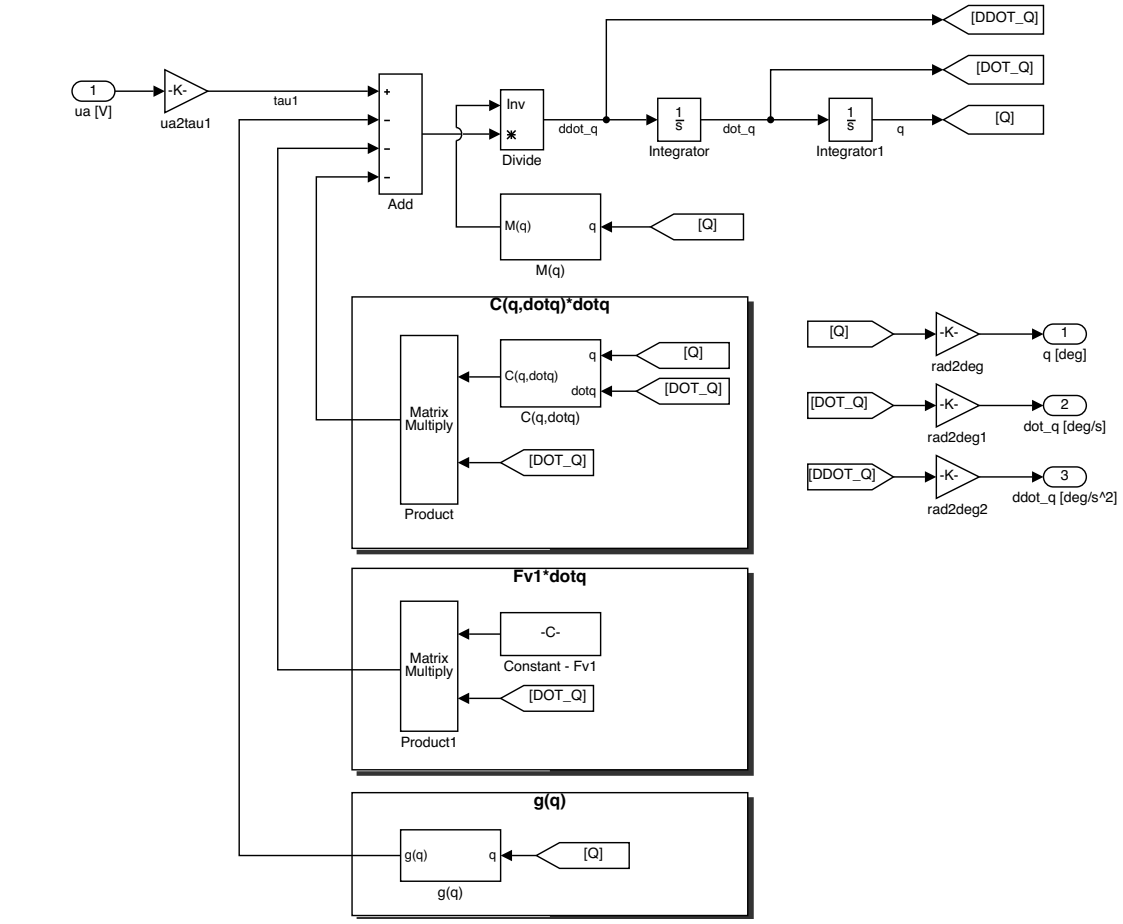
   - use the *User–Defined Functions* → *Fcn* block to implement the state–dependent elements of the aforementioned matrices. For the state–independent (i.e. constant) elements, use the *Sources* → *Constant* block.
   - use the *Signal Routing* → *Mux* block to combine the single matrix elements into matrix columns.
   - use the *Math Operations* → *Matrix Concatenate* block to form a single matrix by concatenation of its columns.
   - use the *Math Operations* → *Product* block to multiply a matrix by a column vector. In the block options, select *Matrix* among the *Multiplication* options: this instructs Simulink to compute the "row–by–column" product, instead of the conventional "element–by–element" product.

   The torque input $\boldsymbol{\tau}'$ can be obtained by multiplying the scalar input u_a by the column vector gain $(2Nk_t/R_a)\,[1,\, -1]^T$.

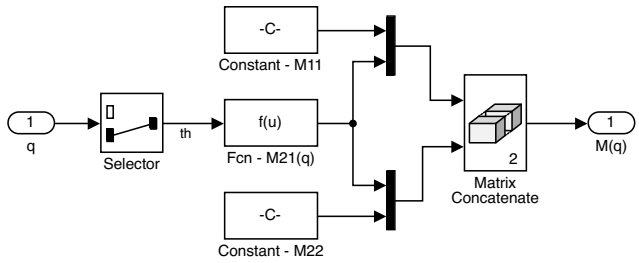
   An alternative, more elegant Simulink implementation of the nonlinear dynamics (54)–(55) consists of using MATLAB *S-Functions*. A possible implementation based on S–Function is reported in Appendix 7.1.

2. Implement a Simulink model of the Motion Processing Unit (MPU), according to the mathematical model derived in Sec. 2.3 – see (64)–(65). A possible Simulink implementation is shown in Fig. 6. In particular:

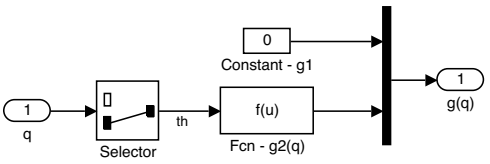
   - use the *User–Defined Functions* → *Fcn* block to compute the x and z axes components of the accelerometer output vector (64).
   - use a *Discrete* → *Zero–Order Hold* block to sample the accelerometer and gyroscope outputs with a sampling time equal to `Ts` (see line 4 in Listing 1), which corresponds to the controller sampling time $T = 0.01\,\mathrm{s}$.
   - the accelerometer outputs are in $[\mathrm{g}]$ units ($1\,\mathrm{g} = 9.81\,\mathrm{m/s^2}$). Therefore, use a `ms2g` gain to convert the accelerometer outputs from $[\mathrm{m/s^2}]$ to $[\mathrm{g}]$ units. On the other hand, the gyroscope output is in $[\mathrm{deg/s}]$ units, so that if the gyroscope input is expressed in such units, no extra units conversion is required.



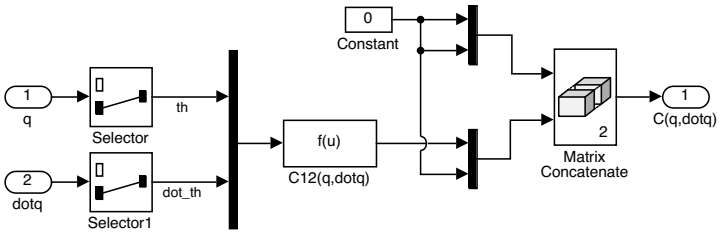
(a) Electromechanical dynamics



(b) $\boldsymbol{M}(\boldsymbol{q})$ matrix



(c) $\boldsymbol{g}(\boldsymbol{q})$ vector



(d) $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ matrix

Figure 5: Simulink model implementation details: electromechanical dynamics (implemented by using the "chain of integrators" approach).

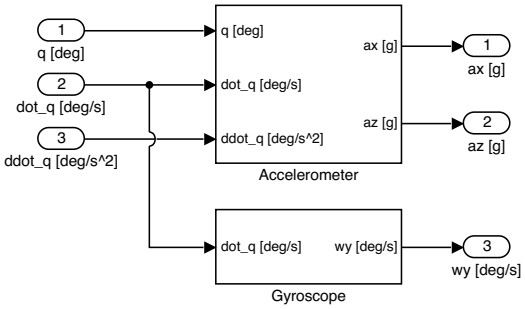
- use a *Sources* → *Random Number* block to model the normally (Gaussian) distributed noise affecting the accelerometer and gyroscope outputs. Use `sens.mpu.acc.noisevar` and `sens.mpu.gyro.noisevar` (see lines 210 and 223 in Listing 1) as the noise variances for, respectively, the accelerometer and gyroscope output noises, and `Ts` as the sampling time.
- use a *Discontinuities* → *Quantizer* block to model the finite resolution of the accelerometer and gyroscope outputs. Use `sens.mpu.acc.LSB2g` and `sens.mpu.gyro.LSB2degs` (see line 207 and 219 of Listing 1) as the quantization steps for, respectively, the accelerometer and gyroscope output quantizations.

3. Implement a Simulink model of the incremental encoder used to measure the motor shaft position. A possible Simulink implementation is shown in Fig. 7. In particular:
   - the incremental encoder measures the angular displacement $\Delta\vartheta_{rot} = \vartheta_{rot} - \vartheta$ of the rotor with respect to the stator (rigidly connected to the robot chassis). It holds that $\Delta\vartheta_{rot} = N(\gamma - \vartheta)$, where $\gamma - \vartheta$ is the angular displacement of the wheel with respect to robot body, and N is the gearbox ratio.
   - use a *Discrete* → *Zero–Order Hold* block to sample the encoder output with a sampling time equal to `Ts`.
   - use a *Discontinuities* → *Quantizer* block to model the finite resolution of the encoder. Use `sens.enc.pulse2deg` (see line 189 in Listing 1) as the quantization step (provided that the rotor angular displacement $\Delta\vartheta_{rot}$ is computed in $[\mathrm{deg}]$ units).
   - the encoder output is in "$[\mathrm{pulses}]$" units. Therefore, use a `sens.enc.deg2pulse` gain to convert the encoder output from $[\mathrm{deg}]$ to $[\mathrm{pulses}]$ units.

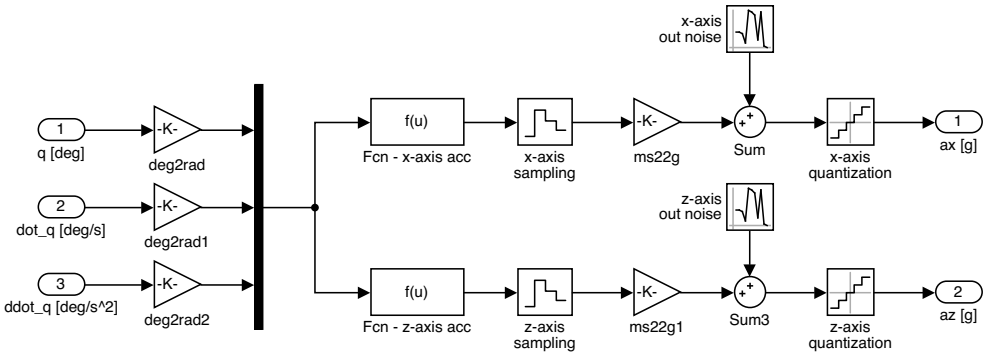
4. Implement a Simulink model of the motor voltage driver. A possible Simulink implementation is shown in Fig. 8. In particular:
   - the PWM command to the voltage driver consists of a byte, plus a sign flag. Hence, the voltage driver input can be considered as an integer number in the range $[-255, 255]$. The maximum value corresponds to apply the maximum voltage to the motor armature, which is equal to the battery nominal voltage, i.e. $11.1\,\mathrm{V}$. Use a *Discontinuities* → *Saturation* block to limit the driver input to the the voltage range specified above. Use $\pm$`drv.dutymax` (see line 114 in Listing 1) as the saturation levels.
   - use a *Math Operations* → *Rounding Function* to convert the driver input into an integer number. Select *fix* as the rounding method.
   - the driver output is the voltage applied to the motor armature. Therefore, use a `drv.duty2V` gain to convert the driver voltage command into a voltage signal in $[\mathrm{V}]$ units. Since the driver output voltage can never exceed the battery nominal voltage, consider to insert a saturation block to limit the driver output. Use $\pm$`drv.Vbus` (see line 110 in Listing 1) as the saturation levels.
   - use a *Discrete* → *Zero–Order Hold* block to hold the voltage command within each sampling period (equal to `Ts`).

5. Combine all the models prepared in points 1–4, to form a Simulink model of the whole balancing

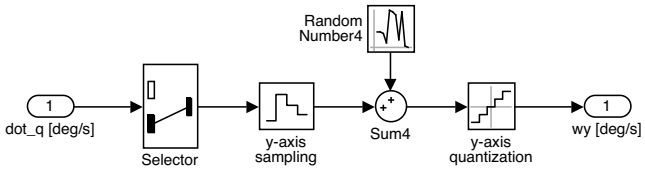
robot hardware equipment (i.e. electromechanical system + motor driver + sensors). A possible Simulink implementation is shown in Fig. 9.



(a) Motion Processing Unit (MPU)



(b) Accelerometer



(c) Gyroscope

Figure 6: Simulink model implementation details: MPU.

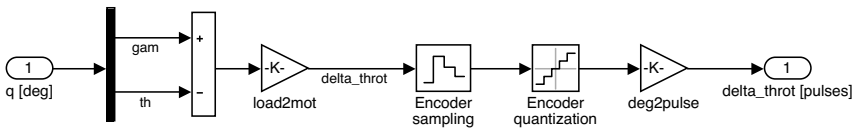


Figure 7: Simulink model implementation details: encoder.

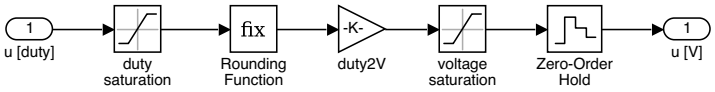


Figure 8: Simulink model implementation details: voltage driver.

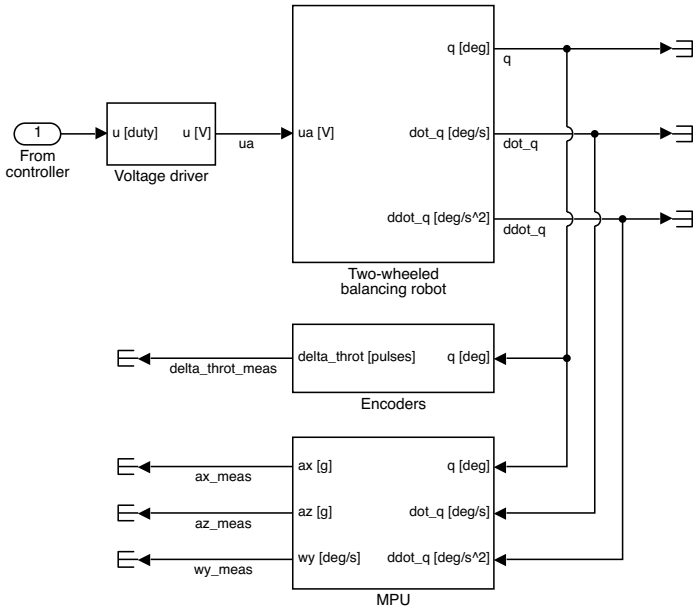


Figure 9: Simulink model implementation details: overall balancing robot hardware equipment.

Listing 1: `balrob_params.m`

```
%% General parameters and conversion gains

%   controller sampling time
Ts = 1e-2;

%   gravity acc [m/s^2]
g = 9.81;

%   conversion gains
rpm2rads = 2*pi/60;         %   [rpm]   -> [rad/s]
rads2rpm = 60/2/pi;         %   [rad/s] -> [rpm]
rpm2degs = 360/60;          %   [rpm]   -> [deg/s]
degs2rpm = 60/360;          %   [deg/s] -> [rpm]
deg2rad = pi/180;           %   [deg]   -> [rad]
rad2deg = 180/pi;           %   [rad]   -> [deg]
g2ms2 = g;                  %   [acc_g] -> [m/s^2]
ms22g = 1/g;                %   [m/s^2] -> [acc_g]
ozin2Nm = 0.706e-2;         %   [oz*inch] -> [N*m]

%   robot initial condition
x0 =[ ...
    0, ...                  %   gam(0)
    5*deg2rad, ...          %   th(0)
    0, ...                  %   dot_gam(0)
    0];                     %   dot_th(0)

%% DC motor data

%   motor id: brushed DC gearmotor Pololu 30:1 37Dx68L mm

%   electromechanical params
mot.UN   = 12;                                  %   nominal voltage
mot.taus = 110/30 * ozin2Nm;                    %   stall torque @ nom voltage
mot.Is   = 5;                                   %   stall current @ nom voltage
mot.w0   = 350 * 30 * rpm2rads;                 %   no-load speed @ nom voltage
mot.I0   = 0.3;                                 %   no-load current @ nom voltage

```

```
mot.R    = mot.UN/mot.Is;                        %   armature resistance
mot.L    = NaN;                                  %   armature inductance
mot.Kt   = mot.taus/mot.Is;                      %   torque constant
mot.Ke   = (mot.UN - mot.R*mot.I0)/(mot.w0);     %   back-EMF constant
mot.eta  = NaN;                                  %   motor efficiency
mot.PN   = NaN;                                  %   nominal output power
mot.IN   = NaN;                                  %   nominal current
mot.tauN = NaN;                                  %   nominal torque

%   dimensions
mot.rot.h   = 30.7e-3;                           %   rotor height
mot.rot.r   = 0.9 * 17e-3;                       %   rotor radius

mot.stat.h  = 68.1e-3;                           %   stator height
mot.stat.r  = 17e-3;                             %   stator radius

%   center of mass (CoM) position
mot.rot.xb  = 0;                                 %   (left) rot CoM x-pos in body frame
mot.rot.yb  = 42.7e-3;                           %   (left) rot CoM y-pos in body frame
mot.rot.zb  = -7e-3;                             %   (left) rot CoM z-pos in body frame

mot.stat.xb = 0;                                 %   (left) stat CoM x-pos in body frame
mot.stat.yb = 52.1e-3;                           %   (left) stat CoM y-pos in body frame
mot.stat.zb = -7e-3;                             %   (left) stat CoM z-pos in body frame

%   mass
mot.m       = 0.215;                             %   total motor mass
mot.rot.m   = 0.35 * mot.m;                      %   rotor mass
mot.stat.m  = mot.m - mot.rot.m;                 %   stator mass

%   moment of inertias (MoI) wrt principal axes
mot.rot.Ixx  = mot.rot.m/12 * (3*mot.rot.r^2 + mot.rot.h^2);    %   MoI along r dir
mot.rot.Iyy  = mot.rot.m/2 * mot.rot.r^2;                       %   MoI along h dim
mot.rot.Izz  = mot.rot.Ixx;                                     %   MoI along r dir

mot.stat.Ixx = mot.stat.m/12 * (3*mot.stat.r^2 + mot.stat.h^2); %   MoI along r dir
mot.stat.Iyy = mot.stat.m/2 * mot.stat.r^2;                     %   MoI along h dir
mot.stat.Izz = mot.stat.Ixx;                                    %   MoI along r dir

%   viscous friction coeff (motor side)
mot.B  = mot.Kt*mot.I0/mot.w0;

%%  Gearbox data

gbox.N = 30;        %   reduction ratio
gbox.B = 0.025;     %   viscous friction coeff (load side)

%%  Battery data

%   electrical data
batt.UN = 11.1;     %   nominal voltage

%   dimensions
batt.w = 136e-3;    %   battery pack width
batt.h = 26e-3;     %   battery pack height
batt.d = 44e-3;     %   battery pack depth

%   center of mass (CoM) position
batt.xb = 0;        %   CoM x-pos in body frame
batt.yb = 0;        %   CoM y-pos in body frame
batt.zb = 44e-3;    %   CoM z-pos in body frame

%   mass
```

```
batt.m = 0.320;

%   moment of inertias (MoI) wrt principal axes
batt.Ixx = batt.m/12 * (batt.w^2 + batt.h^2);   %   MoI along d dim
batt.Iyy = batt.m/12 * (batt.d^2 + batt.h^2);   %   MoI along w dim
batt.Izz = batt.m/12 * (batt.w^2 + batt.d^2);   %   MoI along h dim

%% H-bridge PWM voltage driver data

drv.Vbus = batt.UN;                     %   H-bridge DC bus voltage
drv.pwm.bits = 8;                       %   PWM resolution [bits]
drv.pwm.levels = 2^drv.pwm.bits;        %   PWM levels
drv.dutymax = drv.pwm.levels-1;         %   max duty cycle code
drv.duty2V = drv.Vbus/drv.dutymax;      %   duty cycle code (0-255) to voltage
drv.V2duty = drv.dutymax/drv.Vbus;      %   voltage to duty cycle code (0-255)

%% Wheel data

%   dimensions
wheel.h = 26e-3;                %   wheel height
wheel.r = 68e-3/2;              %   wheel radius

%   center of mass (CoM) position
wheel.xb = 0;                   %   (left) wheel CoM x-pos in body frame
wheel.yb = 100e-3;              %   (left) wheel CoM y-pos in body frame
wheel.zb = 0;                   %   (left) wheel CoM z-pos in body frame

%   mass
wheel.m = 50e-3;

%   moment of inertias (MoI) wrt principal axes
wheel.Ixx = wheel.m/12 * (3*wheel.r^2 + wheel.h^2);     %   MoI along r dim
wheel.Iyy = wheel.m/2 * wheel.r^2;                      %   MoI along h dim
wheel.Izz = wheel.Ixx;                                  %   MoI along r dim

%   viscous friction coeff
wheel.B = 0.0015;

%% Chassis data

%   dimensions
chassis.w = 160e-3;       %   frame width
chassis.h = 119e-3;       %   frame height
chassis.d = 80e-3;        %   frame depth

%   center of mass (CoM) position
chassis.xb = 0;           %   CoM x-pos in body frame
chassis.yb = 0;           %   CoM x-pos in body frame
chassis.zb = 80e-3;       %   CoM x-pos in body frame

%   mass
chassis.m = 0.456;

%   moment of inertias (MoI) wrt principal axes
chassis.Ixx = chassis.m/12 * (chassis.w^2 + chassis.h^2);   %   MoI along d dim
chassis.Iyy = chassis.m/12 * (chassis.d^2 + chassis.h^2);   %   MoI along w dim
chassis.Izz = chassis.m/12 * (chassis.w^2 + chassis.d^2);   %   MoI along h dim

%% Body data

%   mass
body.m = chassis.m + batt.m + 2*mot.stat.m;

```

```
%   center of mass (CoM) position
body.xb = 0;                                            %   CoM x-pos in body frame
body.yb = 0;                                            %   CoM y-pos in body frame
body.zb = (1/body.m) * (chassis.m*chassis.zb + ...      %   CoM z-pos in body frame
    batt.m*batt.zb + 2*mot.stat.m*mot.stat.zb);

%   moment of inertias (MoI) wrt principal axes
body.Ixx = chassis.Ixx + chassis.m*(body.zb - chassis.zb)^2 + ...       %   MoI along d dim
    batt.Ixx + batt.m*(body.zb - batt.zb)^2 + ...
    2*mot.stat.Ixx + ...
    2*mot.stat.m*(mot.stat.yb^2 + (body.zb - mot.stat.zb)^2);

body.Iyy = chassis.Iyy + chassis.m*(body.zb - chassis.zb)^2 + ...       %   MoI along w dim
    batt.Iyy + batt.m*(body.zb - batt.zb)^2 + ...
    2*mot.stat.Iyy + ...
    2*mot.stat.m*(body.zb - mot.stat.zb)^2;

body.Izz = chassis.Izz + batt.Izz + ...                                 %   MoI along h dim
    2*mot.stat.Izz + 2*mot.stat.m*mot.stat.yb^2;

%% Sensors data - Hall-effect encoder

%   Hall-effect encoder
sens.enc.ppr = 16*4;    %   pulses per rotation at motor side (w/ quadrature decoding)
sens.enc.pulse2deg = 360/sens.enc.ppr;
sens.enc.pulse2rad = 2*pi/sens.enc.ppr;
sens.enc.deg2pulse = sens.enc.ppr/360;
sens.enc.rad2pulse = sens.enc.ppr/2/pi;

%% Sensors data - MPU6050 (accelerometer + gyro)

%   center of mass (CoM) position
sens.mpu.xb = 0;
sens.mpu.yb = 0;
sens.mpu.zb = 13.5e-3;

%       MPU6050 embedded accelerometer specs
sens.mpu.acc.bits = 16;
sens.mpu.acc.fs_g = 16;                                                 %   full-scale in "g" units
sens.mpu.acc.fs = sens.mpu.acc.fs_g * g2ms2;                            %   full-scale in [m/s^2]
sens.mpu.acc.g2LSB = floor(2^(sens.mpu.acc.bits-1)/sens.mpu.acc.fs_g);  %   sensitivity [LSB/g]
sens.mpu.acc.ms22LSB = sens.mpu.acc.g2LSB * ms22g;                      %   sensitvity [LSB/(m/s^2)]
sens.mpu.acc.LSB2g = sens.mpu.acc.fs_g/2^(sens.mpu.acc.bits-1);         %   out quantization [g/LSB]
sens.mpu.acc.LSB2ms2 = sens.mpu.acc.LSB2g * g2ms2;                      %   out quantization [ms2/LSB]
sens.mpu.acc.bw = 94;                                                   %   out low-pass filter BW [Hz]
sens.mpu.acc.noisestd = 400e-6*sqrt(100);                               %   output noise std [g-rms]
sens.mpu.acc.noisevar = sens.mpu.acc.noisestd^2;                        %   output noise var [g^2]

%       MPU6050 embdedded gyroscope specs
sens.mpu.gyro.bits = 16;
sens.mpu.gyro.fs_degs = 250;                                            %   full scale in [deg/s (dps)]
sens.mpu.gyro.fs = sens.mpu.gyro.fs_degs * deg2rad;                     %   full scale in [rad/s]
sens.mpu.gyro.degs2LSB = floor(2^(sens.mpu.gyro.bits-1)/sens.mpu.gyro.fs_degs); %   sensitivity [LSB/degs]
sens.mpu.gyro.rads2LSB = sens.mpu.gyro.degs2LSB * rad2deg;              %   sensitivity [LSB/rads]
sens.mpu.gyro.LSB2degs = sens.mpu.gyro.fs_degs/2^(sens.mpu.gyro.bits-1);%   out quantization [degs/LSB]
sens.mpu.gyro.LSB2rads = sens.mpu.gyro.LSB2degs * deg2rad;              %   out quantization [rads/LSB]
sens.mpu.gyro.bw = 98;                                                  %   out low-pass filter BW [Hz]
sens.mpu.gyro.noisestd = 5e-3*sqrt(100);                                %   output noise std [degs-rms]
sens.mpu.gyro.noisevar = sens.mpu.acc.noisestd ^2;                      %   output noise var [degs^2]
```

5.2 Balance–and–position state–space control using LQR methods

1. For the implementation of a state–space balance–and–position controller, it is first necessary to estimate the robot state $\boldsymbol{x} = [\gamma,\, \vartheta,\, \dot{\gamma},\, \dot{\vartheta}]^T$ from the measurements provided by the onboard sensors, namely the incremental encoder and the MPU (accelerometer and gyroscope). For such purpose, consider to implement a "simple" state observer as follows:

   – for estimating the robot body tilt angle ϑ, use the complementary filtering approach described in Sec. 3. Consider to use a pair of first–order complementary filters such as (72). The filters must be discretized, since the whole control system operates in the discrete–time domain. Say $H(z)$ the discrete equivalent of $H(s)$, obtained with any discretization method. From (71) it follows that

   $$\hat{\vartheta} \;=\; H(z)\,\hat{\vartheta}_a \;+\; [1 - H(z)]\,\hat{\vartheta}_g \;=\; \hat{\vartheta}_g \;+\; H(z)\,(\hat{\vartheta}_a \;-\; \hat{\vartheta}_g) \tag{84}$$

   where $\hat{\vartheta}_a$ is computed as specified in (67), and $\hat{\vartheta}_g$ by discrete–time integration of the gyroscope output y_g. A possible Simulink implementation of the complementary filtering (84) is shown in Fig. 10a. A tentative value for the filter cut–off frequency $f_c = 1/(2\pi\, T_c)$ is $0.35\,\mathrm{Hz}$.

   Notes:

   (a) it is worth to notice here that if both $H(s)$ and the integrator in (69) are discretized with the *backward Euler* discretization method, then the generic implementation (84) can be further simplified as follows. The backward Euler discretization of $H(s)$ yields

   $$H(z) \;=\; \frac{C}{1 - (1 - C)\,z^{-1}} \qquad \text{with} \qquad C = \frac{T}{T_c + T} \tag{85}$$

   where T denotes the sampling time. Hence, from (84) it follows that

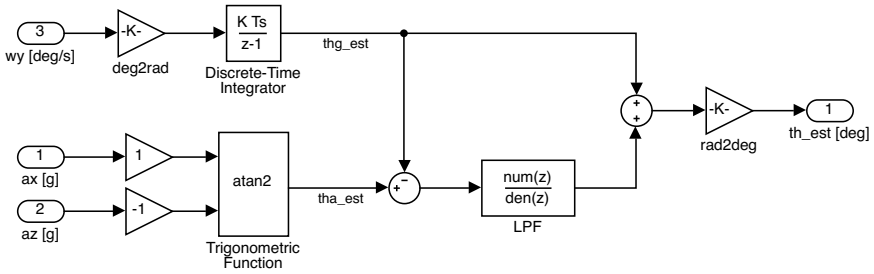
   $$\hat{\vartheta} \;=\; \frac{C}{1 - (1 - C)\,z^{-1}}\,\hat{\vartheta}_a \;+\; \frac{(1 - C)\,(1 - z^{-1})}{1 - (1 - C)\,z^{-1}} \cdot \frac{T}{1 - z^{-1}}\,y_g \tag{86}$$

   where it has been used the fact that $\hat{\vartheta}_g$ is obtained by integration of y_g, using the discrete–time integrator $T/(1 - z^{-1})$ (backward Euler discretization of the continuous–time integrator $1/s$). In time domain, the expression (86) becomes

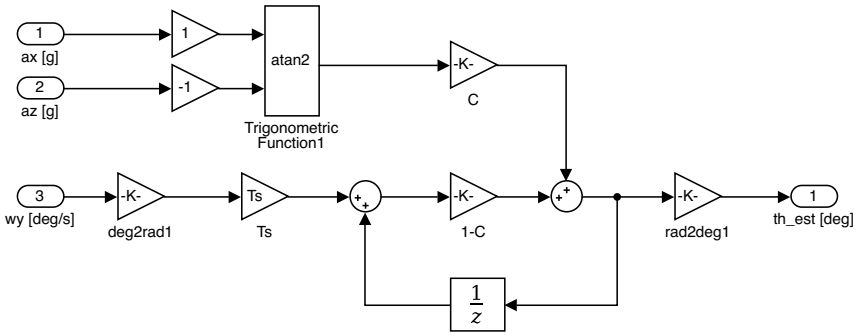
   $$\begin{aligned} \hat{\vartheta}[k] \;&=\; (1 - C)\,\hat{\vartheta}[k-1] \;+\; C\,\hat{\vartheta}_a[k] \;+\; (1 - C)\,T\,y_g[k] \\ &=\; C\,\hat{\vartheta}_a[k] \;+\; (1 - C)\,\Big(\hat{\vartheta}[k-1] \;+\; T\,y_g[k]\Big) \end{aligned} \tag{87}$$

   A possible Simulink implementation of the complementary filtering (87) is shown in Fig. 10b.

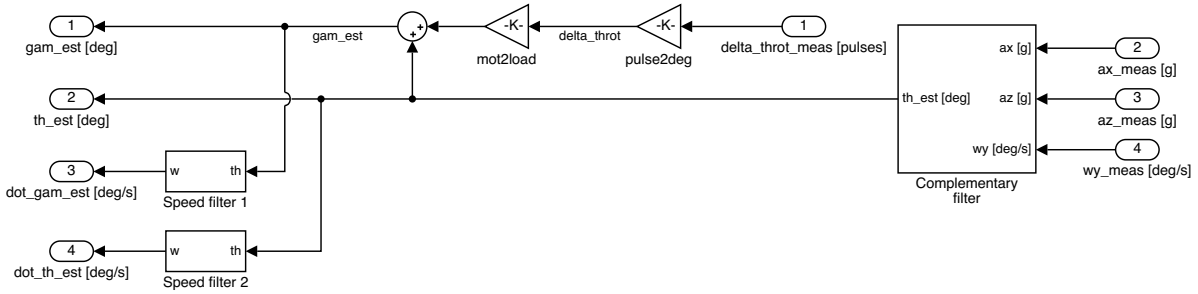
   (b) when using the balancing robot model implemented with the "chain of integrators" approach, an algebraic loop could arise if the complementary filters are implemented either as in Fig. 10a, and the transfer function $H(z)$ of the discretized low–pass filter is not strictly proper, or as in Fig. 10b.
   In fact, in these situations, the estimated tilt angle depends instantaneously on the acceleration output of the robot model; but the robot acceleration depends instantaneously



(a) Tilt estimation by complementary filtering (implementation based on (84))



(b) Tilt estimation by complementary filtering (implementation based on (87))



(c) "Simple" state observer

Figure 10: Possible Simulink implementation of the "simple" state observer.

on the control command, which in turns depends instantaneously (through the multiplication by the feedback gain) from the estimated tilt angle, and hence an algebraic loop originates[2].

To avoid the algebraic loop issue when using the robot model based on the "chain of integrators" implementation, consider to use the complementary filtering scheme shown in Fig. 10a, and choose a discrete–time low–pass filter with a strictly proper transfer function (e.g. the filter obtain by discretizing a continuous–time, first–order low–pass filter with the Forward Euler method).

– for estimating the wheel angle γ, use the identity $\Delta\vartheta_{rot} = N(\gamma - \vartheta)$ to obtain the

[2]The real reason for the existence of the algebraic loop is that both the models of the motor voltage driver (see Fig. 8) and the inertial sensors (see Fig. 6) have no dynamics, i.e. the outputs instantaneously depend on the inputs. This idealized situation never occurs in practice, and typically both the actuators and sensors have a low-pass dynamics that should be taken into account when modelling them for simulation purposes. Indeed, this is what has been done for voltage driver model of the Quanser DC servomotor used in the previous laboratory activities. Unfortunately, in the case of the balancing robot, there is no explicit knowledge about the voltage driver bandwidth, and this was the original reason that motivated the choice of neglecting the driver dynamics in the model proposed in this handout.

estimate

$$\hat{\gamma} \;=\; \Delta\vartheta_{rot}/N \;+\; \hat{\vartheta} \tag{88}$$

where $\Delta\vartheta_{rot}$ is the rotor angular displacement measured by the encoder. A possible Simulink implementation of (88) is shown in Fig. 10c.

– use a "real derivative" filter of the type

$$H_\omega(z) \;=\; \frac{1 - z^{-N}}{N\,T} \qquad \text{with} \qquad N = 3 \tag{89}$$

to obtain the angular speeds estimates $\dot{\hat{\vartheta}}$ and $\dot{\hat{\gamma}}$ from $\hat{\vartheta}$ and $\hat{\gamma}$.

2. Design a discrete–time state–space controller that simultaneously stabilizes the robot body to its upward vertical position, and guarantees the *nominal* perfect tracking of a constant wheel angle position set–point γ^*. For such purpose, consider first to discretize the continuous–time plant model (57) with the exact discretization method, and sampling time equal to the controller sampling time $T = 0.01\,\mathrm{s}$; let

$$\begin{cases} \boldsymbol{x}[k+1] \;=\; \boldsymbol{\Phi}\,\boldsymbol{x}[k] \;+\; \boldsymbol{\Gamma}\,u[k] \\ \qquad y[k] \;=\; \boldsymbol{H}\,\boldsymbol{x}[k] \end{cases} \tag{90}$$

denote the discretized plant model. For the design of the tracking controller, the model output y has to be equal to the signal to track, namely the wheel angle position γ. Therefore, the matrix $\boldsymbol{H}$ is chosen equal to $\boldsymbol{H} = [1,\, 0,\, 0,\, 0]$.

The discrete–time state–space control law has the following structure:

$$u[k] \;=\; N_u\,r[k] \;-\; \boldsymbol{K}\,(\hat{\boldsymbol{x}}[k] - \boldsymbol{N}_x\,r[k]) \;=\; -\boldsymbol{K}\,\boldsymbol{x}[k] \;+\; \underbrace{(N_u \;+\; \boldsymbol{K}\,N_x)}_{=\,N_r}\,r[k] \tag{91}$$

where $r = \gamma^*$ is the wheel angle reference signal, and $\boldsymbol{x}$ is the state vector estimated with the simple state observer designed in point 1. As explained in the handout of laboratory activity 2, the feedforward gains $\boldsymbol{N}_x$ and N_u are determined by solving the following set of linear equations:

$$\begin{bmatrix} \boldsymbol{\Phi} - \boldsymbol{I} & \boldsymbol{\Gamma} \\ \boldsymbol{H} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ N_u \end{bmatrix} \;=\; \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix} \tag{92}$$

The feedback gain $\boldsymbol{K}$ can instead be computed with LQR methods, in order to minimise the discrete–time quadratic cost function

$$J \;=\; \sum_{k=0}^{+\infty} \boldsymbol{x}^T[k]\,\boldsymbol{Q}\,\boldsymbol{x}[k] \;+\; \rho\,r\,u^2[k] \tag{93}$$

Select the cost weights $\boldsymbol{Q}$ and r according to the *Bryson's rule*. At steady–state, it is desired to have:

$$|\gamma - \gamma^*| \;<\; \pi/36\;(5\,\mathrm{deg})\,, \qquad |\vartheta| \;<\; \pi/360\;(0.5\,\mathrm{deg})\,, \qquad |u| \;<\; 1\,\mathrm{V} \tag{94}$$

Therefore, according to the Bryson's rule, the cost weights $\boldsymbol{Q}$ and r are selected as follows:

$$\boldsymbol{Q} = \mathrm{diag}\left\{\frac{1}{\bar{\gamma}^2},\ \frac{1}{\bar{\vartheta}^2},\ 0,\ 0\right\}, \qquad r = \frac{1}{\bar{u}^2} \tag{95}$$

where

$$\bar{\gamma} = \pi/18\,, \qquad \bar{\vartheta} = \pi/360\,, \qquad \bar{u} = 1 \tag{96}$$

In the $\boldsymbol{Q}$ matrix, note that the weights of the two angular velocities $\dot{\gamma}$ and $\dot{\vartheta}$ have been set equal to zero. The extra weight ρ in (93) is used to adjust the relative weighting between the state and input contributions to the total cost function value. Consider the following choices for such weight:

$$\rho \in \{\,500,\ 5000\,\} \tag{97}$$

For the computation of the feedback matrix, use the routine `dlqr` of the Control System Toolbox (CST).

3. Validate the design of point 2 in simulation, by using the Simulink model of the balancing robot developed in Sec. 5.1. A possible Simulink implementation of the whole control system is shown in Fig. 11. In the model of Fig. 11a, the state observer is implemented as described in point 1. Test the controller in the following situations:

   – initial state $\boldsymbol{x}(0) = [0,\ \pi/36,\ 0,\ 0]^T$ (i.e. initial body tilt angle equal to $\vartheta(0) = 5\,\mathrm{deg}$), reference input equal to zero, and no load disturbance (i.e. disturbance entering at the plant input).

     This test is aimed to verify that the controller is capable of restoring the balance after that the robot is released from a position off the upward vertical equilibrium.

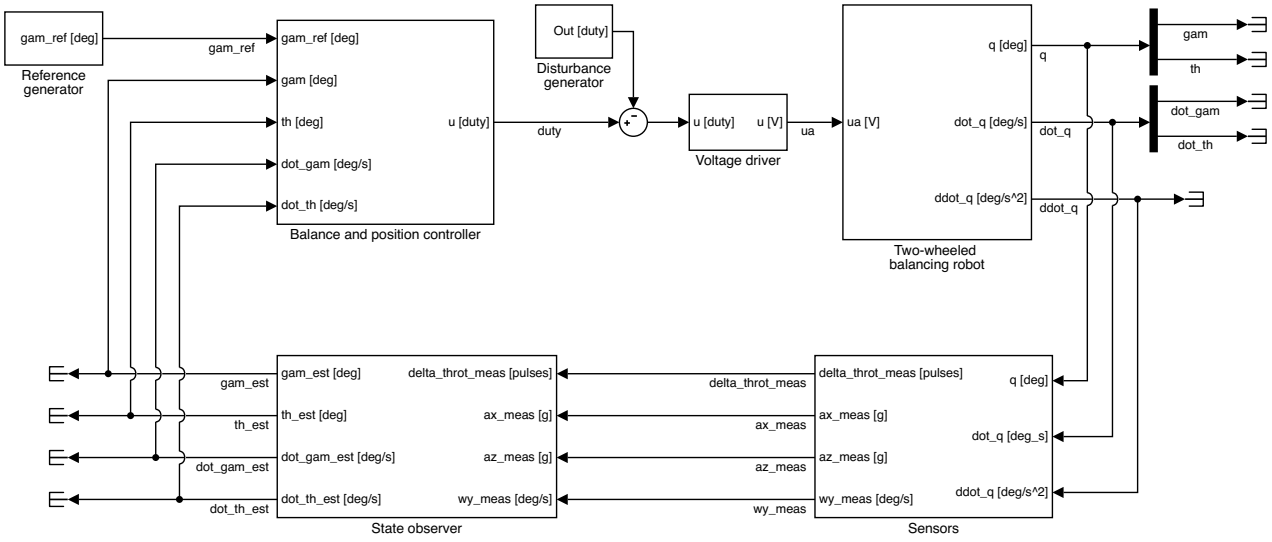
   – initial state $\boldsymbol{x}(0) = [0,\ 0,\ 0,\ 0]^T$, step reference input applied at $t = 0$ with amplitude $\gamma^* = 0.1/r\,\mathrm{rad}$, where r is the wheel radius (this choice corresponds to a longitudinal position displacement of $10\,\mathrm{cm}$), and no load disturbance.

     This test is aimed to verify that the controller guarantees perfect tracking of the constant position set–point in the nominal case, when no external disturbances are present.

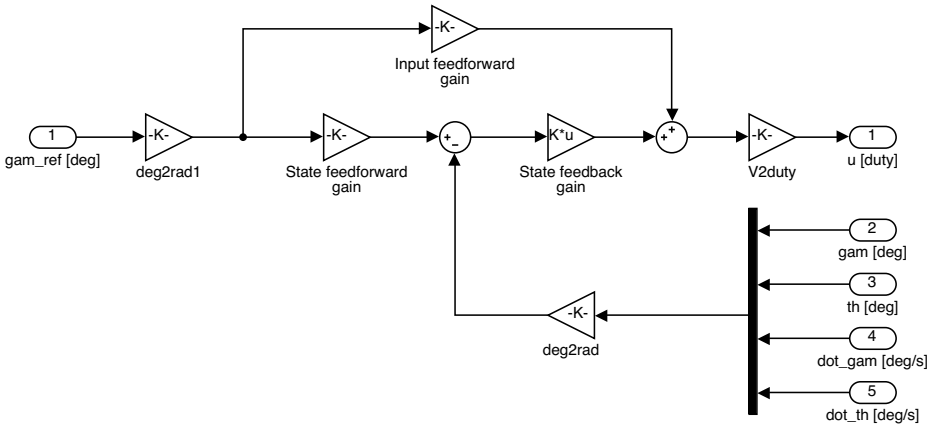
   – initial state and reference input as in the previous point, and a load disturbance of amplitude $5.0/k_{duty\to V} \approx 115$ (where $k_{duty\to V}$ is the voltage driver input–to–output conversion gain), applied at the voltage driver input at $t = 10\,\mathrm{s}$. This is equivalent to a voltage disturbance of $5\,\mathrm{V}$ applied at the motor input.

     This test is aimed to verify that the controller is unable to guarantee the perfect tracking of the constant position set–point when an external disturbance is present. In this case, the disturbance is considered as an equivalent constant voltage disturbance entering at the plant input. Such disturbance could result from the application of a constant longitudinal force to the robot, similarly to what happens when the robot is forced to climb an inclined plane.

4. Repeat the design of point 3 by introducing the integral action in the controller, in order to achieve *robust* tracking of constant wheel angle position set-points.



(a) Balance–and–position state–space control system



(b) State–space controller

Figure 11: Possible Simulink implementation of the balance–and–position state–space control system (for the *nominal* perfect tracking of constant position set–points).

By introducing the integral action in (91), the control law becomes

$$\begin{cases} x_I[k+1] \;=\; x_I[k] \;+\; (\, y[k] \;-\; r[k]\,) \\ \qquad u[k] \;=\; N_u\, r[k] \;-\; \boldsymbol{K}\,(\boldsymbol{x}[k] - \boldsymbol{N}_x\, r[k]) - K_I\, x_I[k] \end{cases} \tag{98}$$

where x_I is the integrator state variable[3]. After introducing the augmented state vector $\boldsymbol{x}_e = [x_I,\, \boldsymbol{x}]^T$ of the augmented state system

$$\Sigma_e \;:\; \begin{bmatrix} x_I[k+1] \\ \boldsymbol{x}[k+1] \end{bmatrix} \;=\; \underbrace{\begin{bmatrix} 1 & \boldsymbol{H} \\ \boldsymbol{0} & \boldsymbol{\Phi} \end{bmatrix}}_{\triangleq\, \boldsymbol{\Phi}_e} \underbrace{\begin{bmatrix} x_I[k] \\ \boldsymbol{x}[k] \end{bmatrix}}_{\triangleq\, \boldsymbol{x}_e} + \underbrace{\begin{bmatrix} 0 \\ \boldsymbol{\Gamma} \end{bmatrix}}_{\triangleq\, \boldsymbol{\Gamma}_e} u[k] \;-\; \begin{bmatrix} 1 \\ \boldsymbol{0} \end{bmatrix} r[k] \tag{99}$$

[3] In (98), note that the tracking error is defined as $e[k] = y[k] - r[k]$.

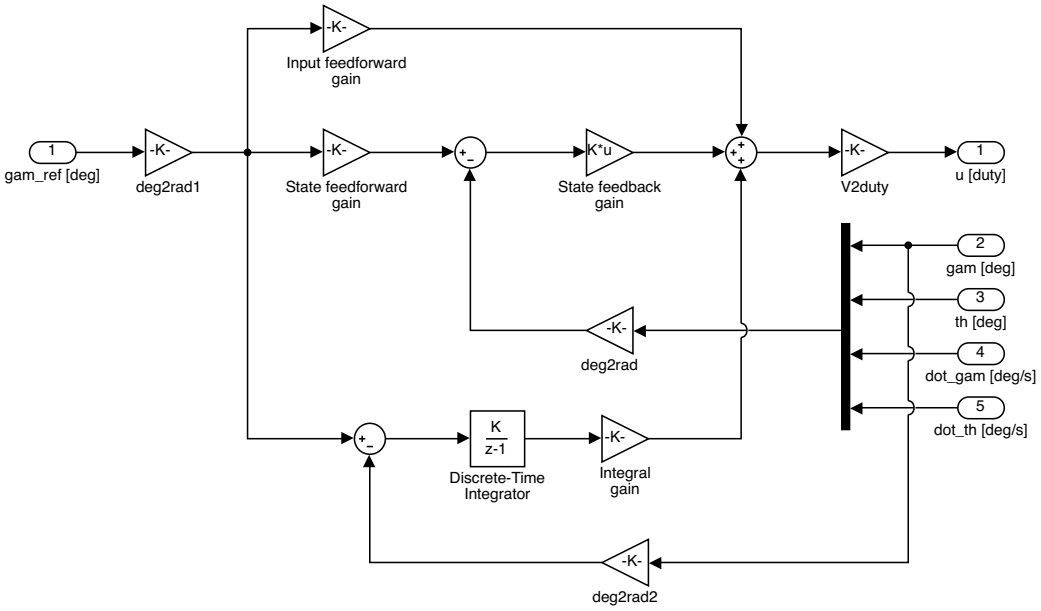


Figure 12: Possible Simulink implementation of the state–space controller, for the balance control and the *robust* perfect tracking of constant position set–points, using the integral action approach.

the control law (98) can be rewritten as follows:

$$u[k] \;=\; -\boldsymbol{K}_e\,\boldsymbol{x}_e[k] \;+\; (N_u \;+\; \boldsymbol{K}\,\boldsymbol{N}_x)\,r[k] \;=\; -\boldsymbol{K}_e\,\boldsymbol{x}_e[k] \;+\; N_r\,r[k] \qquad (100)$$

The augmented state feedback matrix $\boldsymbol{K}_e = [K_I,\, \boldsymbol{K}]^T$ can be designed again by resorting to LQR methods. Compared to the design of point 2, in this case the cost matrix $\boldsymbol{Q}$ contains an extra weight q_{11} for the integrator state variable x_I. This weight must be chosen different from zero, otherwise the LQR cannot be designed. However, the weight cannot be chosen with the Bryson's rule, since there is no reasonable and immediate way to identify the maximum deviation of the integrator state from its steady–state value. In practice, the weight q_{11} must be chosen by trial and error. Two possible choices to consider for such weight are:

$$q_{11} \;\in\; \{0.1,\; 1\} \qquad (101)$$

Choose the remaining weights as suggested in point 2.

A possible Simulink implementation of the control law (98) is shown in Fig. 12. In (98), note that the integrator state x_I consists of the discrete–time *accumulation* of the tracking error $e[k] = y[k] - r[k]$. Therefore, the *Integrator method* property of the *Discrete–Time Integrator* block in Fig. 12 must be set as "*Accumulation: Forward Euler*" (i.e. the integrator has to be implemented as a discrete–time forward Euler accumulator).

5. Validate the design of point 4 in simulation, using the Simulink model of the whole control system prepared in point 3 (simply replace the controller block of Fig. 11 with that of Fig. 12).

   Repeat the same tests of point 3. In particular, verify that the controller is capable of perfectly tracking a constant position reference at steady state, even in presence of a constant load disturbance.

6 Laboratory assignments: experimental tests

1. Prepare a Simulink model for testing the balance–and–position controllers designed in Sec. 5.2 on the balancing robot available in laboratory. For such purpose, it is sufficient to proceed as follows:

   – make a copy of the Simulink model prepared in point 3 of Sec. 5.2 for the numerical simulations (see also Fig. 11).

   – replace the models of the balancing robot and the sensors (encoder and MPU) with the blocks of the Balancing Robot Toolbox (BRT) that allow to interface with the robot hardware.

   – configure the model parameters to enable the execution on the balancing robot micro–controller unit (MCU), according to the details provided in the introductory guide to the experimental setup (laboratory guide 2).

     In particular, in the *Hardware Implementation* settings, select *Arduino Mega 2560* as the *Hardware Board*, and *Automatically* as the detection method for the *Host–board connection* port. Regarding the Solver parameters, choose a *Fixed–step discrete (no continuous states)* solver, and a sample time (fixed–step size) equal to $T = 0.01\,\mathrm{s}$.

   A possible implementation is shown in Fig. 13. In addition to the base controller, the proposed implementation includes some extra "logic", to enable the generation of the position reference and the load disturbance via the two pushbuttons available on the robot. The logic operates as follows:

   – the controller is enabled by pressing either the pushbutton 1 or 2.

     This logic is implemented by using two *BRT* → *Utilities* → *Up–Counter (with upper bound)* counters to detect when the pushbuttons are pressed for the first time. The upper bounds of the two counters are both set equal to 1. The controller is enabled when at least one counter output is equal to 1.

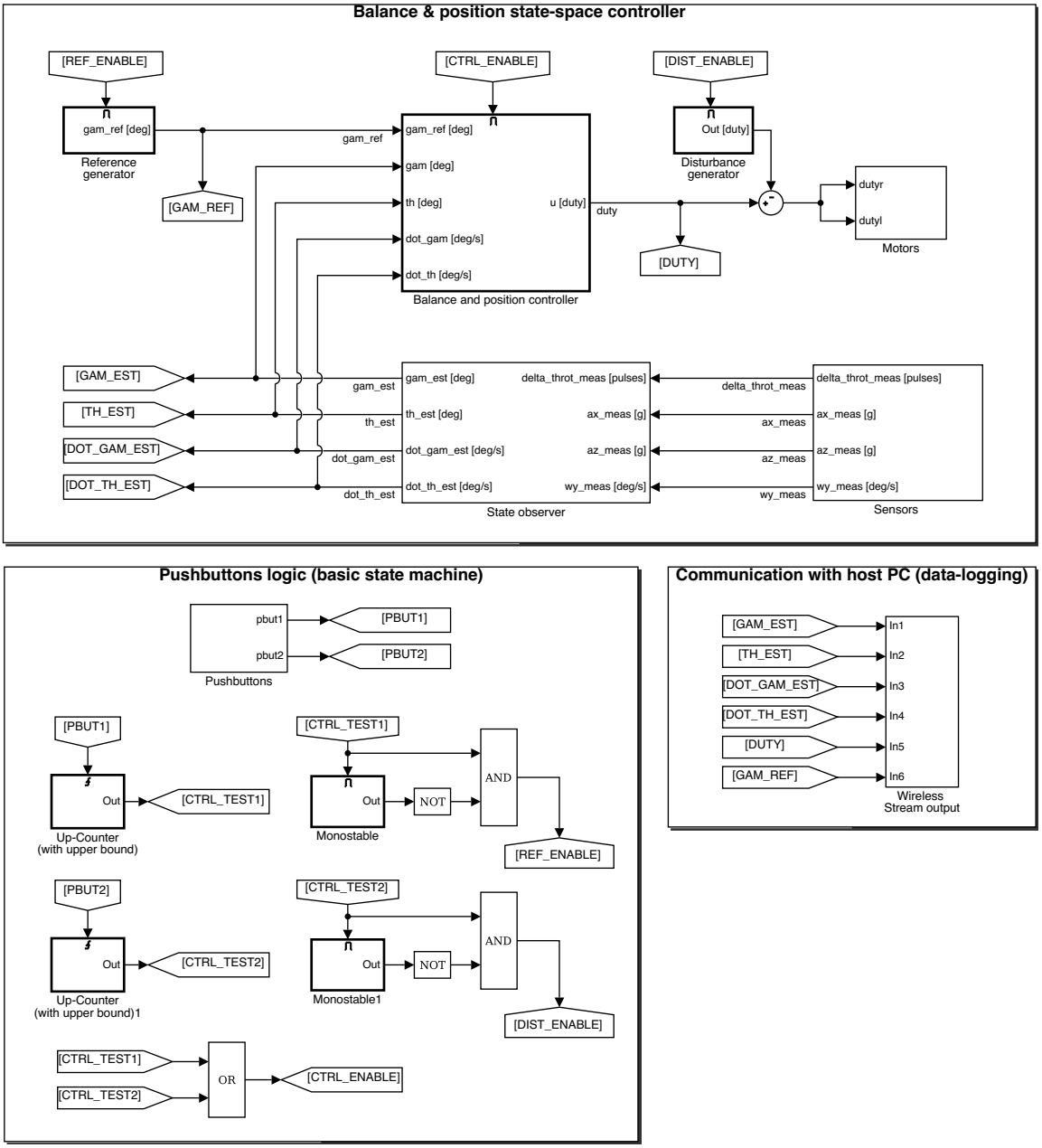
     Note that the two up–counters (with upper bound set to 1) behave as two set–reset (SR) latches, with the S inputs driven by the two pushbuttons.

   – a constant reference signal is generated a certain amount of time after pressing the pushbutton 1.

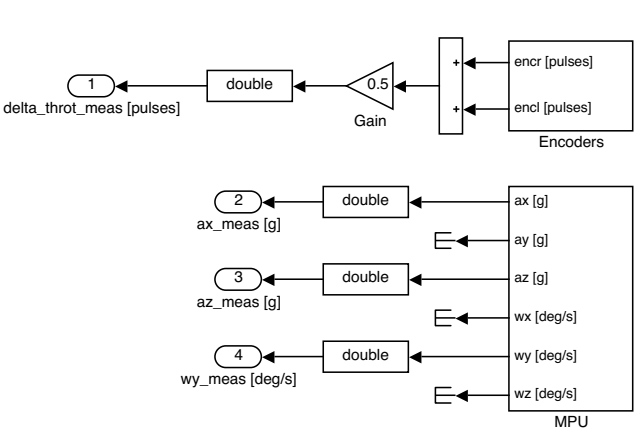
     This logic is implemented by enabling a *BRT* → *Utilities* → *Monostable* block with the output of the up–counter triggered by the push–button 1. In this way, a pulse of specified duration is generated after pressing the pushbutton. Before this event, or after the pulse expiration, the output of the monostable is equal to zero. Therefore, the condition for enabling the generation of the position reference is that the pushbutton 1 has been pressed (i.e. the output of the corresponding up–counter is equal to 1), and the output of the monostable is not equal to 1.

     Consider to set a pulse duration of at least $10\,\mathrm{s}$ in the monostable: this amount of time should be sufficient for the robot to reach a stable vertical balance, before moving according to the provided position reference.

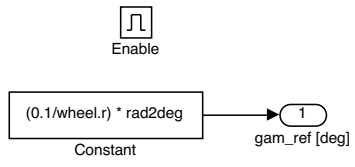
   – a constant load disturbance is generated a certain amount of time after pressing the pushbutton 2.



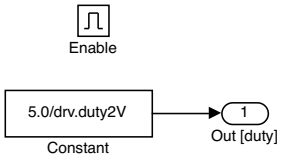
(a) Balance–and–position state–space control system



(b) Sensors interface



(c) Reference generator



(d) Disturbance generator

Figure 13: Possible Simulink implementation of the balance–and–position controller to be uploaded on the balancing robot MCU.

This logic is identical to that of the previous point (for enabling the generation of the position reference), except for the pushbutton used.

Both the balance–and–position controller, and the reference/disturbance generator are *enabled* blocks (use a *Ports & Subsystems* → *Enable* to create a subsystem with an enable port). When working with an enabled block, it is necessary to specify how to set its outputs when the block is disabled. For each output, this can be done by setting the properties of the *Sinks* → *Out* port appropriately (double–click on the port block to access its parameters). In particular, the *Output when disabled* option specifies to either hold or reset the output port value when the block is disabled. If the *held* option is selected, then an initial output value has to be provided (in the *Initial output* field). In the Simulink model of Fig. 13, the option *reset* is selected for the outputs of both the controller and the reference/disturbance generator.

In the sensors interface implementation of Fig. 13b, note that the angular displacement $\Delta\vartheta_{rot}$ of the rotor with respect to the stator (in the the planar robot approximation) is obtained as the average of the angular displacements $\Delta\vartheta_{rot,r}$ and $\Delta\vartheta_{rot,l}$ measured by the two encoders:

$$\Delta\vartheta_{rot} = \frac{\Delta\vartheta_{rot,r} + \Delta\vartheta_{rot,l}}{2} \tag{102}$$

Moreover, note that a *Signal Attributes* → *Data Type Conversion* has been used to convert the outputs of the *BRT* → *Sensors* → *Encoders* and *BRT* → *Sensors* → *MPU* blocks from their original data types (*uint32* and *single*, respectively) to the *double* type (in the block settings, select the option *double* in the *Output data type* drop–down list), which is the internal type adopted for controller implementation. This choice guarantees the best numerical accuracy; however, it is also demanding in terms of memory occupation and computational effort, which are both scarce resources on a typical embedded system. The *single* floating point data type could be an alternative option to consider in order to save space and computational resources.

2. Test the controller designed in point 2 of Sec. 5.2 on the balancing robot, using the model prepared in point 1.

   For the tests, use a constant position reference equal to $\gamma^* = 0.1/r\,\mathrm{rad}$, where r is the wheel radius, which corresponds to a longitudinal position displacement of $10\,\mathrm{cm}$, and a load disturbance equal to $u_d = 5.0/k_{duty\to V} \approx 115$, where $k_{duty\to V}$ is the PWM duty–cycle to voltage conversion gain, which corresponds to a voltage disturbance of $5.0\,\mathrm{V}$. The position reference and/or the load disturbance should be enabled at least $10\,\mathrm{s}$ after turning the controller on, to leave enough time to the robot to initially stabilize on the vertical position.

3. Test the controller designed in point 4 of Sec. 5.2 on the balancing robot, using the model prepared in point 1.

   For the tests, use the same position reference and load disturbance of the previous point 3.

4. (**optional**) From the experimental tests of points 2 and 3 it can be noticed that the balancing robot has the tendency of drifting laterally, and in practice it never moves along a perfectly straight line. This problem is caused by the fact that even if the two motors are driven by the same voltage command, they do not necessarily move by the same angle, because of unavoidable differences in the motor parameters, and the presence of friction and backlash in the mechanical transmission (gearboxes).

To avoid the lateral drift motion, an extra controller for the robot heading angle (yaw angle) ψ is required. The design of such controller is beyond the scope of this laboratory activity, and will be addressed in a possible follow–up (i.e. "Combined longitudinal and heading–angle state–space control of the balancing robot"). In the following, a simple implementation based on a PI controller is presented, for the only purpose of illustrating how to test the longitudinal controller under "more stable" working conditions. The proposed implementation is shown in Fig. 14 and 15. It consists of controlling the two motors with the following two voltage commands (for the right and left motors, respectively):

$$u_r \,=\, u_\Sigma \,+\, u_\Delta \,, \qquad u_l \,=\, u_\Sigma \,-\, u_\Delta \tag{103}$$

where u_Σ is a "common–mode" command generated by the longitudinal controller, and u_Δ a "differential–mode" command generated by the heading angle (yaw) controller. The longitudinal controller is the state–space controller designed in point 2 or 4 of Sec. 5.2. The heading angle controller is instead a simple PI regulator, as shown in Fig. 14c. For the purpose of this laboratory activity, the following proportional and integral gains can be used:

$$K_P \,=\, 3.3 \,, \qquad K_I \,=\, 0.7 \tag{104}$$

The heading angle (yaw) ψ is estimated in the block of Fig. 15b by using the expression (10), under the assumption that $\psi(0) = 0$, namely

$$\psi \,=\, \frac{r}{w}\,(\vartheta_r \,-\, \vartheta_l) \tag{105}$$

where ϑ_r and ϑ_l are the wheels angles derived from the encoders measurements. From (6) and (7), these quantities are equal to:

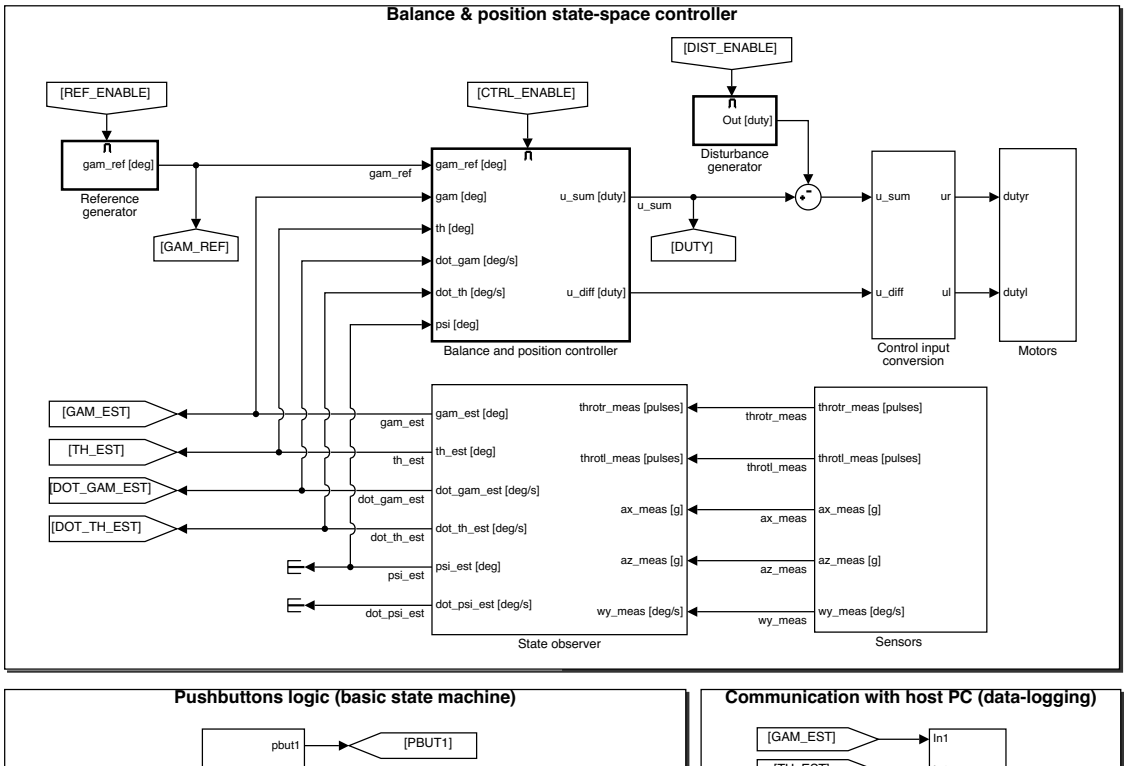
$$\vartheta_l \,=\, \frac{\Delta\vartheta_{rot,l}}{N} \,+\, \vartheta \,, \qquad \vartheta_r \,=\, \frac{\Delta\vartheta_{rot,r}}{N} \,+\, \vartheta \tag{106}$$

where $\Delta\vartheta_{rot,l}$ and $\Delta\vartheta_{rot,r}$ are the measurements provided by the two encoders.

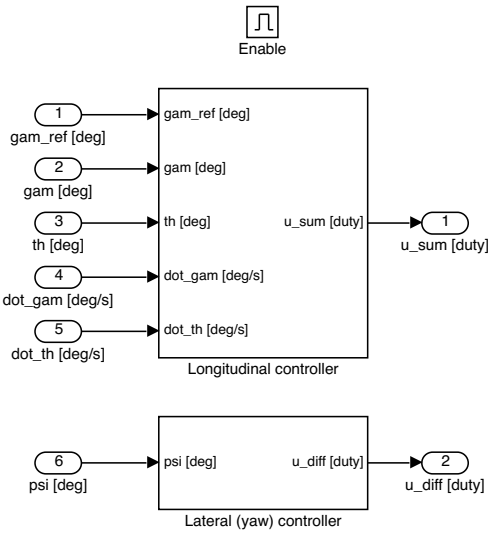
Instead, the variable γ is estimated by simply considering the average value of the two wheels angles, namely

$$\gamma \,=\, \frac{\vartheta_r \,+\, \vartheta_l}{2} \tag{107}$$

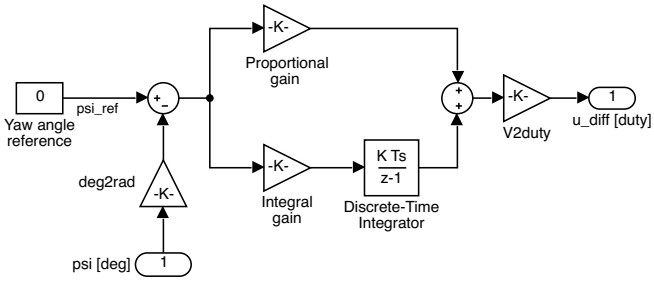
Consider to repeat the experimental tests of points 2 and 3 with the controller configuration proposed above.



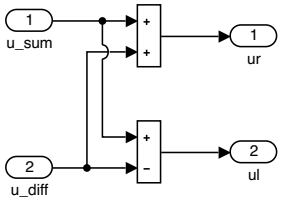
(a) Balance–and–position state–space control system



(b) Balance–and–position controller

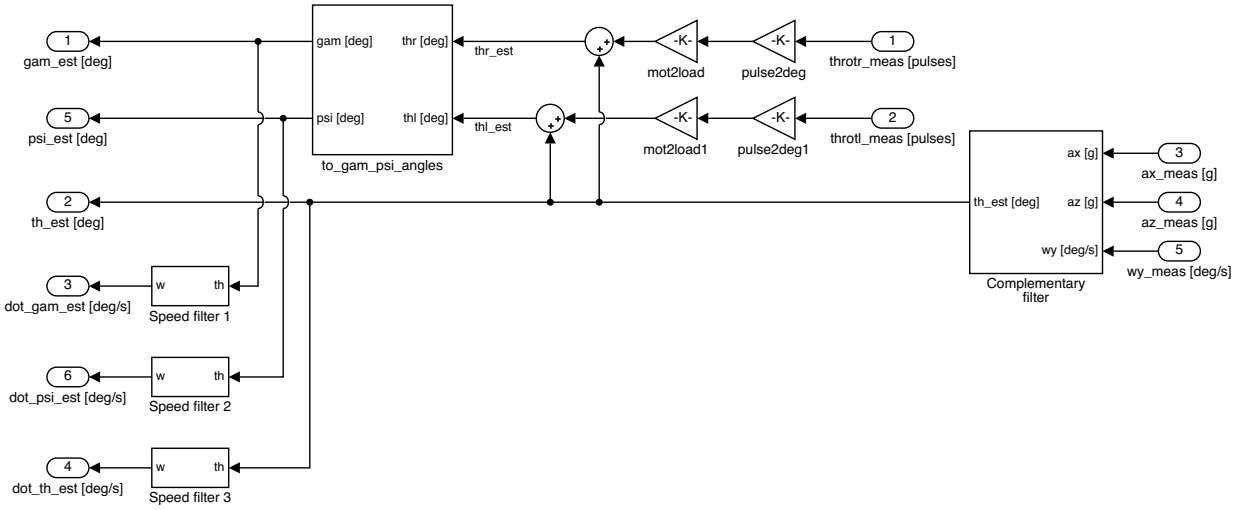


(c) Lateral (yaw) controller

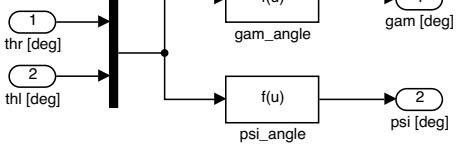


(d) Control input conversion

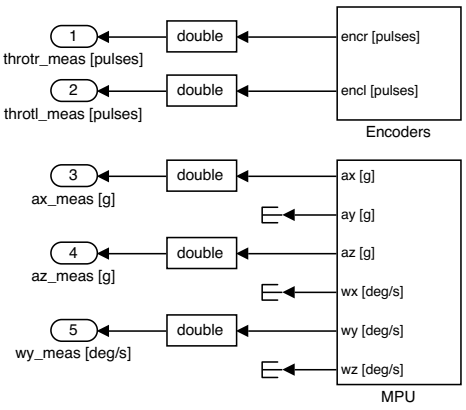
Figure 14: Possible Simulink implementation of a combined longitudinal and heading–angle controller for the balancing robot.



(a) State observer



(b) "to_gam_psi_angles" block



(c) Sensors interface

Figure 15: Possible Simulink implementation of a combined longitudinal and heading–angle controller for the balancing robot (cont'd).

7 Appendix

7.1 Simulink implementation of the electromechanical dynamics with *S–Functions*

The nonlinear dynamical model (54)–(55) can be implemented in Simulink using a MATLAB *S–Function*. Consult the Simulink documentation for the details regarding how to write a MATLAB S–Function, using either the *Level–1* or *Level–2* API (Application Programming Interface).

A possible Level–2 MATLAB S–Function implementation is reported in Listing 2. The corresponding block to be used in the Simulink model is the *User–Defined Functions* → *Level–2 MATLAB S–Function* block. The block has two parameters: the *S–function name* (required parameter) is the name of the MATLAB script containing the implementation of the S–function (i.e. `sfun_balrob_long_dyn.m`), while *Parameters* (optional parameter) are the extra parameters to be passed to the S–function (if required by the implementation). The Simulink model based on the S–function of Listing 2 is shown in Fig. 16.

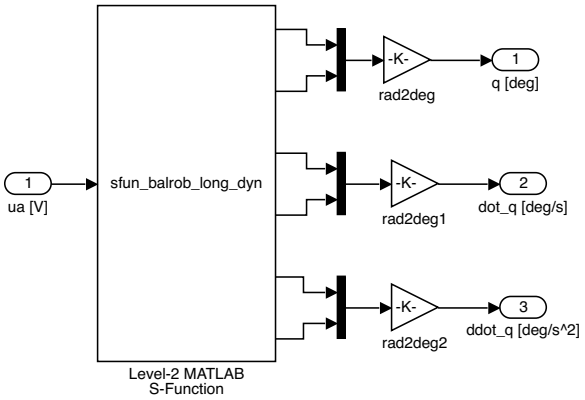


Figure 16: Simulink model implementation details: electromechanical dynamics (implemented by using a MATLAB S–Function).

Note that the S–function implementation specifies a block with 1 input (see line 11), 6 outputs (see line 12), 4 continuous states (see line 15) and 5 parameters (see line 37). Accordingly, the S–function block in the Simulink model is shown with 1 input (the armature voltage u_a) and 6 outputs (the vector of generalized variables $\boldsymbol{q}$, and its first two time derivatives $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$). The 5 expected parameters are the vector of initial conditions `x0` (see line 72), and the four data structures `body`, `mot`, `gbox` and `wheel` (see lines 145–148). The 4 continuous–state variables are the generalized variables in $\boldsymbol{q}$, and their derivatives in $\dot{\boldsymbol{q}}$. The acceleration vector $\ddot{\boldsymbol{q}}$ is computed by the function `get_acc` (see lines 140–242) according to the expression derived in (83).

Listing 2: `sfun_balrob_long_dyn.m`

```
1  function sfun_balrob_long_dyn(block)
2
3  setup(block);
4
5  end %function
6
7
8  function setup(block)
9
10 % Register number of ports
11 block.NumInputPorts  = 1;
12 block.NumOutputPorts = 2*3;
13
14 % Register number of continuous states
15 block.NumContStates = 2*2;
16
17 % Setup port properties to be inherited or dynamic
18 block.SetPreCompInpPortInfoToDynamic;
19 block.SetPreCompOutPortInfoToDynamic;
20
21 % Override input port properties
22 for k = 1:block.NumInputPorts,
23     block.InputPort(k).Dimensions        = 1;
24     block.InputPort(k).DatatypeID        = 0;  % double
25     block.InputPort(k).Complexity        = 'Real';
26     block.InputPort(k).DirectFeedthrough = false;
27 end;
28
29 % Override output port properties
30 for k = 1:block.NumOutputPorts,
31     block.OutputPort(k).Dimensions  = 1;
```

```
    block.OutputPort(k).DatatypeID  = 0; % double
    block.OutputPort(k).Complexity  = 'Real';
end;

% Register parameters
block.NumDialogPrms = 5;

% Register sample times
%  [0 offset]            : Continuous sample time
block.SampleTimes = [0 0];

% Specify the block simStateCompliance.
%    'DefaultSimState', < Same sim state as a built-in block
block.SimStateCompliance = 'DefaultSimState';

%   Register block methods
block.RegBlockMethod('Start',                       @Start);
block.RegBlockMethod('SetInputPortSamplingMode',    @SetInputPortSamplingMode);
block.RegBlockMethod('Outputs',                     @Outputs);      % Required
block.RegBlockMethod('Derivatives',                 @Derivatives);
block.RegBlockMethod('Terminate',                   @Terminate);    % Required

for k = 1:block.NumOutputPorts,
    block.OutputPort(k).SamplingMode = 0;
end;

end % setup


function SetInputPortSamplingMode(block, port, mode)

block.InputPort(port).SamplingMode = mode;

end % SetInputPortSamplingMode


function Start(block)

%%  Get init state

x0  = block.DialogPrm(1).Data;      %   get init state

%%  Set init state

block.ContStates.Data(1) =  x0(1);  %   gam(0)
block.ContStates.Data(2) =  x0(2);  %   th(0)

block.ContStates.Data(3) =  x0(3);  %   dot_gam(0)
block.ContStates.Data(4) =  x0(4);  %   dot_th(0)

end % Start


function Outputs(block)

%%  Extract state components

gam     = block.ContStates.Data(1);
th      = block.ContStates.Data(2);

dot_gam = block.ContStates.Data(3);
dot_th  = block.ContStates.Data(4);

```

```
%%  Get accelerations

[ddot_gam, ddot_th] = get_acc(block);

%%  Set outputs

block.OutputPort(1).Data = gam;
block.OutputPort(2).Data = th;

block.OutputPort(3).Data = dot_gam;
block.OutputPort(4).Data = dot_th;

block.OutputPort(5).Data = ddot_gam;
block.OutputPort(6).Data = ddot_th;

end % Outputs


function Derivatives(block)

%%  Extract state components

dot_gam = block.ContStates.Data(3);
dot_th  = block.ContStates.Data(4);

%%  Get accelerations

[ddot_gam, ddot_th] = get_acc(block);

%%  Set state derivative

block.Derivatives.Data(1) = dot_gam;
block.Derivatives.Data(2) = dot_th;

block.Derivatives.Data(3) = ddot_gam;
block.Derivatives.Data(4) = ddot_th;

end % Derivatives


function Terminate(block)

end % Terminate


function [ddot_gam, ddot_th] = get_acc(block)

%%  Get inputs and parameter structs

%   parameters
body  = block.DialogPrm(2).Data;    %   body data struct
mot   = block.DialogPrm(3).Data;    %   mot data struct
gbox  = block.DialogPrm(4).Data;    %   gbox data struct
wheel = block.DialogPrm(5).Data;    %   wheel data struct

%   input voltages
ua = block.InputPort(1).Data;       %   armature voltage (right/left motor)

%%  Extract state components

th      = block.ContStates.Data(2);
dot_gam = block.ContStates.Data(3);
dot_th  = block.ContStates.Data(4);
```

```
158
159 %% Extract params
160
161 %   body params
162 l       = body.zb;
163 mb      = body.m;
164 Ibyy    = body.Iyy;
165
166 %   wheel params
167 w       = 2*wheel.yb;
168 r       = wheel.r;
169 mw      = wheel.m;
170 Iwyy    = wheel.Iyy;
171
172 %   (motor) rotor params
173 zbrot   = mot.rot.zb;
174 mrot    = mot.rot.m;
175 Irotyy  = mot.rot.Iyy;
176
177 %   gear ratio
178 n   = gbox.N;
179
180 %   friction params
181 bw  = wheel.B;      %   wheel viscous fric coeff
182 bm  = mot.B;        %   motor viscous fric coeff (motor side)
183 bg  = gbox.B;       %   gbox viscous fric coeff (load side)
184 b   = n^2*bm+bg;    %   motor+gbox viscous fric coeff (load side)
185
186 %   gravity acc
187 g = 9.81;
188
189 %% Get motor torques
190
191 %   back-EMFs
192 ue = mot.Ke * n*(dot_gam-dot_th);
193
194 %   motor torque (single motor)
195 tau = n*mot.Kt * (ua-ue)/mot.R;
196
197 %% Evaluate accelerations of generalised coords
198
199 %   inertia matrix
200 MM = zeros(2,2);
201
202 MM(1,1) = 2*Iwyy + 2*Irotyy*n^2 + (mb + 2*(mrot+mw))*r^2;
203 MM(1,2) = 2*(1-n)*n*Irotyy + r*(l*mb + 2*mrot*zbrot)*cos(th);
204
205 MM(2,1) = MM(1,2);
206 MM(2,2) = Ibyy + 2*(1-n)^2*Irotyy + mb*l^2 + 2*mrot*zbrot^2;
207
208 %   Coriolis + centrifugal terms matrix
209 CC = zeros(2,2);
210
211 CC(1,1) = 0;
212 CC(2,1) = 0;
213
214 CC(2,2) = 0;
215 CC(1,2) = -r*(mb*l + 2*mrot*zbrot)*sin(th)*dot_th;
216
217 %   viscous friction matrix
218 Fv = zeros(2,2);
219
220 Fv(1,1) = 2*(b+bw);
```

```
Fv(1,2) = -2*b;

Fv(2,1) = Fv(1,2);
Fv(2,2) = 2*b;

%   gravity loading
 GG = zeros(2,1);
GG(2) = -g*(mb*l + 2*mrot*zbrot)*sin(th);

%   generalized actuator forces
TT = zeros(2,1);
TT(1) = 2*tau;
TT(2) = -2*tau;

%   get accelerations of generalised coords (q = [gam, th].')
dotq = [dot_gam, dot_th].';
ddotq = MM \ (TT - CC*dotq - Fv*dotq - GG);

ddot_gam = ddotq(1);
ddot_th  = ddotq(2);

end % get_acc
```