Laboratory guide 2: Introduction to the balancing robot experimental setup

Riccardo Antonello* Francesco Ticozzi*

May 19, 2025

1 Activity goal

The purpose of this laboratory activity is to become familiar with the hardware of the balancing robot (also referred to as a "two-wheeled inverted pendulum robot" or "Segway-like robot") available in the lab, as well as the associated control system development process. This process is based on the same *rapid control prototyping methodology* previously introduced for the Quanser SRV-02 servomotor. By the end of this activity, you should be able to implement and configure a Simulink model that runs on the embedded system installed on the balancing robot.

2 Hardware description

A balancing robot is a robotic vehicle with two independent coaxial driving wheels and a chassis that can swing around their rotation axis. The robot can freely move on a surface, provided that its body is maintained in vertical balance on the two wheels. Since this configuration is naturally unstable, an automatic control system (balance controller) is required to keep the balance. The design of such control system will be the subject of the next laboratory activities.

The structure of the balancing robot available in laboratory is shown in Fig. 1. The robot has two coaxial wheels (right wheel ① and left wheel ②) driven by two DC gearmotors (right motor ③ and left motor ④). Each gearmotor is equipped with a gearbox (⑤) and an incremental magnetic encoder (⑥). The motors are driven by two H–bridge pulse–width modulated (PWM) voltage drivers (⑦), that are controlled by a microcontroller unit (MCU) installed on board of the robot (⑧). The control of the two motors is performed by processing the information provided by two extra boards connected to the MCU: a board that counts the encoder pulses (⑨), and a board equipped with a motion processing unit (MPU), namely a device that combines a three–axis accelerometer and a three–axis gyroscope into a single integrated chip (⑩). The former sensor measures the linear accelerations (including the gravity) of the robot body, while the latter measures its angular velocities. Both measurements are referred to the sensors intrinsic axes. The encoder pulses are used to determine the longitudinal motion and the heading–angle of the robot (assuming that the wheel motion satisfies the "pure rolling" and "no side–slip" conditions); instead, the acceleration and velocity measurements provided by the MCU can be "fused" together (by resorting to a suitable "sensor fusion" algorithm) to estimate the tilt angle of the robot body with respect to the vertical upward position. The control program is developed by the user on a host computer, and transferred to the MCU via a USB connection (⑪). Once the control program is uploaded and stored on the flash memory of the MCU, the robot becomes fully autonomous, and the USB connection can be

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

removed. During its normal operation, the robot can communicate with the host computer through a wireless connection, established by using the onboard Bluetooth module (⑫). The electrical power required by the motors and all the electronic boards is supplied by a lithium–polymer (LiPo) battery pack (⑬). A switch is present to turn the robot on or off (⑭). In addition to that, two general purpose pushbuttons are also present (⑮), whose function can be defined by the user. The details of the hardware components of the robot are provided in Tab. 1. The nominal values of the parameters reported in table are deduced from manufacturers' data–sheets.

3 Control system development procedure

All the activities concerning the development of the control system for the balancing robot (i.e. design, simulation, implementation and experimental validation) are carried out within the MATLAB/Simulink environment, running on a host computer. The control system is implemented in the form of a Simulink model, which is then configured to run on the robot MCU. A custom toolbox, named *Balancing Robot Toolbox*, provides the blockset required to interface with the robot hardware. In order to run on the embedded target, the Simulink model has to be converted into a binary (i.e. executable) file for the specific MCU installed on the balancing robot. The whole conversion process (*deployment to the embedded target*) is automatically managed by MATLAB, and is transparent to the user. The process is articulated as follows. In the first stage, the Simulink model is automatically converted by the *Simulink/Embedded Coder* into an equivalent C–language code for the embedded platform installed on the balancing robot. Then, in the second stage, the generated source code is compiled with a cross–compiler for the specific embedded target, and the resulting binary file is uploaded into the embedded target MCU.

3.1 Model configuration settings for deployment on the embedded target

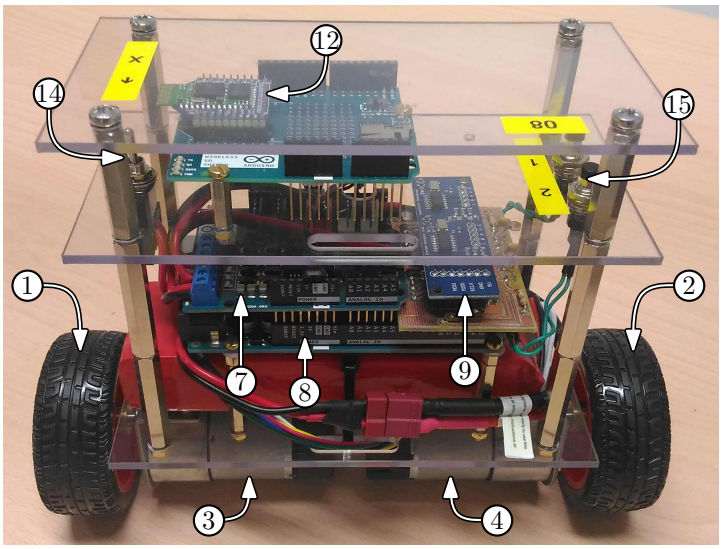
The steps required to configure a Simulink model to run in real–time on the balancing robot MCU are listed below:

1) Prepare a working directory/folder (e.g. `Lab[...]` in your Desktop or Documents folders, with `[...]` denoting an identification label for the lab activity (e.g. `Lab01` for activity 1). Change the current working directory to your new working directory, by using the MATLAB commands `pwd` and `cd` or the Current Directory field in the toolbar.

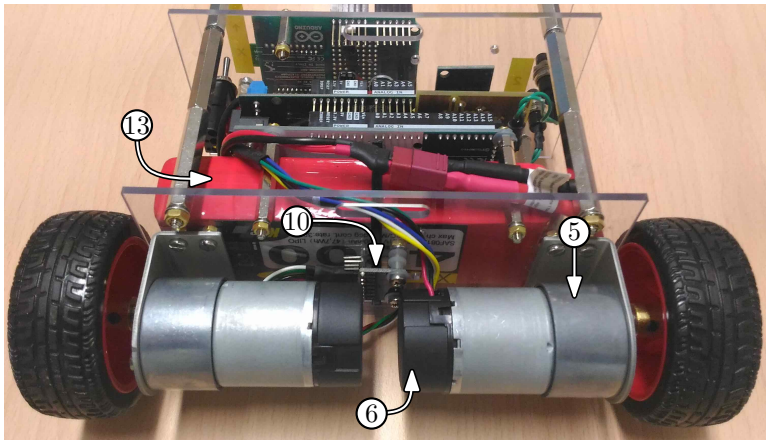
2) Open a new Simulink model, either by selecting **New** → **Simulink Model** from the MATLAB window toolbar, or by invoking `slLibraryBrowser` from the command window and then using the *New Blank Model* button in the toolbar of the *Simulink Library Browser* window.

3) Save the Simulink model in the working directory, by using the **Simulation** → **Save** button in the toolbar of the Simulink model window. Note that the Simulink Coder generates some auxiliary files in the current working directory during the code generation process. Therefore, ensure to save the Simulink model file in your current working directory, to prevent the generation of these files in other undesired locations.

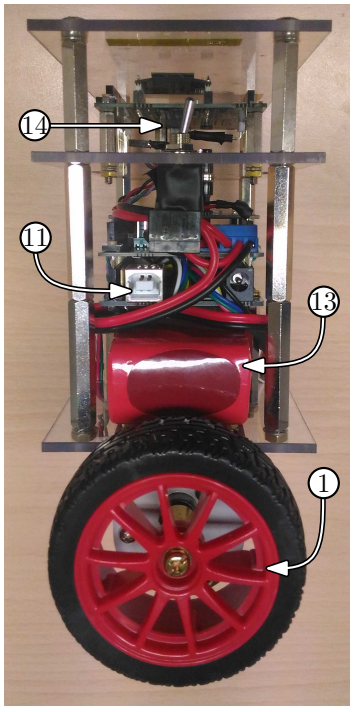
4) To specify the embedded target on which the Simulink model will run, open the *Model Configuration Parameters* window by selecting **Modeling** → **Model Settings** in the toolbar of the Simulink model window.



(a) Front view.



(b) Bottom view.



(c) Side view.

Figure 1: Balancing robot details.

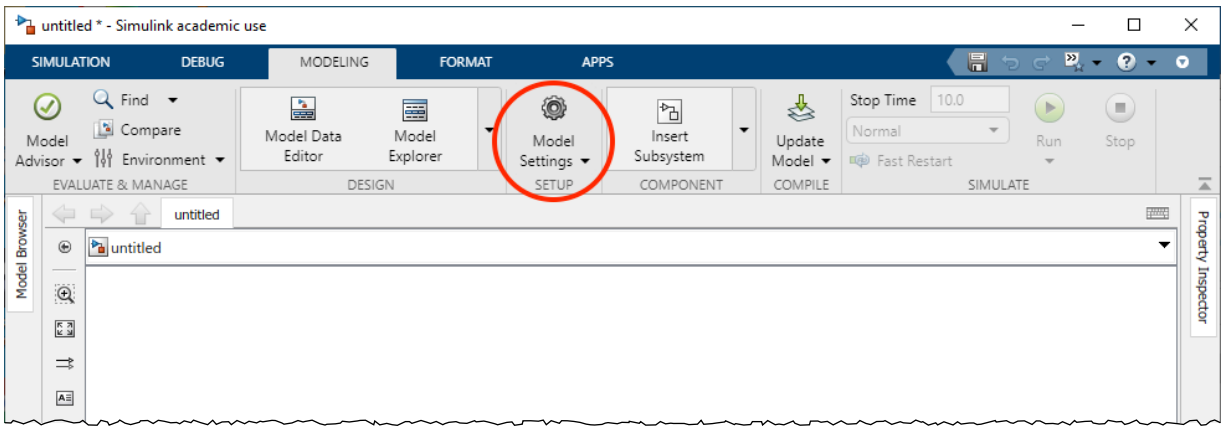
DC gearmotors	
Motor type	brushed DC motor
Motor id	Pololu 30:1 metal gearmotor 37D× 68L with 64 CPR encoder
Nominal voltage	$12\,\mathrm{V}$
Armature resistance	$2.4\,\Omega$
Armature inductance	n.a.
Electric (BEMF) constant	$10.3 \times 10^{-3}\,\mathrm{V\,s/rad}$
Torque constant	$5.2 \times 10^{-3}\,\mathrm{Nm/A}$
Gearbox ratio	30
Size, Weight	$37\,\mathrm{mm}$ (diameter) $\times$ $68\,\mathrm{mm}$ (length), $215\,\mathrm{g}$

Sensors	
Sensor type	magnetic incremental encoder
Sensed quantity	motor shaft angular position
Pulses–per–rotation (ppr)	64 (motor side)[(1)]; 1920 (wheel side)
(1) assume to use a quadrature encoding mode for counting the encoder pulses.	
Sensor type	motion processing unit (MPU) – 3–axis accelerometer + gyroscope
Sensed quantity	robot body linear accelerations and angular velocities
Sensor id	GY–521 module based on Invensense MPU–6050
Accelerometer full–scale	$16\,\mathrm{g}$ [(2)]
Accelerometer resolution	16 bits
Accelerometer bandwidth	$94\,\mathrm{Hz}$
Accelerometer output noise	$4\,\mathrm{mg}$
Gyroscope full–scale	$250\,\mathrm{deg/s}$
Gyroscope resolution	16 bits
Gyroscope bandwidth	$98\,\mathrm{Hz}$
Gyroscope output noise	$0.05\,\mathrm{deg/s}$
(2) $1\,\mathrm{g} = 9.81\,\mathrm{m/s^2}$	

Electronic boards	
MCU board	Arduino Mega 2560
MCU id	8–bit Atmel ATmega2560
Clock speed	$16\,\mathrm{MHz}$
Flash memory, SRAM, EEPROM	$256\,\mathrm{kB}$ ($8\,\mathrm{kB}$ used by bootloader), $8\,\mathrm{kB}$, $4\,\mathrm{kB}$
Digital I/O pins, Analog input pins	54 (of which 15 provide PWM output), 16
Operating voltage	$5\,\mathrm{V}$
Motors driver board	Arduino motor shield
Motors driver type	Dual H–bridge voltage driver (L298)
Encoder interface board	Superdroid Dual Quadrature Encoder Buffer
Encoder counters id	LS7366R quadrature counter
Wireless board	Arduino Wireless SD shield
Wireless module	Bluetooth HC–05 (XBee socket compatible)
Wireless standard	Bluetooth V2.0 + EDR $3\,\mathrm{Mbps}$ modulation

Power supply	
Battery type	Lithium–Polymer
Nominal voltage	$11.1\,\mathrm{V}$ (3S)
Capacity	$4300\,\mathrm{mAh}$
Discharge current (continuos / peak), charge current	30 C / 60 C, 2 C
Size, weight	$26\,\mathrm{mm} \times 44\,\mathrm{mm} \times 136\,\mathrm{mm}$, $320\,\mathrm{g}$

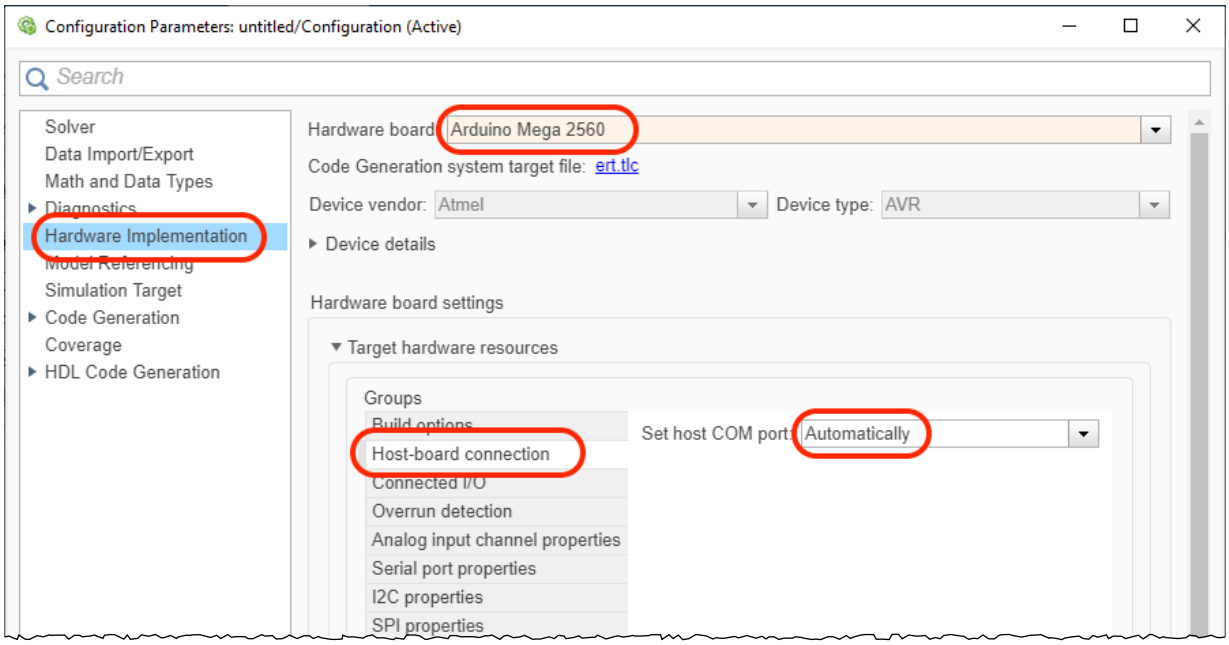
Table 1: Hardware components details.



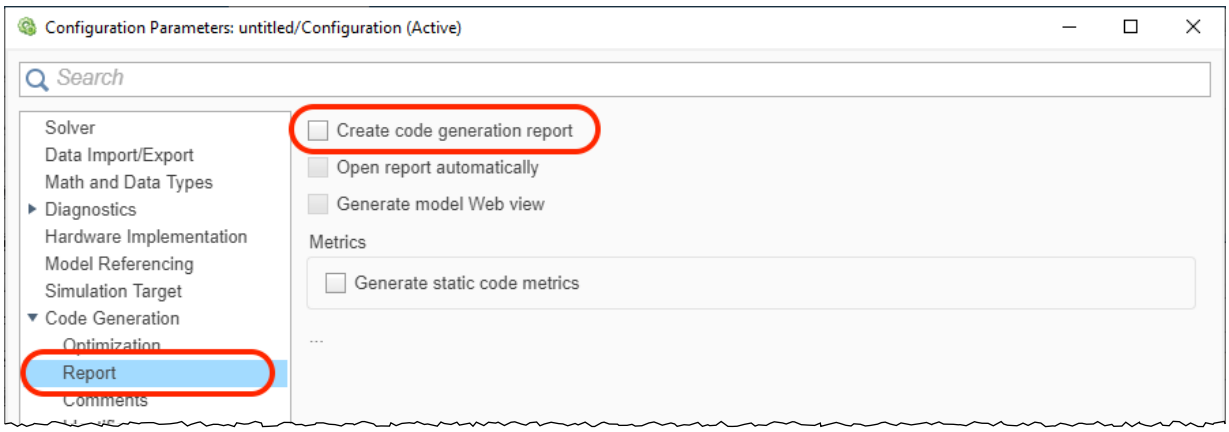
5) Click **Hardware Implementation** in the *Select* list on the left side of the *Model Configuration Parameters* window.

In the **Hardware board** drop–down menu of the Hardware Implementation pane, select `Arduino Mega 2560`, which identifies the embedded system on which the Simulink model will be executed.

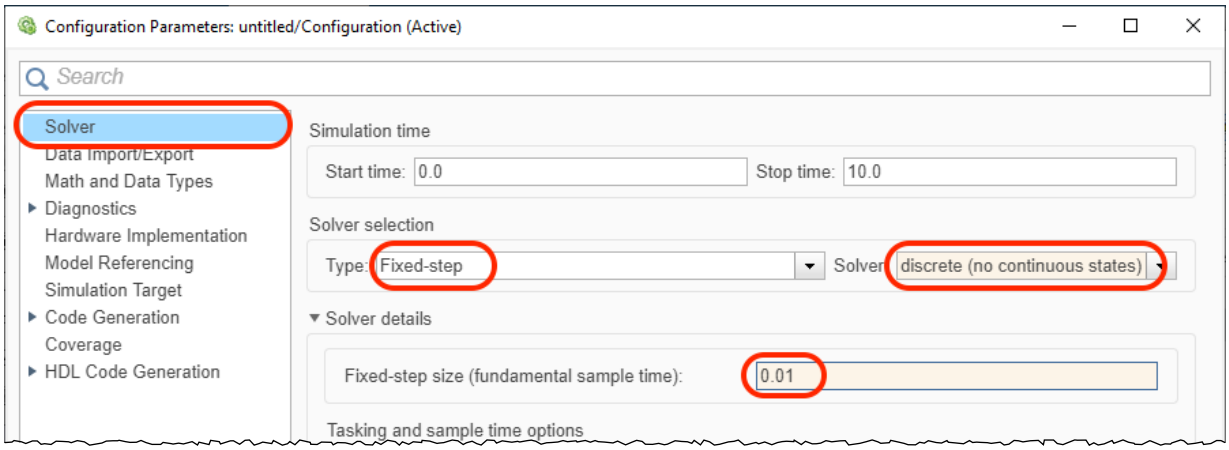
In the *Hardware board settings* section, select **Host–board connection** in the **Target Hardware Resources** group list (click on the triangle ▸ to expand the list). Then, verify that **Set host COM port** option is set to `Automatically`: this forces the host computer to automatically detect the (emulated) serial port used to communicate with the embedded target board (once connected to the host computer via the USB cable).



6) Click **Code Generation** → **Report** in the *Select* list, and then uncheck the option **Create code generation report**: this informs the Simulink Coder to skip the creation of the "code generation report" at the end of the code generation process.



7) Click **Solver** in the *Select* list; in the **Solver options** section, set **Type** to `Fixed-Step` and **Solver** to `discrete (no continuous states)`. In the additional options, set **Fixed-step size (fundamental sample time)** to `0.01`, which is the value of the sample time used to execute the Simulink model on the MCU.



8) Use the **Simulation** → **Save** button in the toolbar of the Simulink model window to save the Simulink model, together with all the above settings required to enable the real–time execution on the embedded target.

3.2 Interfacing with the balancing robot hardware

A specific toolbox, namely the *Balancing Robot Toolbox (BRT)*, has been developed to ease the interfacing of a Simulink model with the balancing robot hardware. The BRT blockset is located under **Balancing Robot Toolbox** in the **Simulink Library Browser** window (see Fig. 2) of the MATLAB installation available in laboratory. A brief description of each block in the BRT blockset is reported below:

- **Outputs** → **Motors**: the block allows to set the duty–cycle command for the H–bridge PWM voltage drivers of the two motors (left/right). The PWM command consists of a byte, plus a sign flag. Hence, the two block inputs are integer numbers in the range $[-255, 255]$. A duty–cycle command equal to 255 corresponds to apply the maximum voltage to the motor armature, which is equal to the battery nominal voltage ($11.1\,\mathrm{V}$).

- **Inputs** → **Encoders**: the block allows to read the pulse count of the two encoders. The encoders are directly connected to the rotor shaft, and hence they provide a measure of the angular position of the motor shaft *at the motor side*. The pulses–per–rotation of the two encoders is equal to 64: with a gearbox ratio of 30, this corresponds to $64 \times 30 = 1920$ pulses–per–rotation at the wheel side (i.e. at the gearbox output shaft).

- **Inputs** → **MPU**: the block allows to retrieve the inertial measurements provided by the MPU, namely the linear accelerations measured by the 3–axis accelerometer, and the angular velocities measured by the 3–axis gyroscope. The measurements are referred to the MPU reference frame (see the handout of laboratory activity 4 for the details about the location and orientation of such frame). The acceleration outputs are in $[\mathrm{g}]$ units ($1\,\mathrm{g} = 9.81\,\mathrm{m/s^2}$), while the angular velocity outputs are in $[\mathrm{deg/s}]$ units.

- **Inputs** → **Pushbuttons**: the block allows to determine the status of the two pushbuttons. The two outputs are normally equal to 0, and are set to 1 while the corresponding pushbutton is being pushed.

- **Communication** → **Serial Packet Send**: the block allows to send a data packet to the host computer, using either the wired (USB) or wireless (Bluetooth) communication link.

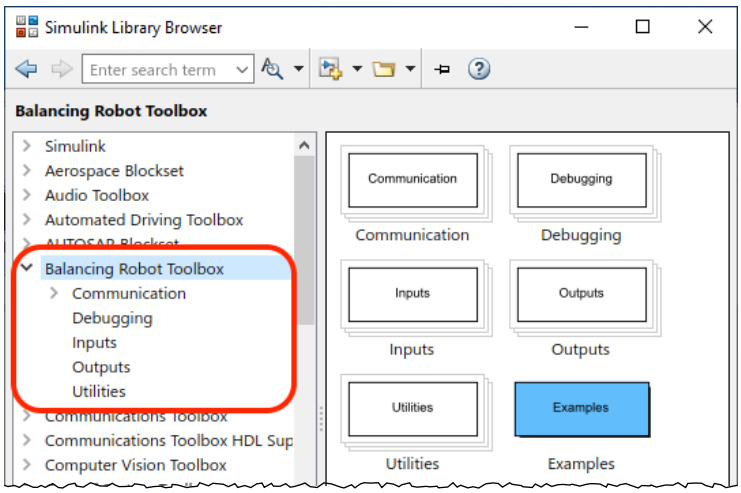


Figure 2: Simulink Library Browser – Balancing Robot Toolbox blockset.

The packet contains the (numeric) value of one or more signals of user–selectable dimension and data-type. The packet structure is defined by a *cell array*; each element of the cell array specifies the size and data–type of a signal stored in the packet. The signals are provided as inputs to the block; an input port is created for each element of the cell array. The input ports numbers correspond to the indexes of the cell array elements.

The packet contains plain binary data; it can be optionally encoded with a special scheme, called *Consistent Overhead Byte Stuffing* (COBS), that allows to reserve a byte value, typically zero, to serve as packet delimiter (terminator). When zero is used as a delimiter, the algorithm replaces each zero data byte in the payload with a non–zero value, so that no misinterpretations occur between a valid data value and the symbol reserved for packet delimitation.

As the name already suggests, the COBS encoding scheme has a constant encoding overhead, equals to 1 byte, e.g. the encoded packet is 1 byte longer than the original packet.

- **Communication → Serial Packet Receive**: the block allows to receive a data packet from the host computer, using either the wired (USB) or wireless (Bluetooth) communication link.

  It is complementary to the *Serial Packet Send* block.

- **Communication → Data coding (COBS)**: group of blocks (left double–click to access the block subset) for encoding/decoding a packet with the COBS encoding scheme.

  Within the Simulink model, the packet is represented by a vector–valued signal of `uint8` elements. The number od elements of the vector–valued signal is equal to the packet byte size.

- **Communication → Data packing**: group of blocks (left double–click to access the block subset) for packing/unpacking one or more signals into/from a single packet, and for appending/removing the null terminator to a packet.

- **Utilities → Flip–Flop Toggle**: the block toggles its output between the logic levels 0 and 1 at each rising edge of the enable input. It can be used to transform the pushbuttons into toggle buttons.

- **Utilities → Monostable**: the block generates a unit pulse of specified duration once triggered by a rising edge of the enable input.

- **Utilities → Up–Counter (modulo N)**: the block implements a modulo–N up–counter. The count is increased by one at each rising edge of the enable input, and is reset once it reaches the specified counter modulo.

- **Utilities → Up–Counter (with upper bound)**: the block implements a up–counter with upper–bound. The count is increased by one at each rising edge of the enable input, until the upper–bound value is reached.

- **Debugging → Execution Pause**: inserts a pause of specified microseconds in every iteration (integration step) of the Simulink model.

- **Debugging → Execution Time**: measures the execution time of one iteration (integration step) of the Simulink model.

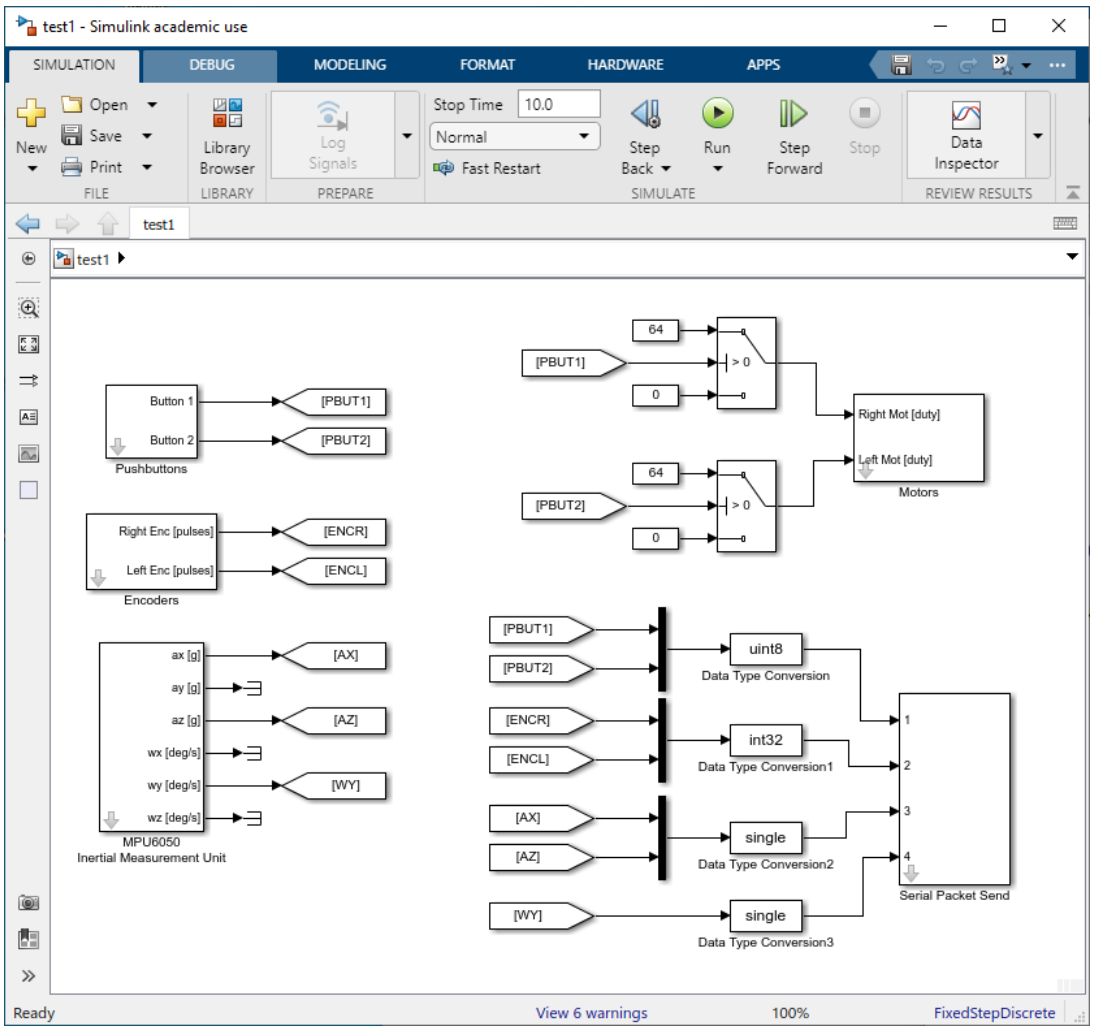


Figure 3: Simulink model used for testing the balancing robot hardware.

In Fig. 3 is shown a simple Simulink model that uses the BRT blockset to test the balancing robot hardware. It works by continuously reading the status of the two pushbuttons, the pulse count of the two encoders, and the accelerations along the x and z axes of the MPU frame. These data are sent to the host computer by using the Bluetooth connection. Moreover, each pushbutton enables, while pressed, the rotation of a motor (pushbutton 1 for the right motor, and 2 for the left motor).

For a correct data logging on the host computer (with the available data logging routine, as described in Sec. 3.4), the Serial Packet Send block has to be configured as shown in Fig. 4. In particular:

- in the *Communication link* section, the option **Wait for Tx Start/Stop commands from Host (data logger)** has to be checked. The option enables the transmission of a new packet every sample time (i.e. continuous data stream) only when the embedded target receives the **Tx Start Command** byte from the host computer. The start command is automatically sent by the data logging routine when the host is ready to receive the data. This prevents the embedded target to saturate the input buffer of the host computer with data sent before the beginning of the data logging process.

  The data transmission is stopped when the embedded target receives a **Tx Stop Command** from the host. The command is automatically sent by the data logging routine before its termination. This prevents the embedded target to continue to send data after the completion of the data

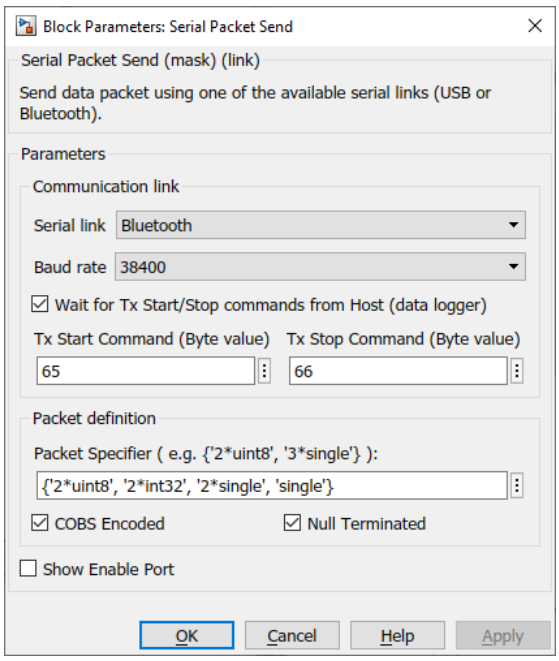


Figure 4: Serial Packet Send: Block Parameters settings to enable data logging (of a continuous data stream) on the host computer.

logging process.

- in the *Packet definition* section, the packet structure has to be specified in the **Packet Specifier** field. The packet specifier is a cell array, whose elements specify the size and data type of each signal to be stored in the packet. The elements are strings with the format:

  $$\text{`}size * data\text{-}type\text{'}$$

  where *size* is a positive integer equal to the size of the signal, and *data-type* is a data-type identifier. Valid data-type identifiers are `uint8`, `uint16`, `uint32` (8-bit, 16-bit, and 32-bit unsigned integers), `int8`, `int16`, `int32` (8-bit, 16-bit, and 32-bit signed integers), `single` (single precision floating point number), `double` (double precision floating point number), `boolean` (boolean number). The size specifier can be omitted in case of scalar signals (i.e. when the size is equal to 1).

- in the *Packet definition* section, both the **COBS Encoded** and **Null Terminated** options have to be checked.

  The first option specifies that the packet has to be encoded with the COBS encoding scheme. The second option instead specifies to add a null terminator at the end of the packet, so that the data-logger can correctly synchronize with the continuous data stream generated by the embedded target.

  The resulting packet size is equal to the payload size (determined by the size and data types of the signals stored in the packet), plus two bytes (one due to the COBS encoding overhead, and the other required for storing the null terminator).

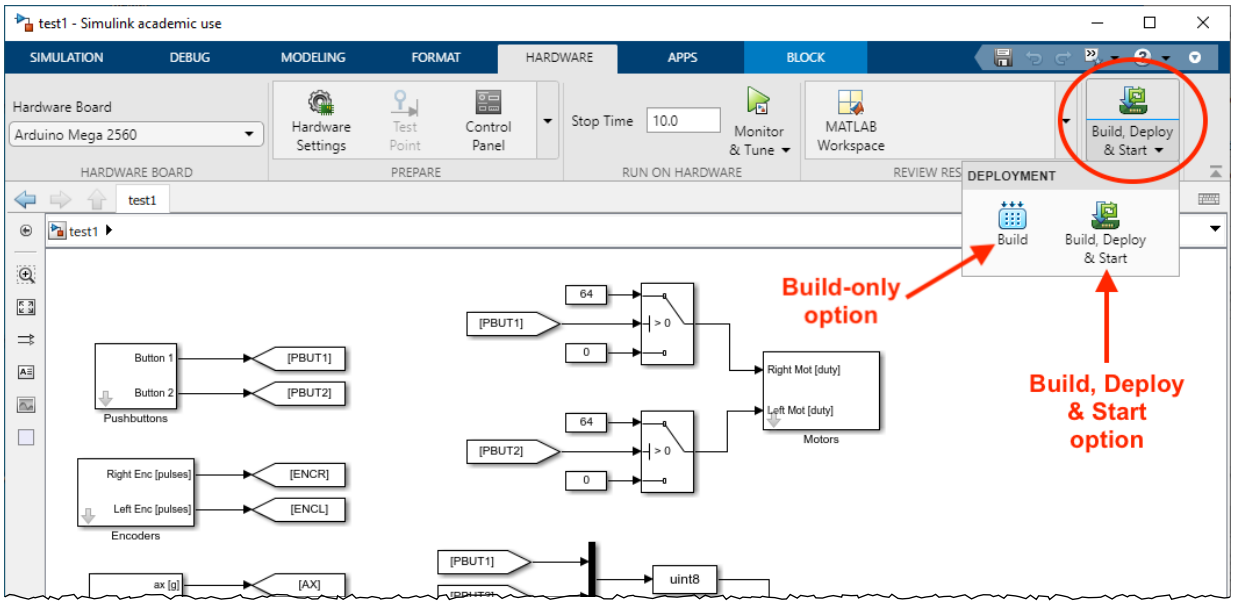
3.3 Building and deploying the model to the embedded target

Once the Simulink model is ready, it has to be converted into a binary file, and then uploaded on the flash memory of the MCU. This process, named *deployment to embedded target*, is automatically managed by MATLAB. It can be started as follows:

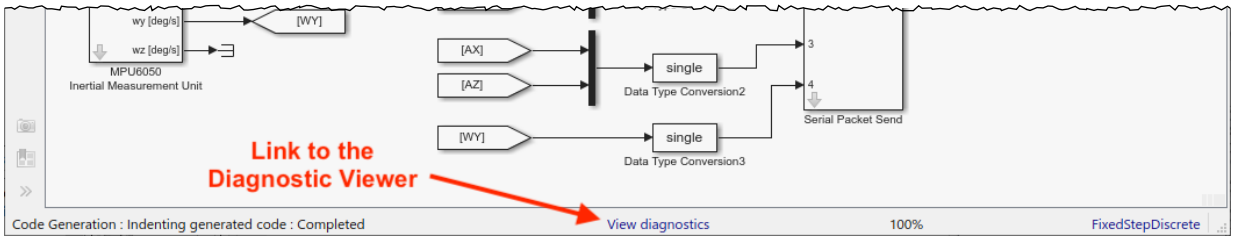
1) Turn off the power switch on the balancing robot (see (14) in Fig. 1c).

2) Connect the balancing robot to the host computer using an USB cable. On the balancing robot, the USB type–B port is located on the right side of the robot, beneath the power switch (see (11) in Fig. 1c).

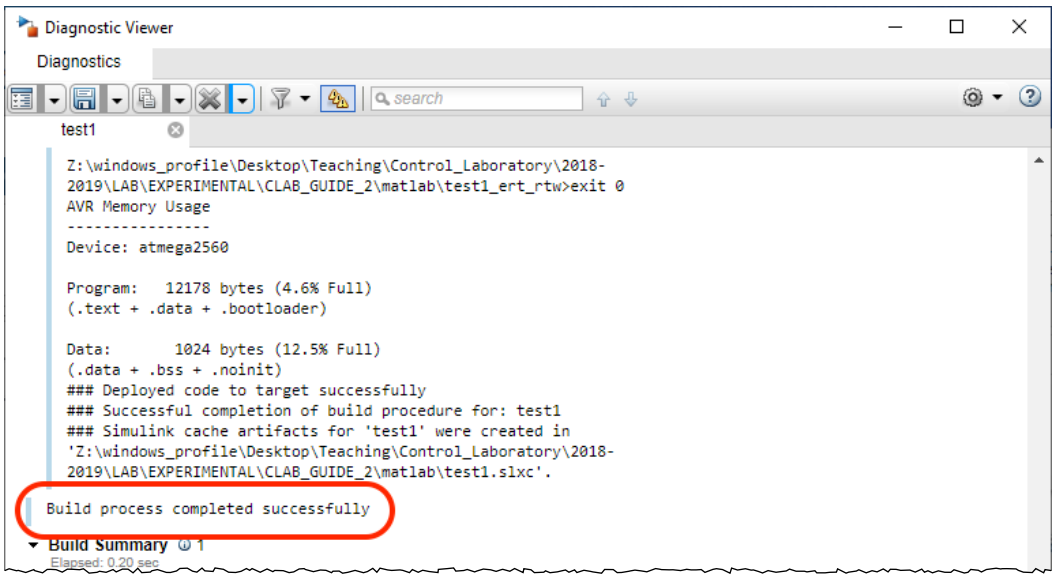
3) Push the *Build, Deploy & Start* button in the Simulink model window toolbar to start the "deployment to target" procedure, which consists of the following steps: (1) conversion of the Simulink model into an equivalent C–language code for the embedded target specified in the *Hardware Implementation* section of the *Model Configuration Parameters* window; (2) compilation (using a target–specific cross-compiler) of the source code into a target–dependent binary file; (3) uploading of the binary file into the MCU flash memory; (4) starting the code execution on the MCU.



You can check the progress of the deployment to target procedure by clicking the **View diagnostic** link at the bottom of the Simulink model window. The **Diagnostic Viewer** window opens (see next). This is the standard window used by Simulink to display information and error messages related to the current deployment to target procedure.



4) Wait until the message *"Build process completed successfully"* appears in the *Diagnostic Viewer* window. This message informs that the deployment to target procedure has been successfully completed, and the binary file has been uploaded into the flash memory of the MCU.



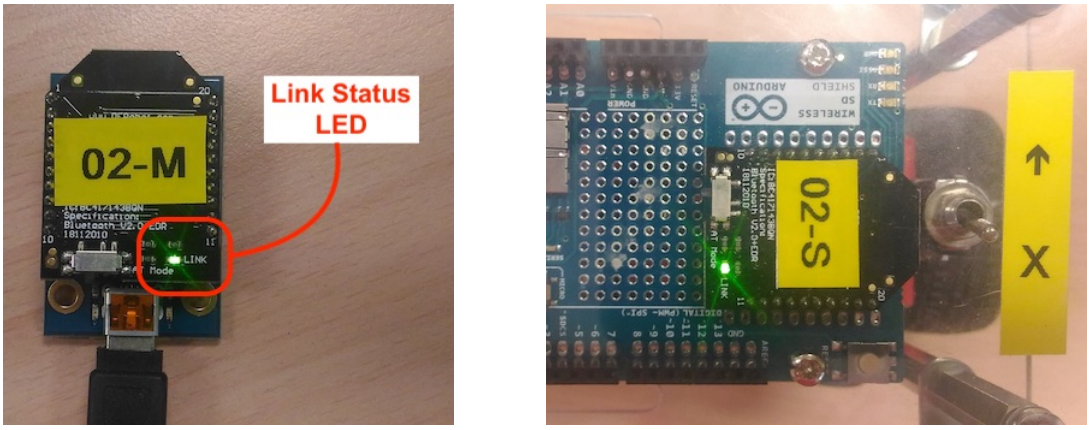
5) Remove the USB cable. The uploaded binary file is automatically started once the power switch on the balancing robot is turned on.

3.4 Data logging using the Bluetooth connection

The Bluetooth connection is established between two mated Bluetooth modules, one connected via a USB cable to the host computer (see Fig. 5a), and the other installed on the balancing robot (see Fig. 5b). The two mated modules are identified by the same number, printed on top of the module. A Bluetooth connection is automatically established between them once both the devices are turned on. A link status LED indicates when the connection is active.

Once the Bluetooth connection between the balancing robot and the host computer has been successfully established, use the `serial_datalog` routine of the BRT to receive the data from the balancing robot, and save them into a MATLAB structure. The routine can be invoked from the MATLAB Command Window with the following syntax:

`data = serial_datalog(`*COM_port*, *packet_spec*`)`



(a) (b)

Figure 5: Bluetooth modules pair: (a) *master* module connected to host computer (via USB cable); (b) *slave* module installed on the balancing robot.

The *COM_port* argument is a string with the name of the serial port used by the host computer to communicate with the Bluetooth module. By default, the port is `COM3` (on Windows machines). However, the port currently in use can be immediately detected by invoking the `seriallist` routine from the MATLAB Command Window, provided that the Bluetooth module is the only device connected to the host computer. If so, the routine should return a list of two items, of which the first is the `COM1` port reserved by the system for internal uses, and the other is the port used by the Bluetooth module to communicate with the host computer.
The *packet_spec* argument is a cell array that specifies the structure of the packet. It has to be matched with the packet specifier defined in the *Serial Packet Send* block of the Simulink model running on the embedded target (see Fig. 4).

The data received from the robot are simultaneously displayed on a MATLAB figure in real–time, and saved into a memory buffer. Once the figure is closed, the buffer content is copied into the output variable `data`, which is a MATLAB structure with the following fields:

- `data.time` is the field containing the time instants at which the data samples have been transmitted, with respect to the beginning of the data logging process.
- `data.out` is a cell array containing the data samples sent by the balancing robot. Each element of the cell array corresponds to a different logged signal. It is a matrix, whose columns are the values of the logged signal. Therefore, the number of rows is equal to the size of the signal (number of components), and the number of columns is instead equal to the number of received data samples. The number of received data samples never exceeds the size of the buffer used to hold the data during the data logging process (by default, the buffer size is equal to 1000).

  For example, the command `y = data.out{2}(2,:)` retrieves the 2^{nd} component of the 2^{nd} signal sent by the *Serial Packet Send* block (i.e. the pulse count of the left encoder, according to the Simulink model of Fig. 3).

The `serial_datalog` routine can accept some extra configuration parameters, to be passed with the following format:

`data = serial_datalog(`*COM_port, packet_spec, 'Property1', Value1, 'Property2', Value2, ...* `)`

The available properties are:

- `BaudRate`: specifies the baud rate of the serial connection. The value must match with that specified in the *Serial Packet Send* block parameters. The default value is $38400\,\text{bps}$ (i.e. the standard baud rate used for the Bluetooth connection).
- `txSampleTime`: specifies the sample time of the transmitted data. It must coincide with the sample time of the Simulink model uploaded on the MCU. The default value is $0.01\,\text{s}$.
- `BufferSize`: specifies the size (number of samples) of the buffer used for data logging. The default value is 1000 samples (for each signal sent by the balancing robot). The maximum value is 100000 samples.
- `PlotList`: specifies the list of signals to plot during the data logging process. Each signal is identified by the index of its specifier within the packet specifier cell array (i.e. the number of the input port of the *Serial Packet Send* block used to send the data). The list is an array of up to 6 natural numbers, ranging from 1 up to the number of signals stored in the packet.

For example, the list $[3, 2, 1]$ specifies to plot only three signals in the output figure. The top plot will refer to the signal associated with the input port 3 of the *Serial Packet Send* block (x and y axes accelerations), the middle plot will refer to the signal of input port 2 (right and left encoders counts), and the bottom plot will refer to the signal of input port 1 (pushbuttons values).

The default value of this property is $[1, \ldots, m]$, where m is the minimum between the number of signals stored in the packet, and the fixed value 6 (max number of allowed plots in the data logger figure).

- `PlotRefreshRatio`: specifies the number of samples to receive before updating the output figure. The default value is 4 samples.
- `PlotWidth`: specifies the time–axis width (in seconds) of the output plots. The default value is $5\,\mathrm{s}$.
- `PlotYLim1`, ..., `PLotYLim6`: specifies the y–axis range of the corresponding output plot. The default value is $[-0.1, 0.1]$.

The typical invocation of the `serial_datalog` routine for logging the data generated by the Simulink model of Fig. 3 is reported in Listing 1. The related MATLAB figure is shown in Fig. 6. Note that:

- The balancing robot must be ready to transmit data (e.g. powered on) when the `serial_datalog` routine is invoked.
- The `serial_datalog` routine must be stopped *before* turning the robot off. The routine is stopped by closing the related MATLAB figure.
- It may happen that the `serial_datalog` routine does not release the serial port used for host–target communication when stopped inappropriately (e.g. by turning the robot off *before* stopping the `serial_datalog` routine). In this situation, use the commands `clear all` followed by `instrreset` to regain control of the serial port.

Listing 1: Typical invocation of the `serial_datalog` routine.

```
%   packet specifier (must match the specifier in the "Packet Send" block)
packetSpec = {'2*uint8', '2*int32', '2*single', 'single'};

%   interactive data-logging
data = serial_datalog( 'COM3', ...      %   serial port name
    packetSpec, ...                     %   packet specifier
    'BaudRate', 38400, ...              %   serial port baud rate
    'TxSampleTime', 0.01, ...           %   TX sample time
    'TxStartCmd', 65, ...               %   TX start cmd (default)
    'TxStopCmd', 66, ...                %   TX stop cmd (default)
    'BufferSize', 10000, ...            %   RX buffer size
    'PlotWidth', 5, ...                 %   time-axis width (default)
    'PlotRefreshRatio', 10, ...         %   plot refresh ratio
    'PlotList', [1,2,3,4]);             %   plot list
```

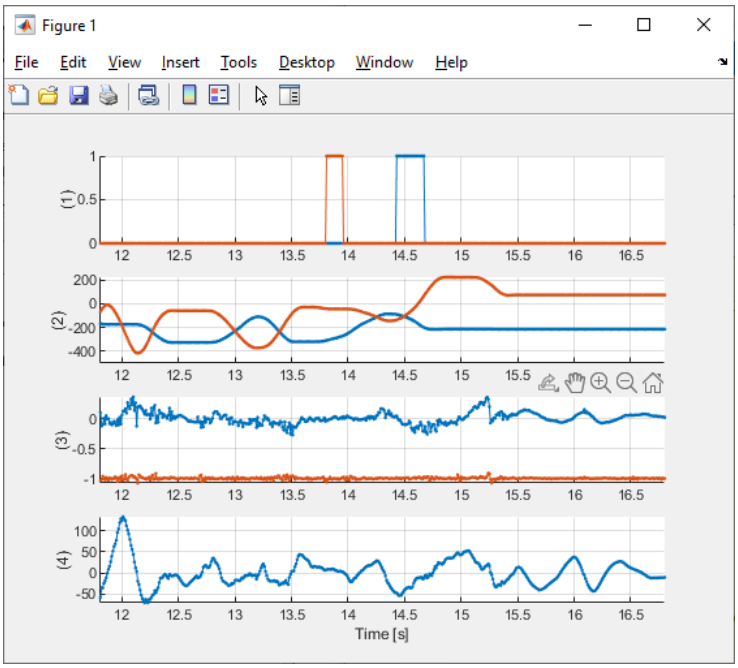


Figure 6: MATLAB figure related to the `serial_datalog` invocation of Listing 1.