

Embedded Code Generation

DEMO MODEL

Inverted Pendulum with Stepper Motor and LQR

Offline Simulation, HIL Testing, and Physical Operation

Last updated in STM32 TSP 1.6.2

www.plexim.com

- Request a PLECS and PLECS Coder trial license
- Get the latest STM32 Target Support Package
- Check the PLECS and STM32 TSP documentation

1 Overview

This STM32G4x demo controls a rotary inverted pendulum driven by a stepper motor. The controller can run in an offline PLECS simulation or as generated C code on an STM32G4x microcontroller. The model uses a linear-quadratic regulator (LQR) to stabilize the pendulum at the inverted equilibrium, while a state machine coordinates startup, calibration, and the transition to LQR regulation.

The model supports three operating configurations:

- Offline simulation (p. 9) of the controller and pendulum plant in PLECS
- Hardware-in-the-loop (HIL) testing (p. 12) with the controller running on an STM32G4x target and the plant model running on a PLECS RT Box real-time simulator
- Physical operation (p. 13) with the STEVAL-EDUKIT01 inverted pendulum kit and the controller running on an STM32G4x target

This document provides the steps required to run each configuration and explains how to configure the STM32G4x peripheral blocks from the PLECS library to acquire the encoder signals and drive the stepper motor. For the underlying control theory, refer to the existing Motor Control and Control Systems Curriculum[1] and the *Edukit Rotary Inverted Pendulum* manual [1].

1.1 Requirements

- PLECS Standalone or PLECS Blockset[2] 5.0.6 or newer
- PLECS Coder[3]
- PLECS STM32 TSP[4] 1.6.2 or newer
- RT Box TSP[5] 4.0.7 or newer

Additionally required for HIL testing

- One STM32G4x NUCLEO board supported by the PLECS STM32 TSP
- One PLECS RT Box[6]
- One RT Box-NUCLEO Interface board[7]

Additionally required for operation with the physical pendulum

- One STM32G4x NUCLEO board supported by the PLECS STM32 TSP
- One X-NUCLEO-IHM14A1[8] stepper motor driver expansion board
- The STEVAL-EDUKIT01[9] evaluation kit providing the pendulum mechanics, stepper motor, quadrature encoder, and 12 V power supply

The STEVAL-EDUKIT01 includes a NUCLEO-F401RE board and an X-NUCLEO-IHM01A1 stepper motor driver expansion board. This demo uses the NUCLEO-G4x and X-NUCLEO-IHM14A1 boards instead.

[1] https://sites.google.com/view/ucla-st-motor-control/home
[2] https://www.plexim.com/products/plecs
[3] https://www.plexim.com/products/plecs/plecs_coder
[4] https://www.plexim.com/download/tsp_stm32
[5] https://www.plexim.com/download/rt_box
[6] https://www.plexim.com/products/rt_box
[7] https://www.plexim.com/products/rt_box/launchpad_interface
[8] https://www.st.com/en/evaluation-tools/x-nucleo-ihm14a1.html
[9] https://www.st.com/en/evaluation-tools/steval-edukit01.html

2 Model

Note

This model contains model initialization commands that are accessible from:
PLECS Standalone: the menu **Simulation > Simulation Parameters... > Initializations**
PLECS Blockset: right click in the **Simulink model window > Model Properties > Callbacks > InitFcn***

For details on the initialization commands, see “Initialization and Operating Configurations” (p. 9).

The top-level schematic of the model contains two separate subsystems representing the controller and plant models, as shown in Fig. 1. Both subsystems are enabled for code generation from the **Subsystem > Execution settings...** context menu. This setting allows PLECS Coder to generate C code for either subsystem.

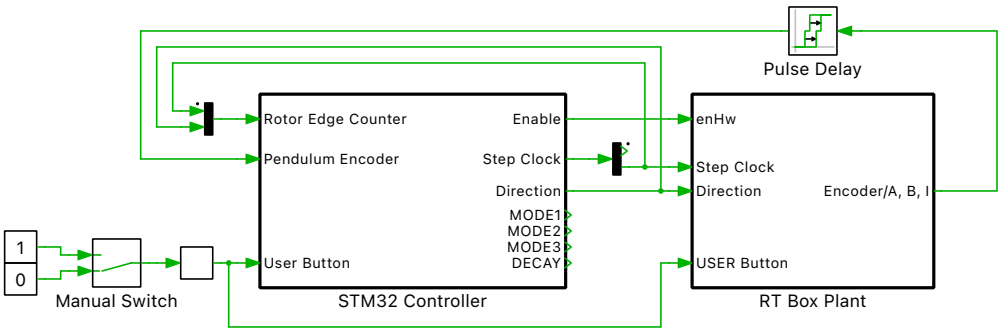


Fig. 1: Top-level schematic of the inverted pendulum demo model

2.1 Plant

The mechanical arrangement of the rotary inverted pendulum is shown in Fig. 2. The L-shaped pendulum has two rotational degrees of freedom: one about the vertical axis and one about the horizontal axis. The stepper motor actuates the pendulum rotation about the vertical rotor axis, represented by the rotor angle ϕ, while its rotation about the horizontal encoder axis, represented by the pendulum angle θ, is passive. Therefore, the inverted pendulum is an underactuated system with an unstable upright equilibrium.

The arm length r and shaft length l shown in Fig. 2 correspond to `r_arm` and `l_shaft` in the initialization commands. For the plant model, the mass of the L-shaped beam is modeled as a point mass at its center of mass. The initialization commands define this point mass as `m_p` and its effective horizontal and vertical distances as `r_cm` and `l_cm`, respectively.

The corresponding PLECS plant model is shown in Fig. 3. It accounts for the nonlinear inertial coupling between the rotor and pendulum motions, gravity, and viscous damping, and provides the signal interfaces required for offline simulation (p. 9) and HIL operation (p. 12).

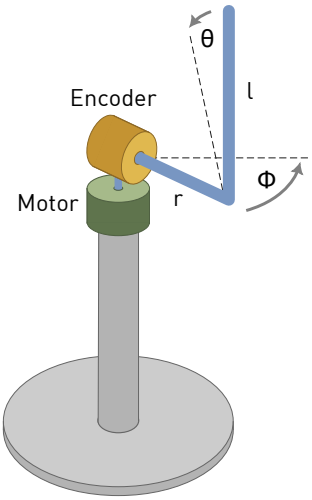


Fig. 2: Mechanical arrangement and angle definitions of the rotary inverted pendulum

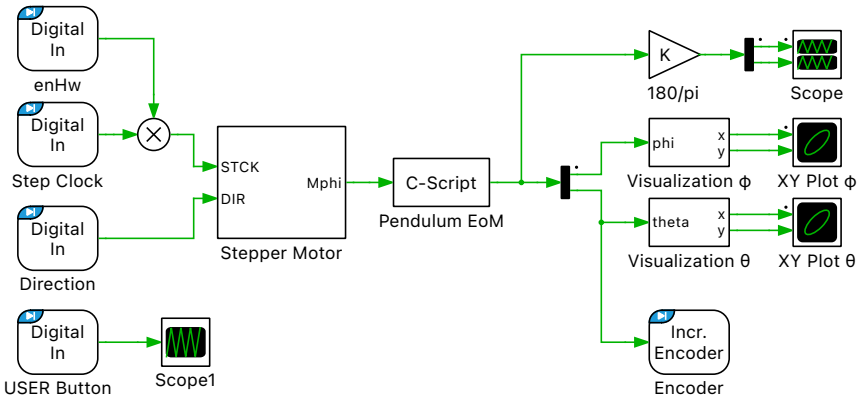


Fig. 3: PLECS plant model of the rotary inverted pendulum

The **Stepper Motor** subsystem models the basic, simplified functionality of the physical stepper motor. In the physical system, each rising edge of the `STCK` signal commands the motor driver to advance the motor by one step in the direction selected by the `DIR` signal. The subsystem represents each step command as a signed torque pulse of fixed duration, with its magnitude determined by `stepper_gain`. The electrical motor dynamics and the switching behavior of the motor driver are not modeled.

The **Pendulum EoM** C-Script block implements the nonlinear **e**quations **o**f **m**otion of the coupled mechanical system using the standard Euler-Lagrange form for rigid-body dynamics. This formulation is commonly used in robotics, multibody mechanics, and control [2]. In compact form, the equations are

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{D}\,\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \begin{bmatrix} M_\phi \\ 0 \end{bmatrix}, \qquad \mathbf{q} = \begin{bmatrix} \phi \\ \theta \end{bmatrix}.$$

In this expression, $\mathbf{M}$ is the configuration-dependent inertia matrix, $\mathbf{c}$ contains the centrifugal and Coriolis terms, $\mathbf{D}$ contains the rotor and pendulum damping coefficients, $\mathbf{g}$ represents gravity, and M_ϕ is the torque generated by the stepper motor model. The subsystem evaluates the nonlinear expressions for $\ddot{\phi}$ and $\ddot{\theta}$ and integrates them to obtain the angular velocities and positions.

The **Encoder** block converts the simulated pendulum angle into quadrature signals. It is configured with `enc_steps/4` $= 600$ line pairs because quadrature decoding counts four state transitions per line pair. This results in an encoder resolution of `enc_steps` $= 2400$ counts per revolution, matching the

encoder used in the STEVAL-EDUKIT01 kit. The A and B signals are assigned to RT Box digital outputs `DO21` and `DO16`, respectively. The block also assigns the index signal to `DO2`, but the STM32 Counter block uses only A and B and the index signal is not used by the controller.

The **USER Button** Digital In block monitors the user-button signal of the STM32G4x board on RT Box channel `DI26` and configures the input with a pull-down characteristic. In HIL operation (p. 12), the pull-down returns the signal to a defined low level when the button is released, ensuring that each press is detected correctly.

2.2 Controller

The STM32 Controller subsystem implements the pendulum control and the interfaces to the stepper motor and encoder. It contains a State Machine block that implements the supervisory control by generating enable and reset signals.

Task Scheduling

The controller executes the two periodic tasks listed in Tab. 1.

Table 1: Controller tasks

Task name	Sample time
Base task	`ts_base` $= 100\,\mu$s
Control task	`ts_ctrl` $= 2\,$ms

To view the task configuration, choose **Coder > Coder Options...**, select **STM32 Controller**, and open the **Scheduling** tab. **Tasking mode** is set to `multi-tasking`, and the table lists both tasks and their sample times. This demo uses a single-core target, so the tasks execute by priority rather than in parallel. The higher-priority `Base task` can interrupt the `Control task`, whose sample time must be an integer multiple of the base sample time.

Fig. 4 shows the elements that configure task execution in the **STM32 Controller** subsystem schematic. The yellow Task Frame assigns the enclosed, less time-critical functions to the `Control task`, while functions outside the frame run in the `Base task`. A Timer block operating at `1/ts_base` provides the event for the Base Task Trigger block, explicitly defining the base-task execution rate.



Fig. 4: Task Frame and Base Task Trigger configuration

Supervisory Control

The user button initiates the supervisory logic shown in Fig. 5. A Monoflop block converts the Digital In signal into a $100\,$ms command pulse for the State Machine block.

The State Machine block coordinates four states:

- `Off` disables the stepper motor and all control functions. It also asserts `rst_phi`, which reloads the initial value of the Rotor Edge Counter and defines the current rotor position as $0°$.
- `Calibration` enables the drive, initiates the calibration motion, and waits until no further pendulum motion is detected. When the pendulum is stationary in the downward position, `rst_theta` is set to 1 to define this position as the zero-angle reference ($\theta = 0$).

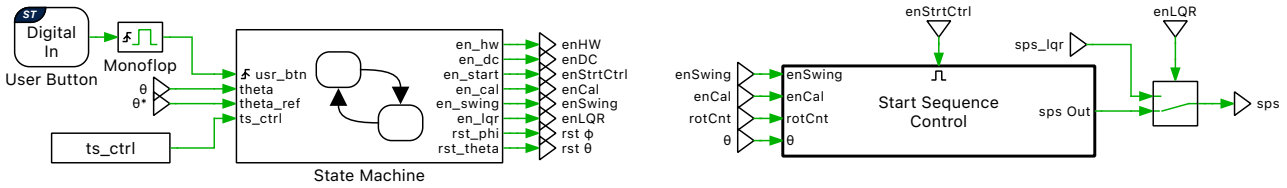


Fig. 5: State machine and start-sequence control

- `SwingUp` enables the swing-up sequence until the pendulum approaches the upright reference.
- `LQRcontrol` enables the LQR controller. In inverted operation, an excessive angle deviation returns the system to `Off`. After $120\,\mathrm{s}$, the state returns to `Calibration`.

The user button starts the sequence from `Off` and stops it from any active state. The `MODE` parameter in the model initialization commands determines the state sequence after `Calibration`. With `MODE = 'Suspended'`, the controller enters `LQRcontrol` directly once the pendulum is stationary. With `MODE = 'Inverted'`, it first enters `SwingUp` and changes to `LQRcontrol` when the pendulum approaches the upright reference. The `enHW` output drives the **Enable** Digital Out block and enables the stepper motor hardware.

During `Calibration`, the **Start Sequence Control** subsystem applies a $10°$ step to the rotor-angle reference ϕ^*, allowing the pendulum to settle in the downward position.

During `SwingUp`, the **Swing Up** subsystem alternates the rotor-angle reference between $\phi^* = \pm 10°$. Within $\pi/8$ of the downward position, the pendulum angle θ determines the kick direction. Beyond this threshold, the previous direction is retained to prevent the angle wrap, as described in "Angle Acquisition and Angular Velocity Estimation", from reversing the reference. The reference is then converted to motor steps and compared with the actual rotor position. A PD controller converts the resulting position error into a step-rate command in steps per second (sps).

LQR Feedback Control

The LQR feedback path is shown in Fig. 6. The reference vector contains the rotor and pendulum angle references and zero angular-velocity references. The manual switches can replace the constant rotor-angle reference with a sine wave and add a step to the pendulum-angle reference for testing. The measured angles and angular velocities are subtracted from this vector.

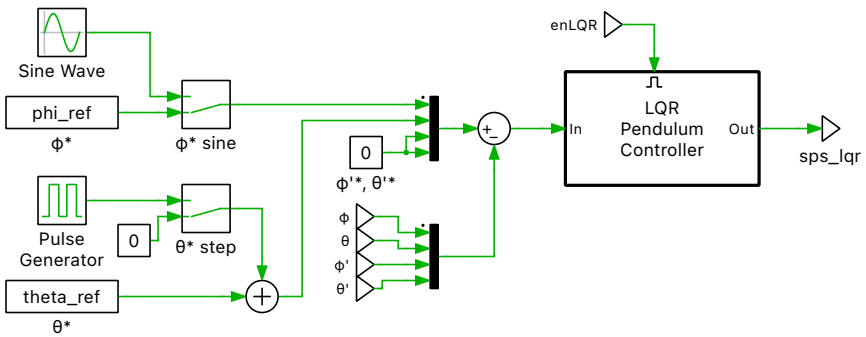


Fig. 6: LQR feedback control and reference signals

The **LQR Pendulum Controller** subsystem multiplies the state error by the gain vector `K_dlqr` calculated in the initialization commands. The **TF** subsystem approximates the first-order response of the rotor angular velocity to the requested control torque, accounting for rotor inertia and damping. Its angular-velocity output is converted to steps per second (sps) and limited to `max_spd` before it is passed to the stepper motor interface.

Angle Acquisition and Angular Velocity Estimation

The angle-acquisition path is shown in Fig. 7. It uses two STM32 timer peripherals: one to decode the pendulum encoder and one to count the commanded motor steps.

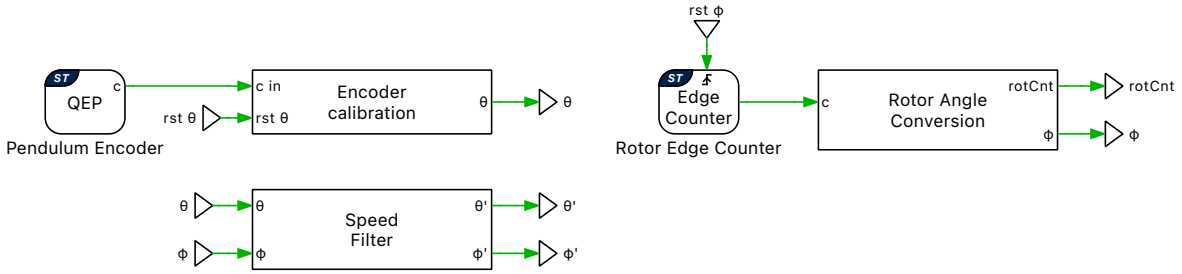


Fig. 7: Angle acquisition, calibration, and angular-velocity estimation

The **Pendulum Encoder** QEP block decodes the quadrature encoder signals with a maximum counter value of `enc_steps-1` $= 2399$, because the encoder provides 2400 counts per revolution and the QEP counter starts at 0. The **Encoder calibration** subsystem wraps the pendulum angle at the equivalent positions $-90°$ and $270°$. This wrap point lies far outside the controlled angular range, so the resulting discontinuity does not require special handling in the controller. The subsystem also stores the zero offset when `rst_theta` is set to 1 and scales the result by `2*pi/enc_steps` to obtain θ.

The **Rotor Edge Counter** block reconstructs the commanded rotor position from the signals generated by the "Stepper Motor Interface". It counts the commanded step-clock pulses and assumes that each pulse rotates the motor through the angle corresponding to one step. As the PWM peripheral generates the step-clock signal asynchronously to the `Base task`, the step-clock edges cannot be counted reliably in software. The step-clock and direction outputs are therefore physically looped back to the Edge Counter inputs. Operating at a much higher clock frequency than the `Base task`, the Edge Counter peripheral captures every step-clock edge sent to the motor driver and maintains an accurate count of the commanded motor steps.

Since the counter is unsigned, it is initialized at `rot_steps`, which is the midpoint of its configured range from 0 to `2*rot_steps`. This provides equal counting range in both directions and avoids immediate underflow for negative movement. The **Rotor Angle Conversion** subsystem subtracts `rot_steps` from the counter value to obtain the signed count `rotCnt` and scales it by `2*pi/rot_steps` to obtain ϕ.

The **Speed Filter** subsystem calculates angle differences at `ts_base` and smooths them over `10*ts_ctrl` for the pendulum and `20*ts_ctrl` for the rotor. Dividing the filtered differences by `ts_base` yields the angular-velocity estimates θ' and ϕ'.

Stepper Motor Interface

A stepper motor advances by one step whenever the motor driver detects a rising edge on its step-clock input. In Fig. 8, the **Step Clock** PWM block generates this step-clock signal, so its output frequency in Hz corresponds to the motor step rate in steps per second (`sps`). The **Direction** Digital Out block drives the direction input of the motor driver, with 1 selecting positive rotor motion and 0 selecting negative rotor motion. The **Speed Handling** subsystem derives the direction value from the sign of `sps_in` and the pulse frequency from its magnitude.

Within the **Speed Handling** subsystem, a Rate Limiter applies the `max_acc` and `max_dec` limits to avoid excessive mechanical stress. For the physical pendulum (`TARGET = 'HW'`), the **Rot Ang Limit** subsystem suppresses pulses that would move the rotor beyond $\pm 150°$ to prevent excessive twisting of the encoder cable.

The **Step Clock** PWM block uses the frequency and duty-cycle commands to generate the step-clock signal. The duty-cycle input acts as a switch for pulse generation and is set to either $50\,\%$ or $0\,\%$. A value of $50\,\%$ produces one rising edge, and therefore one motor step, per PWM period, while $0\,\%$ turns pulse generation off.

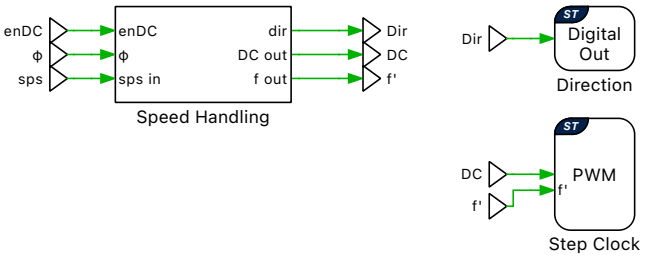


Fig. 8: Stepper motor speed handling and peripheral outputs

With `f' =` 1, the PWM operates at its nominal carrier frequency of `1/ts_ctrl =` $500\,\mathrm{Hz}$. Each PWM period then corresponds to one control cycle and advances the motor by one step, resulting in a motor speed of $500\,\mathrm{sps}$. When `f' >` 1, the PWM frequency increases, allowing multiple step-clock pulses per control cycle and motor speeds above $500\,\mathrm{sps}$.

The block is configured for complementary operation, but only the complementary output is used because `TIM8_CH1N` can be assigned to `PB3`, which is connected to the step-clock input of the stepper motor driver board. The main output `PC6` remains unconnected.

The **Speed to DC and freq** C-Script converts the magnitude of `sps_in` into the `f'` and `DC` commands for the PWM block. The minimum continuously generated step rate is `min_spd = 1/ts_ctrl =` $500\,\mathrm{sps}$, corresponding to `f' =` 1. For speed requests below `min_spd`, the C-Script holds `f'` at 1 and lowers the average step rate through pulse skipping. It commands a duty cycle of $0\,\%$ during cycles in which a pulse is skipped and $50\,\%$ when a pulse is emitted. At or below `min_spd/10`, it continuously commands $0\,\%$ and keeps the motor stationary. For speed requests above `min_spd`, the C-Script increases `f'` in proportion to the requested step rate and commands a duty cycle of $50\,\%$. The comments in the C-Script provide further implementation details.

3 Simulation

This section describes the operating configurations, offline simulation, controller deployment, and HIL testing.

3.1 Initialization and Operating Configurations

The two user selections in the initialization commands (p. 3) are `TARGET` and `MODE`, which are described below.

The `TARGET` selection configures the pendulum encoder interface and the rotor-angle limit as listed in Tab. 2. By default, the model uses `TARGET = 'HIL'`.

Table 2: Target-dependent STM32 Controller configuration

`TARGET`	Pendulum encoder timer	Encoder connection			Rotor-angle limit
`'HIL'`	`TIM4`	Signal	RT Box port	STM32 port	Disabled
		A	D021	PB6	
		B	D016	PB7	
`'HW'`	`TIM1`	Signal wire		STM32 port	Enabled at $\pm 150°$ to protect the encoder cable
		A		PC0	
		B		PC1	

The `MODE` selection defines the pendulum reference and the sequence following calibration as listed in Tab. 3. By default, the model uses `MODE = 'Inverted'`.

Table 3: Operating modes

`MODE`	Pendulum reference	State machine sequence
`'Suspended'`	Downward (`theta_ref = 0`)	`Off` → `Calibration` → `LQRcontrol`
`'Inverted'`	Upright (`theta_ref = pi`)	`Off` → `Calibration` → `SwingUp` → `LQRcontrol`

The initialization commands also execute `LinMod.m` to calculate the continuous-time linearized state-space matrices, check controllability and observability, discretize the model at `ts_ctrl`, and calculate the LQR gains for the selected operating point. The `LinMod.m` file must remain in the same folder as the PLECS model so that the initialization commands can execute it. To inspect the initialization output, open the console using **Window > Console**.

3.2 Offline Simulation

The offline simulation executes the plant and controller together in PLECS without requiring any external hardware. In the initialization commands, select the desired `MODE`, then run the simulation from **Simulation > Start**. After $3\,\mathrm{s}$, the model automatically emulates a user-button press and starts the operating sequence.

Optional: To observe the state sequence step by step, open the **State Machine** block in the **STM32 Controller** subsystem before starting the simulation. Click the configuration icon and select **Animation** in the State Machine Configuration dialog. Then start the simulation and press the **Space** bar on your keyboard to step through the state transitions.

The **Manual Switch** on the top-level schematic emulates additional user-button presses. Each transition from 0 to 1 starts or stops the controller. Return the switch to 0 before issuing another command.

Fig. 9 shows the **Scope** traces from the **STM32 Controller** subsystem for an offline simulation with `MODE = 'Inverted'`. The green traces show the rotor and pendulum angle references, ϕ^* and θ^*, while the red traces show the corresponding actual angles. The black arrows mark the automatic user-button press at $t = 3\,\mathrm{s}$ and two additional presses issued with the **Manual Switch** at $t = 35\,\mathrm{s}$ and $t = 45\,\mathrm{s}$. The labels at the bottom of the rotor-angle plot show the `state sequence` described under Supervisory Control (p. 5).

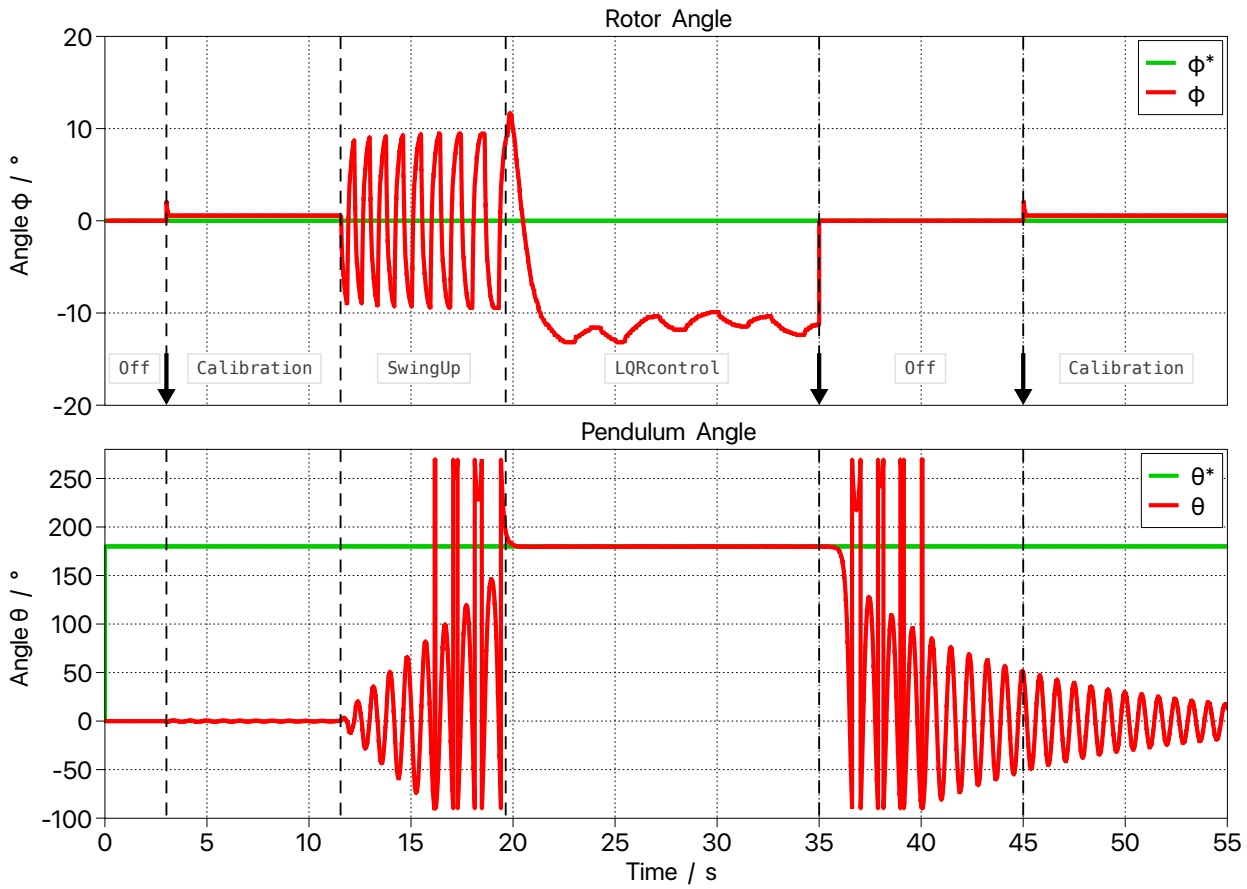


Fig. 9: Rotor and pendulum angles during offline simulation with the `state sequence` at the bottom of the rotor-angle plot and arrows marking user-button presses

The simulation starts in the `Off` state. At $t = 3\,\mathrm{s}$, the first button press starts `Calibration` and generates a brief rotor-angle pulse that disturbs the pendulum. This disturbance induces a small pendulum oscillation, ensuring that it moves toward the downward position, $\theta = 0$. `Calibration` waits until the small pendulum-angle oscillations have decayed and the pendulum reaches standstill at the downward position, which occurs at approximately $t = 12\,\mathrm{s}$. Because `MODE` is set to `Inverted`, the controller then proceeds to `SwingUp`, where the alternating rotor motion increases the pendulum oscillation toward the upright reference $\theta^* = 180°$. As the oscillation grows, the angle wrapping (p. 7) appears as jumps in θ between $-90°$ and $270°$.

Just before $t = 20\,\mathrm{s}$, the pendulum angle enters the $\pm 5\,\%$ range around its reference and triggers `LQR-control`, enabling the LQR controller to take over and stabilize the pendulum at the upright position

$\theta = 180°$. At $t = 35\,\mathrm{s}$, the second button press returns the controller to `Off`, disabling LQR control. With LQR control no longer active, the pendulum falls from the unstable upright position and oscillates freely about the downward position, $\theta = 0°$. Entering `Off` redefines the current rotor position as $\phi = 0°$, visible as the immediate reset of the red rotor-angle trace. At $t = 45\,\mathrm{s}$, the third button press starts `Calibration` again. The pendulum is still moving when the plot ends at $t = 55\,\mathrm{s}$, so the standstill condition has not yet been met and the controller remains in `Calibration`.

3.3 Deploying the STM32 Controller

The required NUCLEO-G4x jumper settings and loopback connections are shown in Fig. 10. Set the `5V_SEL` jumper to `5V_STLK`, fit the `IDD` jumper, and place the `VREF` jumper in the default position.

Connect the direction output `CN9-8` (`PA8`) to `CN8-1` (`PA0`) and the step-clock output `CN9-4` (`PB3`) to `CN8-2` (`PA1`) using jumper wires. These loopback connections route the generated direction and step-clock signals to the Rotor Edge Counter, allowing the controller to reconstruct the commanded rotor position.

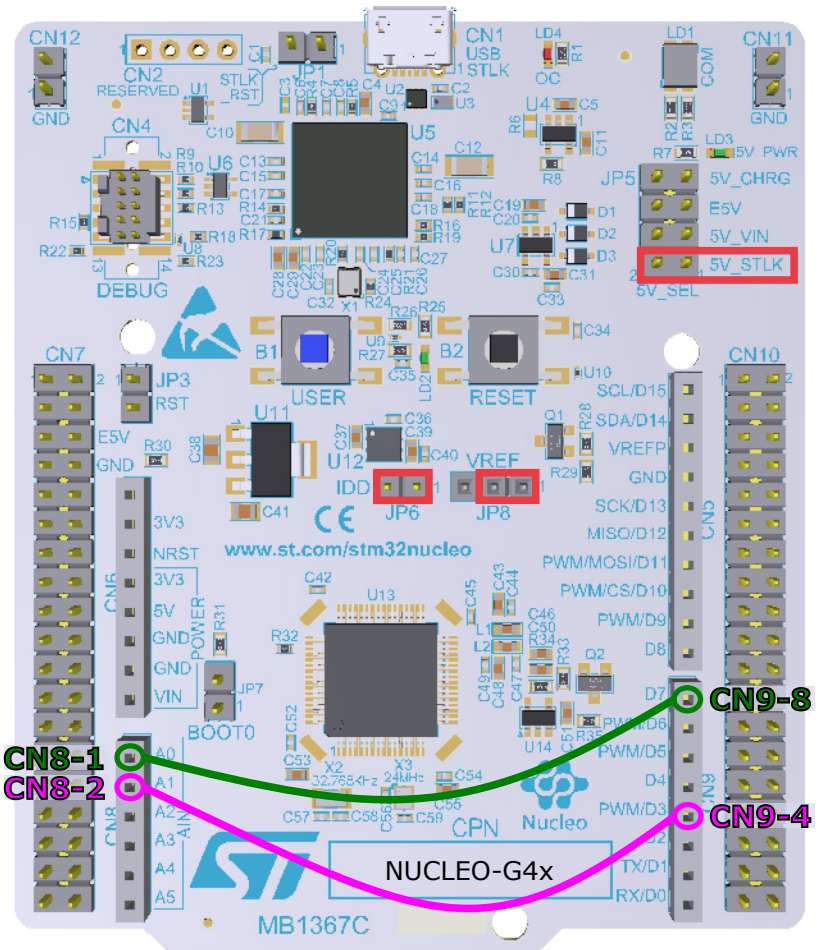


Fig. 10: NUCLEO-G4x jumper settings and loopback connections

To build and upload the controller:

1. Connect the NUCLEO-G4x to the host computer through its ST-LINK USB connector.
2. Choose **Coder > Coder Options...** and select **STM32 Controller** from the list of subsystems.
3. Open the **Target** tab. The `G474RE` chip is already configured. If another supported STM32G4x NUCLEO board is used, select its corresponding chip.

4 Click **Build**. PLECS generates and compiles the controller code and uploads the executable to the NUCLEO board.

To monitor the running controller, open the **External Mode** tab for **STM32 Controller**, select the connected target device, and click **Connect**. Activate autotriggering to display the controller signals continuously in the subsystem scopes.

The **Parameter Inlining** tab in the **Coder Options** dialog lists exceptions to parameter inlining, which remain tunable in External Mode. The parameters used for the reference tests shown in Fig. 6 on p. 6 are already included. If required, additional block parameters can be added by dragging them into the exceptions list before building.

3.4 Hardware-in-the-Loop Test

In the HIL setup, the generated controller runs on the NUCLEO-G4x while the **RT Box Plant** subsystem executes in real time on the PLECS RT Box. Mount the NUCLEO-G4x on the RT Box-NUCLEO Interface Card[10] and connect the interface card to the RT Box front-panel I/O connectors as shown in Fig. 11.

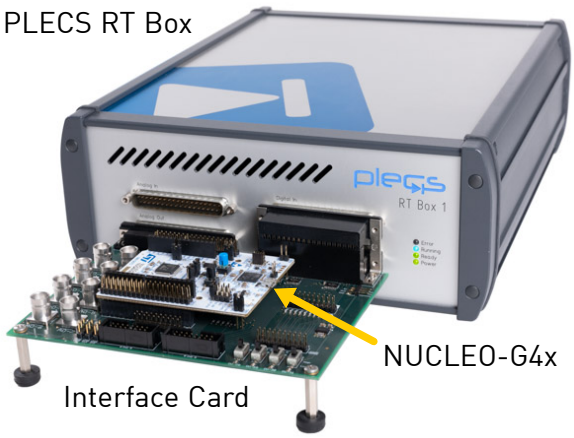


Fig. 11: Hardware setup for HIL testing

The interface card routes the step-clock, direction, and hardware-enable signals from the controller to RT Box digital inputs `DI16`, `DI0`, and `DI4`, respectively. It also connects the user-button signal to `DI26`, which is configured with the required pull-down characteristic. The simulated encoder returns its A and B signals through `DO21` and `DO16` to the controller.

To conduct the HIL test:

1 With the RT Box and NUCLEO powered off, assemble the setup shown in Fig. 11 and add the NUCLEO loopback connections shown in Fig. 10.

2 Connect the RT Box to the host computer via the Ethernet port and power it on.

3 Set `TARGET = 'HIL'` and select `MODE` in the initialization commands.

4 Build and upload the **STM32 Controller** subsystem as described in “Deploying the STM32 Controller”.

5 Choose **Coder > Coder Options...**, select **RT Box Plant** from the list of subsystems, and open the **Target** tab. The model is configured for an `RT Box 2`. If another RT Box is used, select the corresponding target.

6 Click the binoculars button next to **Target device** to scan the network for available RT Boxes, select the connected RT Box, and click **Connect**.

[10] https://www.plexim.com/products/rt_box/launchpad_interface

7 Click **Build** to upload the plant model.

8 Open the **External Mode** tab for **RT Box Plant** and click **Connect**. Activate autotriggering to display the plant signals continuously in the subsystem scopes.

9 Press the blue user button on the NUCLEO board to start the controller.

External Mode for the **STM32 Controller** can also be activated as described in "Deploying the STM32 Controller" (p. 11). With External Mode activated, the switches and reference sources shown in Fig. 6 on p. 6 can be used to test the controller response during operation.

Press the user button again to stop the controller.

4 Physical Pendulum Test

In the physical setup shown in Fig. 12, the generated controller runs on the STM32 target and operates the real pendulum.

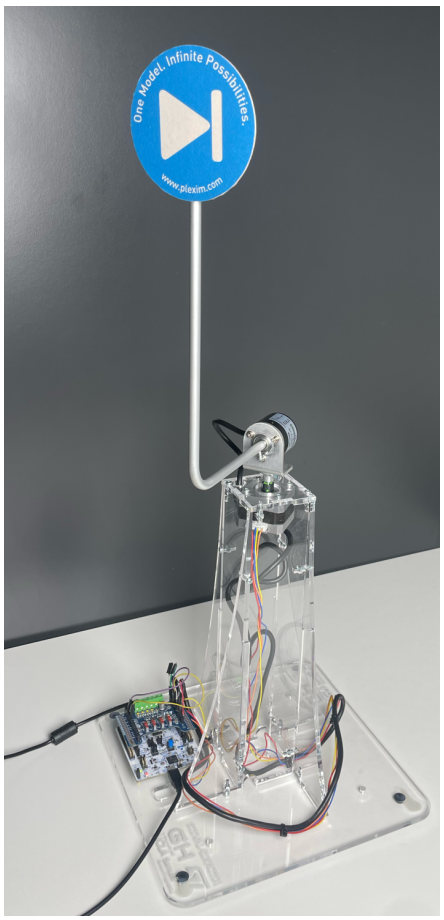


Fig. 12: Rotary inverted pendulum setup

4.1 Connections and Settings

The connections for operation with the physical pendulum are shown in Fig. 13. Use the same NUCLEO jumper settings and loopback connections shown in Fig. 10 on p. 11. With the X-NUCLEO-IHM14A1 expansion board stacked on the NUCLEO, make the loopback connections on its stack-through headers: connect `CN9-8` to `CN8-1` and `CN9-4` to `CN8-2`.

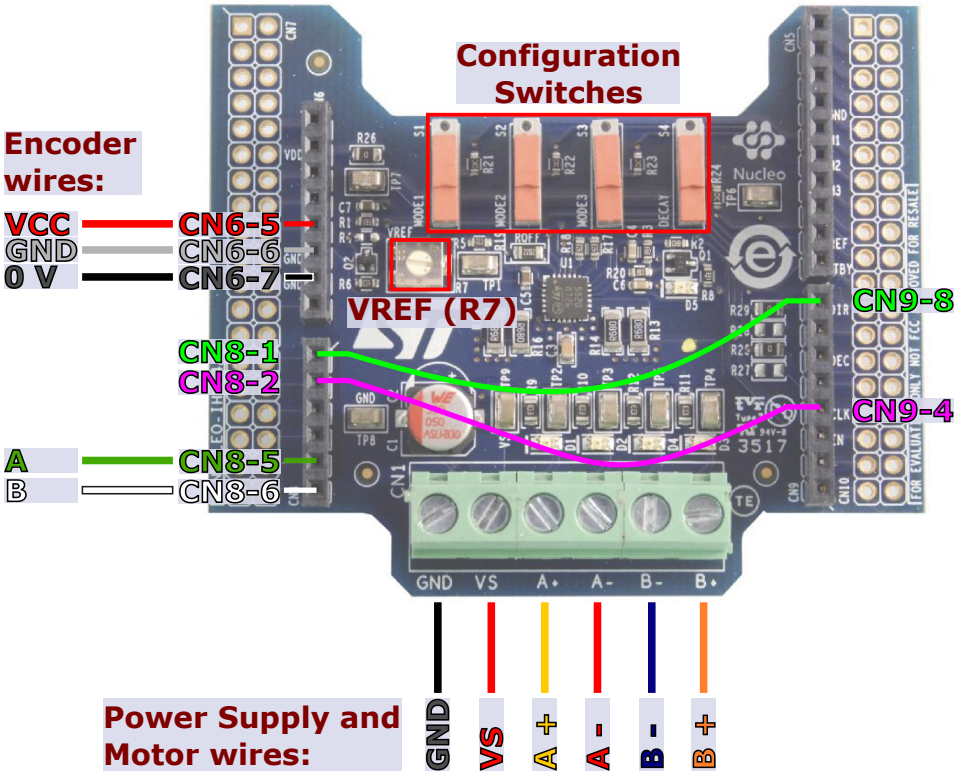


Fig. 13: NUCLEO-G4x and stepper motor driver connections for the physical pendulum

Connect the encoder to the expansion board as listed in Tab. 4.

Table 4: Encoder connections

Encoder connection	Expansion board connector
VCC	CN6-5
Shield	CN6-6
0V	CN6-7
A	CN8-5
B	CN8-6

Connect the 12 V power supply to the screw terminal as listed in Tab. 5.

Table 5: Power supply connections

Power supply connection	Expansion board connector
GND	CN1-6
VS	CN1-5

Connect the stepper motor phases to the remaining screw terminals as listed in Tab. 6.

Table 6: Stepper motor connections

Motor connection	Expansion board connector
A +	CN1 - 4
A -	CN1 - 3
B -	CN1 - 2
B +	CN1 - 1

Keep the four configuration switches on the expansion board in the OFF position, as shown in Fig. 13. The corresponding configuration signals are driven by the controller's Digital Out blocks shown in Fig. 14. The model sets MODE1 (PA5) to 0, MODE2 (PA6) to 0, and MODE3 (PA7) to 1. This selects 1/16-step operation, so each step-clock pulse advances the $1.8°$ motor by a $0.1125°$ microstep. DECAY (PB4) is set to 0. This selects the current-decay mode, which determines how the inductive motor current recirculates and decays during the PWM off-time.

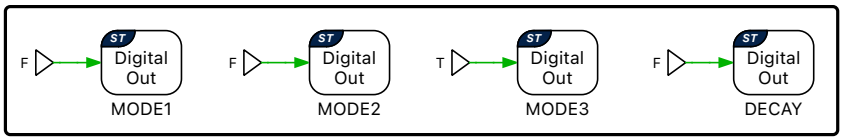


Fig. 14: STSPIN820 configuration for step size and current-decay mode

The VREF potentiometer R7 adjusts the regulated motor current and therefore the available motor torque. Set it approximately to the middle position for stable pendulum operation. The expansion board supplies at most $1.5\,\mathrm{A}$, below the motor rating of $2\,\mathrm{A}$, so the maximum-current position does not exceed the motor rating.

4.2 Operation

To operate the physical pendulum:

1. With the NUCLEO and $12\,\mathrm{V}$ power supply switched off, assemble the setup shown in Fig. 13 and verify the NUCLEO settings and loopback connections shown in Fig. 10 on p. 11.
2. Set TARGET = 'HW' and select MODE in the initialization commands.
3. Connect the NUCLEO-G4x to the host computer through its ST-LINK USB connector, then build and upload the **STM32 Controller** subsystem as described in "Deploying the STM32 Controller" (p. 11).
4. Position the rotor by hand so that it can rotate approximately $180°$ in either direction. Ensure that the encoder cable is untwisted and does not restrict this motion.
5. Switch on the $12\,\mathrm{V}$ power supply.
6. Press the blue user button on the NUCLEO board to start the controller.

External Mode for the **STM32 Controller** can be activated as described in "Deploying the STM32 Controller" (p. 11). With External Mode activated, the switches and reference sources shown in Fig. 6 on p. 6 can be used to test the controller response during operation.

The hardware response to one such test is shown in Fig. 15, with the offline result included for comparison. For this test, the θ^* step switch shown in Fig. 6 on p. 6 was toggled from the zero input to the **Pulse Generator**. This makes the reference jump by $\pm 1.8°$ around the upright position at $180°$.

Each reference change causes the pendulum angle to move and overshoot before returning close to $180°$, while the rotor settles at a different angular position. This behavior reflects that the implemented LQR

is a regulator rather than a dedicated reference-tracking controller. The changed reference does not make the requested tilted pendulum position a new equilibrium. Instead, the controller returns the pendulum toward the upright position and settles at a different closed-loop equilibrium in which the rotor-angle offset balances the remaining pendulum-angle error in the feedback law.

The offline result closely follows the hardware response throughout the test.

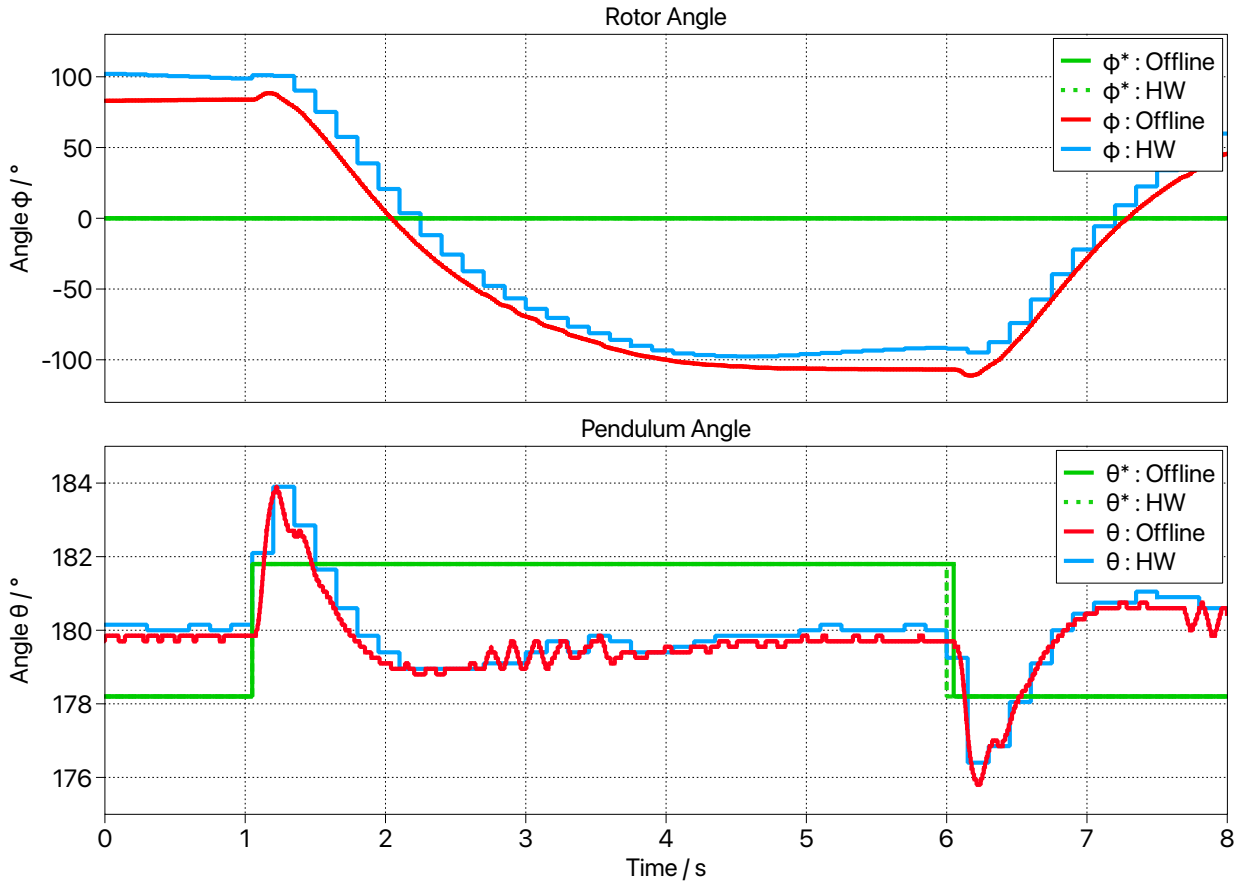


Fig. 15: Hardware and offline responses to pendulum-angle reference disturbances

Press the user button again to stop the controller.

5 Conclusion

This demo demonstrates how PLECS Coder can be used to implement a complete control application for a rotary inverted pendulum on an STM32 microcontroller. The generated code executes the control sequence, interfaces with the required STM32 peripherals, and successfully operates the physical pendulum.

PLECS also enables the complex nonlinear mechanical plant to be simulated offline and in real time on an RT Box. Offline simulation supports rapid controller development without hardware, while HIL testing verifies the generated controller code and its interfaces under realistic and repeatable conditions. The HIL setup also enables tests that are not feasible with the physical setup. For example, HIL enables controller testing with different pendulum dimensions or with rotor motion beyond the encoder-cable limit by assuming wireless encoder signal transmission. Together, the three operating modes provide a consistent workflow from controller development to controller testing and operation of a real hardware system.

6 Bibliography

[1] W. Kaiser, *Edukit Rotary Inverted Pendulum: An Open and Configurable System With Digital Motor Controller Technology*, rev. 2.0, Jan. 12, 2021. URL: https://www.st.com/content/dam/AME/2019/Educational%20Curriculums/motor-control/Introduction_to_Integrated_Rotary_Inverted_Pendulum_v3.pdf.

[2] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*, Springer, 2009.

Revision History:

STM32 TSP 1.6.2 First release

How to Contact Plexim:

+41 44 533 51 00	Phone
+41 44 533 51 01	Fax
Plexim GmbH Technoparkstrasse 1 8005 Zurich Switzerland	Mail
`info@plexim.com`	Email
`https://www.plexim.com`	Web

Embedded Code Generation Demo Model

© 2002–2026 by Plexim GmbH

The software PLECS described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from Plexim GmbH.

PLECS is a registered trademark of Plexim GmbH. MATLAB, Simulink and Simulink Coder are registered trademarks of The MathWorks, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders.