Digital Systems Electronics

Lab11-STM32

DMA and Motor driving

1. Generation of a sine waveform by means of PWM

The PWM (Pulse Width Modulation) method is a low-cost way of implementing a digital-to-analog converter (DAC). By time varying the duty cycle percentage, an arbitrary waveform can be generated. In this project, we want to generate a 50 Hz sine waveform on output pin PA6 of the Nucleo board, by exploiting the PWM method.

Pulse Width Modulation (PWM) is a method of encoding a voltage onto a fixed frequency carrier wave. The frequency of the PWM is constant while the duty cycle changes between 0% and 100%. In the generation of a waveform by means of PWM, the percentage of the on time is proportional to the output signal voltage. For example, a 0% duty cycle produces a 0 V output, while a 100% duty cycle produces a peak-to-peak voltage V_{PP} equal to the I/O supply voltage, typically equal to 3.3 V in MCUs.
The basic idea of generating a sine waveform using the PWM method is to first digitize the sine wave and encode the duty cycle corresponding to each sample point. Figure shows a sine wave digitized over 12 sample points: this means that each sample is taken at the angular step of 30° of a circle. The sine value of each sample and its corresponding duty cycle is given in Table 1, for the case of 12 sample points. In general, for N samples points, the duty cycle must change according to:

$$d_k = \frac{1 + \sin\left(2\pi \frac{k}{N}\right)}{2}$$

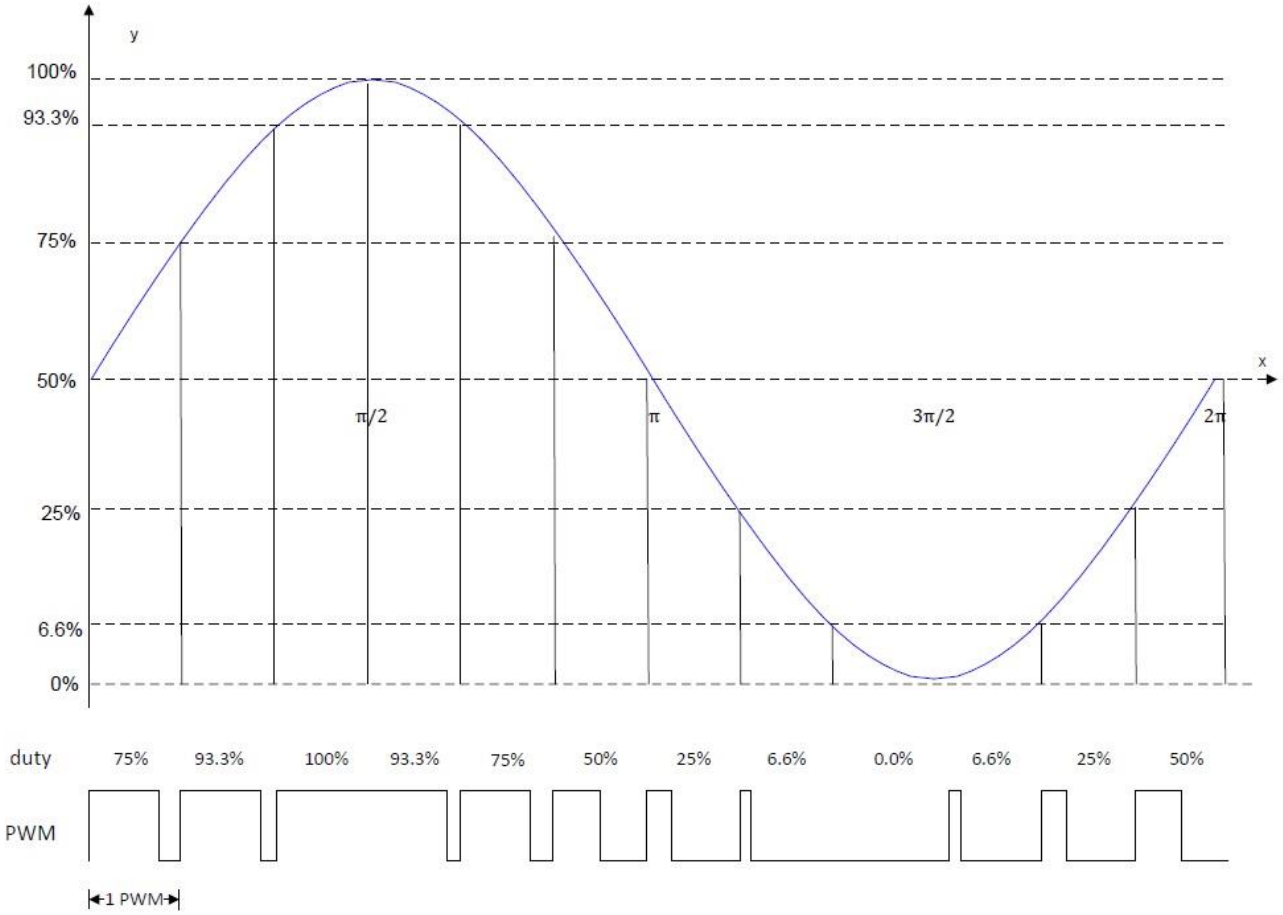


Figure 1: Sampled sine wave and PWM signal

Table 1: Digitized sine wave.

Degree	Radian	Sine value	Offset added	%
0	0.0000	0.0000	1.0000	0.50
30	0.5233	0.4998	1.4998	0.75
60	1.0467	0.8658	1.8658	0.93
90	1.5700	1.0000	2.0000	1.00
120	2.0933	0.8666	1.8666	0.93
150	2.6167	0.5011	1.5011	0.75
180	3.1400	0.0016	1.0016	0.50
210	3.6633	-0.4984	0.5016	0.25
240	4.1867	-0.8650	0.1350	0.07
270	4.7100	-1.0000	0.0000	0.00
300	5.2333	-0.8673	0.1327	0.07
330	5.7567	-0.5025	0.4975	0.25

The duty cycle at each sample point is a representation of the sine amplitude: a duty cycle equal to 1 corresponds to the highest value of the sine function, while the duty cycle 0 corresponds to the minimum value.

The frequency of the sine wave f_{sine} depends on the PWM period $1/ f_{PWM}$ and on the number of sample points N:

$$f_{\text{sine}} = \frac{f_{\text{PWM}}}{N}$$

If the output sine wave frequency is increased, either the PWM frequency needs to be increased or the number of sample points needs to be decreased. Decreasing the number of sample points increases quantization errors. To get a smooth wave, the generated PWM signal is passed through a low-pass filter. Higher is the number of sample points, lower are quantization errors. Therefore, this method introduces a trade-off between accuracy and frequency. In order to provide a more complete analysis of the sine wave generator, the spectrum of the PWM signal can be computed. Let us assume that the signal **in one period** can be written as

$$x(t) = \sum_{k=0}^{N-1} AP_{d_k}\left(t - \frac{k}{N f_{\text{sine}}}\right)$$

where $0 \leq d_k \leq 1$ and

$$P_{d_k}(t-\tau) = \begin{cases} 1 & \text{if } \tau \leq t \leq (\tau + d_k\tau) \\ 0 & \text{otherwise} \end{cases}$$

Since the signal is periodic, the **Fourier series** can be computed in order to evaluate the signal spectrum. It is possible to prove that the Fourier coefficients for this signal in the interval of integration

$$0 \leq t \leq \left(T = \frac{1}{f_{\text{sine}}} = \frac{N}{f_{\text{pulse}}}\right)$$

are equal to

$$\begin{aligned} c_n &\equiv \frac{1}{T}\int_0^T x(t)\mathrm{e}^{-j\frac{2\pi n}{T}t}\mathrm{d}t \\ &= f_{\text{sine}}\int_0^{\frac{1}{f_{\text{sine}}}} \sum_{k=0}^{N-1} AP_{d_k}\mathrm{e}^{-j2\pi n f_{\text{sine}}t}\mathrm{d}t \\ &\dots \\ &= A\sum_{k=0}^{N-1} \mathrm{e}^{-j\frac{2\pi n}{N}\left(k+\frac{d_k}{2}\right)}\frac{d_k}{N}\text{sinc}\left(\pi n\frac{d_k}{N}\right) \end{aligned}$$

It is possible to prove that the DC component is equal - as expected - to the mean value in the interval of interest

$$c_0 = A\frac{1}{N}\sum_{k=0}^{N-1} d_k = \frac{A}{2}.$$

Moreover, it is possible to prove that the only non-null contributions are multiples of $f_{PWM}=Nf_{sine}$ and some other harmonics on the sides of the previous ones. It is possible to observe the spectra for PWM signals with different number of samples N and f_{sine}=50Hz (Figure).

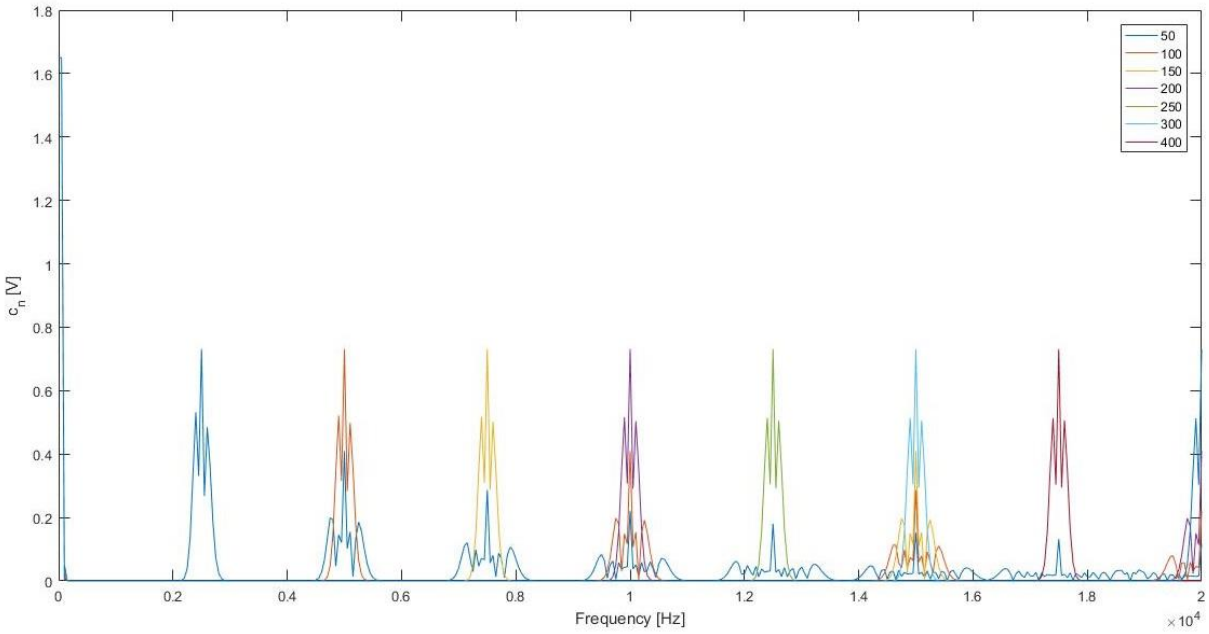


Figure 2: Spectra for different number of samples N

In this project, we use N=200 sample points. Once decided the number of sample points, we set the output frequency by changing the PWM frequency:

$$f_{PWM} = N\, f_{sine} = 200 * 50 = 10\ kHz$$

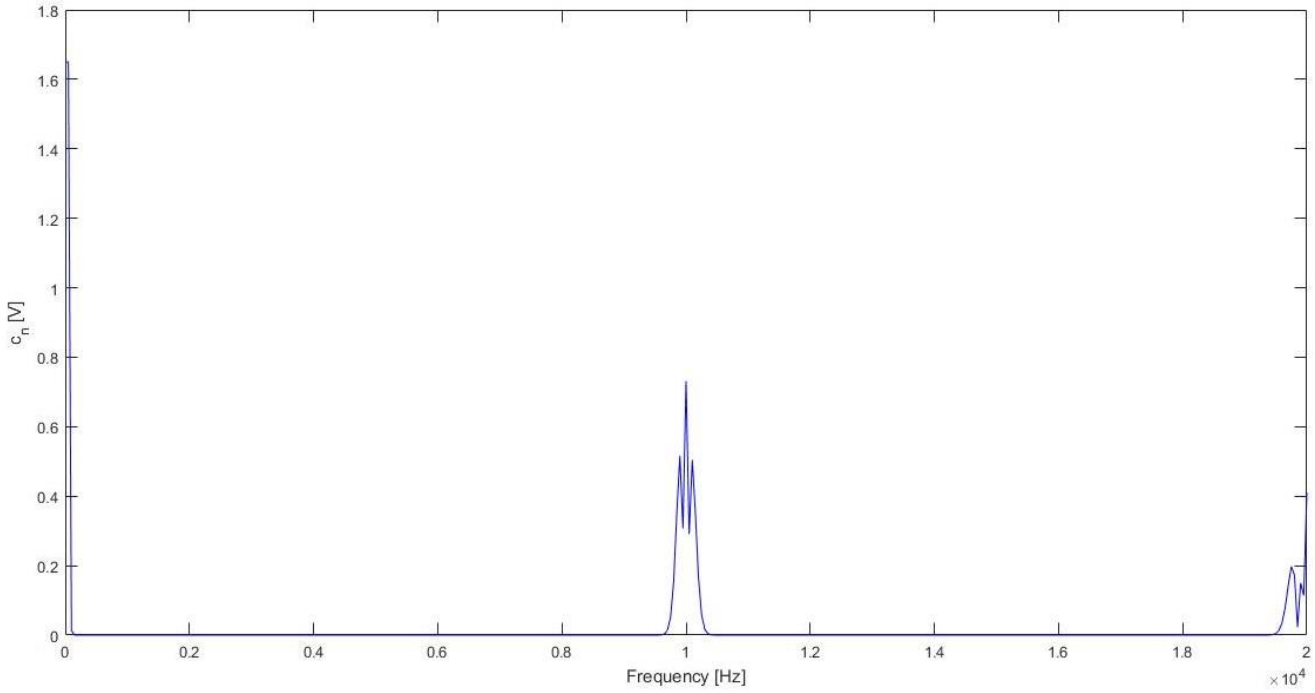


Figure 3: PWM signal spectrum (N=200)

In Figure you can see the two low-frequency components correspond to the DC and the fundamental (50 Hz), while the peak in the middle of the graph is the component at 10 kHz. Since the highest undesired component is at a frequency 200 times greater than the fundamental, a simple RC first order low pass filter is sufficient for sufficiently filtering the high-frequency harmonics. One possible choice is shown in Figure 6, where the cut-off frequency is equal to $1/2\pi RC = 724$ Hz, that is greater than the sine wave frequency by a factor 14 (more than one decade), thus ensuring that it belongs to the flat part of the filter frequency response. Moreover, since the most significant PWM components are distant from the pole frequency for more than one decade, they are attenuated by more than 20 dB, thus ensuring a negligible contribution of them on the output.

Figure shows a few periods of the generated sine waveform, for the case R=220 Ω and C=1 µF. Figure gives the PWM signal corresponding to the first half period of the sine waveform of Figure : between 0 and 10 ms, the PWM signal exhibits a duty cycle decreasing from 50% to 0 and then increasing back to 50%.
Since the sine period is divided into 200 angles, the angle step increment is equal to $360^o/ 200 = 1.8^o = 0.0314$.

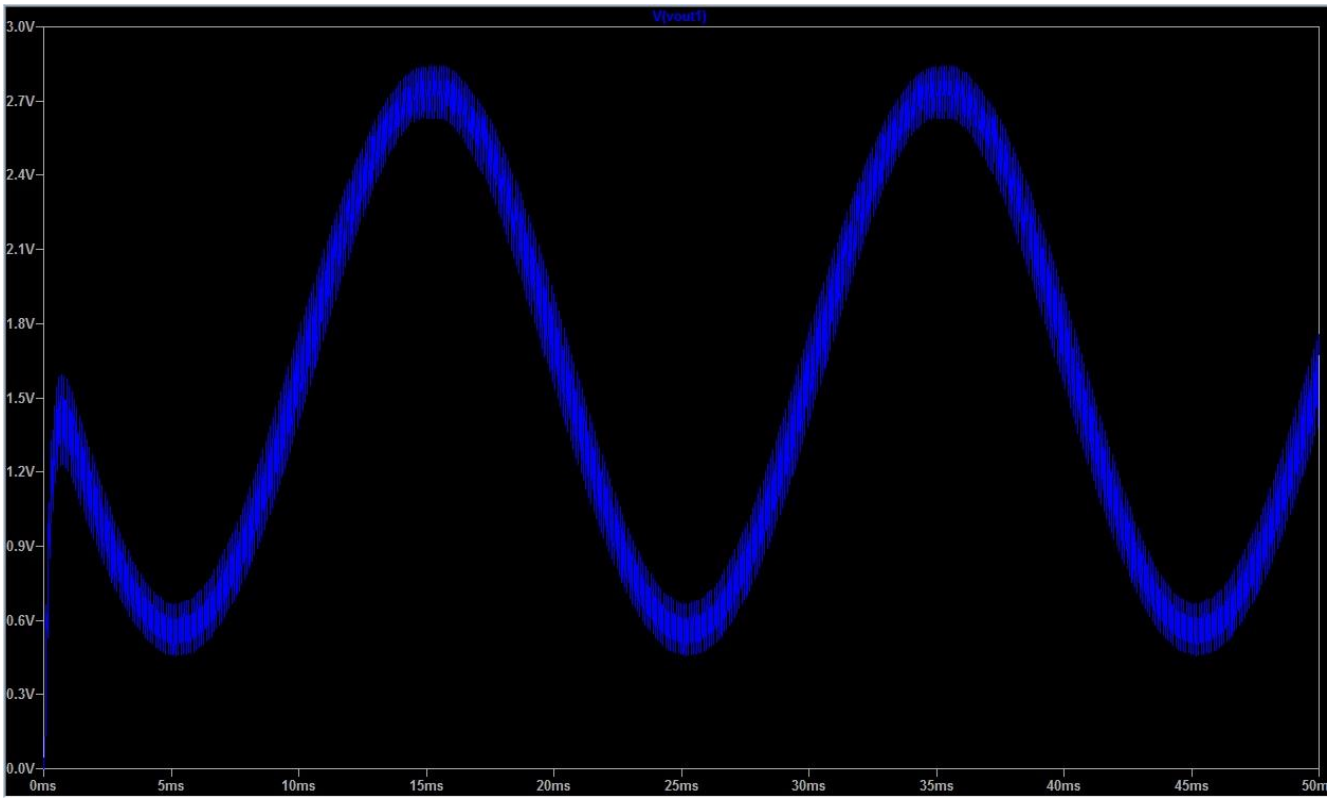


Figure 4: Sine wave generated with R=220 Ω and C=1 µF

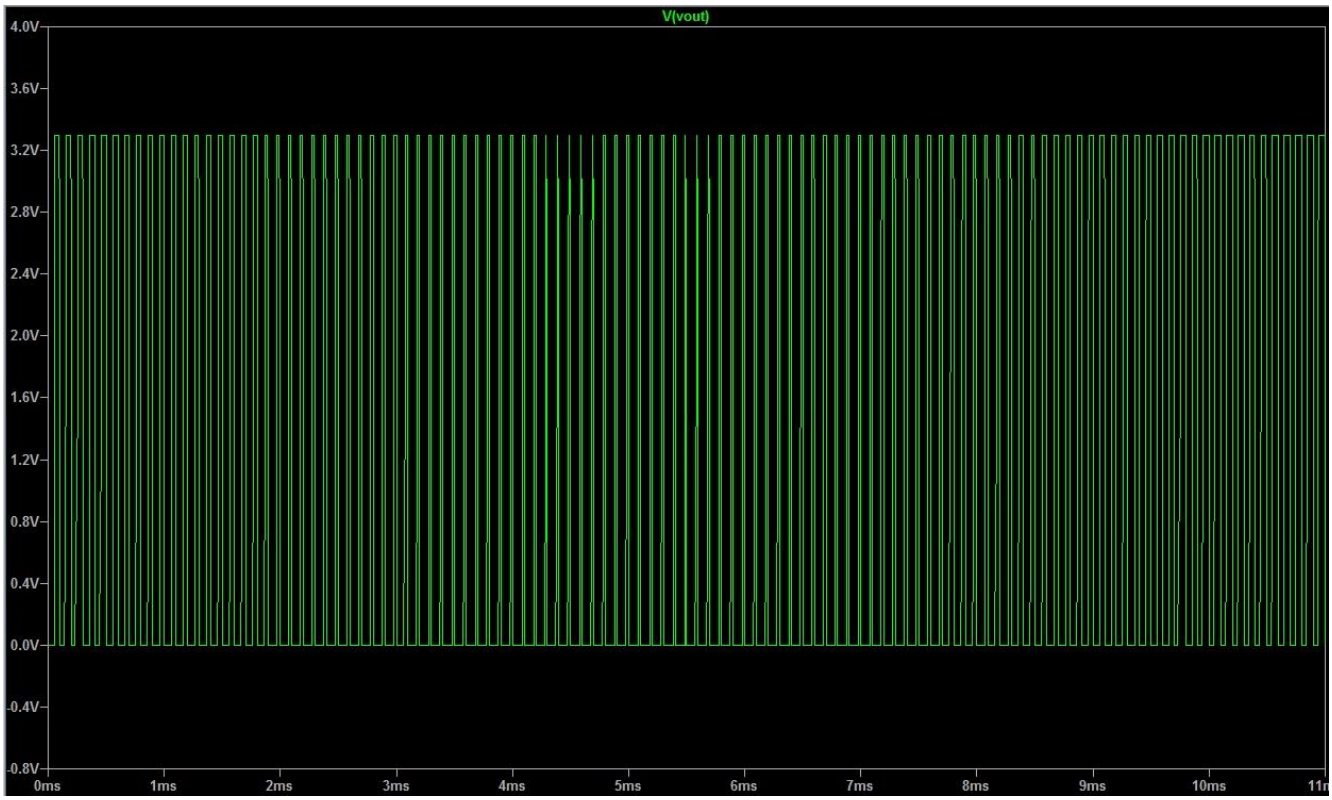


Figure 5: PWM signal (first half period of sine wave)

Two things are necessary in order to implement this method of waveform generation. First, we must store the different duty cycles needed to reconstruct the sine wave digitally. These values can be pre-calculated in a lookup table stored in the memory. Then, a mechanism for continuously transferring the duty cycles from the memory to the CCR register of the PWM unit is needed. One way to do this is to use the processor to read the sine lookup table and then write to the peripheral register. At each PWM cycle, an interrupt is generated and the corresponding ISR transfers the new duty cycle from the memory location to the PWM unit. However, this approach has poor efficiency, as it takes away processor bandwidth from performing other tasks. Moreover, the interrupt involves saving and restoring context; this can take some cycles to complete and the ISR latency may become the limiting factor in determining the fastest sampling frequency. Using a timer in DMA mode is the best way to generate arbitrary function without introducing latency and affecting the Cortex-M core. One DMA Channel can be configured to work in circular mode, so that it automatically sets the value of the CCR register according to the sine values contained in lookup table.

To complete the assigned project, you have to follow these steps:

1. Create a STM32CubeIDE project. In the Pinout view, enable the channel 1 of timer TIM3 in PWM mode (PWM Generation CH1). In the Chip view, one can see that the pin PA6 is highlighted and labeled with the assigned function TIM3 CH1. In the Configuration view, click on the TIM3 button and enter the necessary parameters, *i.e.* Prescaler, Counter Mode and Counter Period, in order to ensure the generation of a 10 kHz PWM signal, with at least 200 available duty cycles (Pulse values). In the PWM Generation part of the window, the Mode field must be set to PWM mode 1, Pulse must be equal to 0 (this value must be continuously updated) and the remaining fields can be left unchanged. Then, open the DMA Configuration window and set the parameters for the TIM3_CH1/TRIG DMA Request: Direction must be Memory to Peripheral and the Mode must be Circular. With these settings, every event

on the channel 1 of TIM3 will trigger a DMA request, which will take a data from a portion of memory handled as a circular buffer and transfer it to the CCR register of the PWM unit. Finally, generate the configuration code.

2. In the main.c file, before the infinite loop, insert instructions to calculate the content of the lookup table. You can generate the sine values by using the *sinf* C function (**search on the internet the C library containing it**), with 200 angles, obtained as the multiples of 2π/200. Then, convert the generated set of floating-point values into a set of integer values in the range 0 to *Period* (pay attention: since sin(x) is between -1 and +1 and the Pulse values are between 0 and *Period*, an offset must be inserted for calculating them). Finally, save these Pulse values in a vector of 200 elements, defined as unsigned integers. An example of definition is

```
uint16_t SINE_LUT[200];
```

Finally, enter the statement to start the DMA:

```
HAL_TIM_PWM_Start_DMA(&htim3,TIM_CHANNEL_1,(uint32_t *)SINE_LUT,200);
```

This statement specifies that channel 1 of TIM3 must be started in DMA mode, that the circular buffer is pointed by SINE_LUT and that it has length equal to 200 words. Notice that you do not need any statement inside the infinite loop, because everything is done autonomously by the three involved peripherals (the timer TIM3, the PWM unit and the DMA controller) with no involvement of the processor, except in the configuration phase.

3. Connect the RC low-pass filter to the MCU output as shown in Figure and look at the generated waveform.

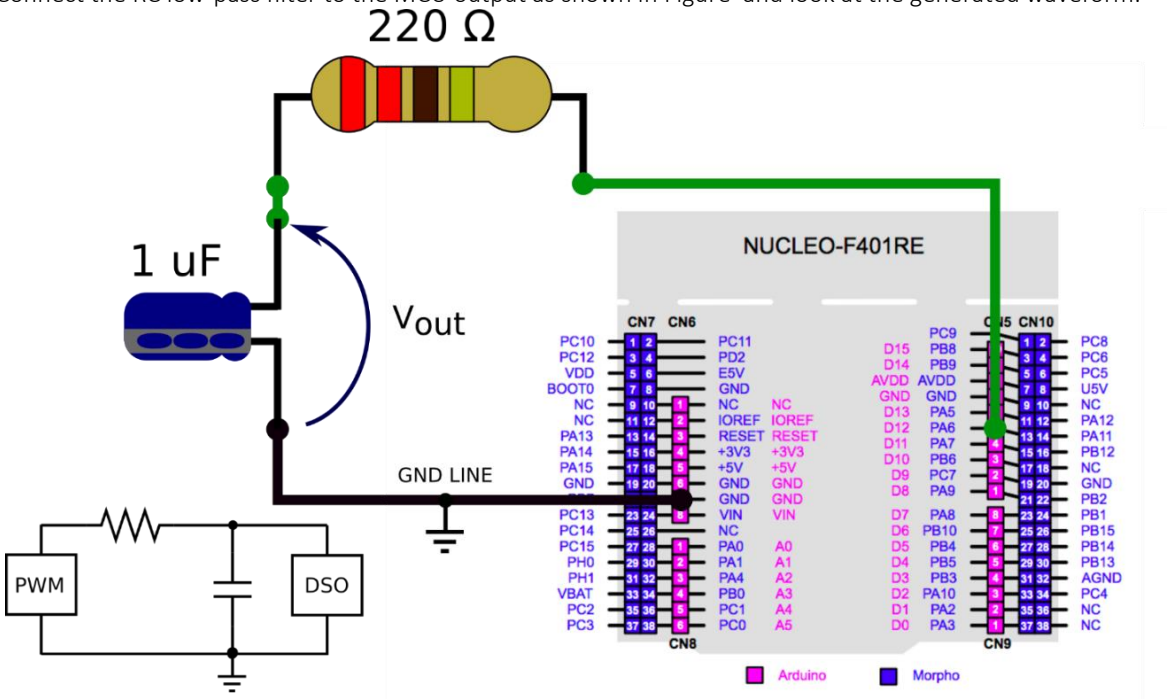


Figure 6: RC first order filter

REVIEW: HOW TO USE THE SERIAL COMMUNICATION

2. The Universal Synchronous Asynchronous Receiver Transmitter (USART)

The Universal Synchronous Asynchronous Receiver Transmitter (USART) unit of the microprocessor provides the capability of receiving and transmitting data to other devices.

The USART unit provides different features for the implementation of ad-hoc transmission among devices:

- Full duplex asynchronous communication
- Fractional baud rate generation
- Programmable data word length
- Configurable stop bits
- Single-wire communication
- Enable bits
- Transfer detection flags
- Error detection
- Parity control
- Interrupt

A detailed description of the USART is available from page 501 of the reference manual. For a standard serial communication, a minimum of two pins is required (one for RX, one for TX). The RX channel is used to receive data (from device to NUCLEO) whereas the TX channel is used to transmit data (from NUCLEO to device). Oversampling techniques are used to discriminate between data and noise. When the transmission is not enabled, the TX channel is put at high level.

The USART might be programmed using the Hardware Abstraction Level. The most important registers are:

- Status register
- Data register
- Baud rate register

For the details about the registers refer to page 542 of the reference manual.

2.1. USART configuration using STM32Cube

We use STM32Cube to obtain the configuration statements for the USART module. We start by launching the STM32Cube tool and creating a new project, with all the peripherals configured with their default Mode. **The default USART configuration is sufficient for the completion of the proposed exercises.** By enabling the default Mode, the USART2 is typically already configured on PA2 and PA3 pins, TX and RX channel respectively. The USART2 is connected to the STLINK chip present on the board (the small chip in the portion of the board with the USB connection). This enables the communication between the board and the personal computer through the same USB that you use to power and program the NUCLEO board. Alternatively, USART1 can be used to implement a communication with a third device, Figure shows the standard configuration of the USART2 module. Check that in the default configuration *Mode* is set as *Asynchronous* and the *Hardware Flow Control (RS232)* is *Disable*.

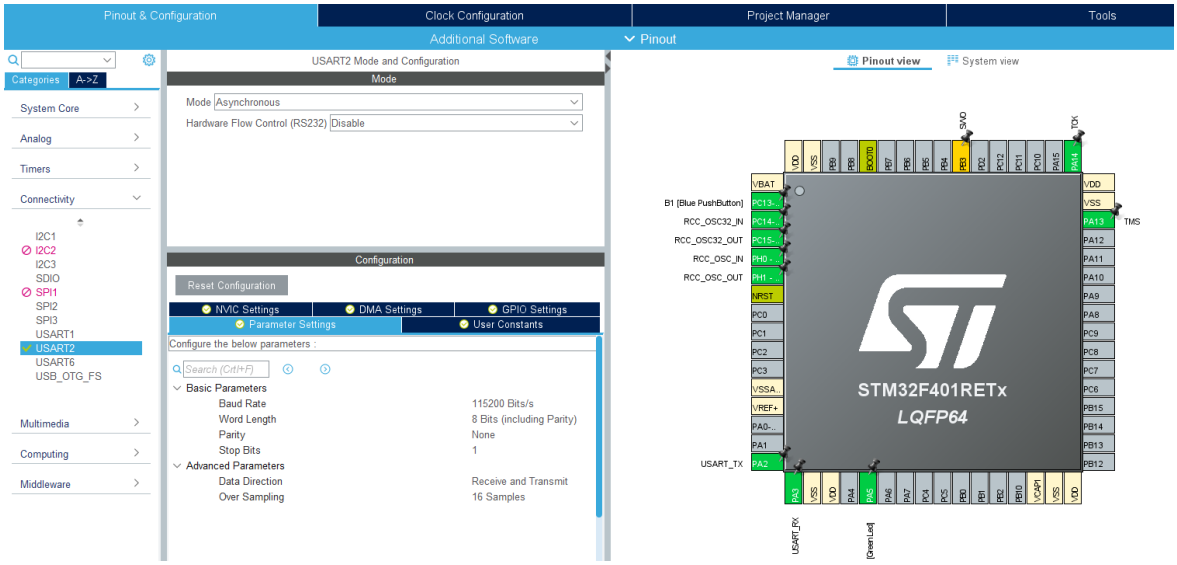


Figure 7: Pinout with USART2 enabled.

In the Configuration View, open the USART2 panel and check the Parameter Settings. All the parameters of the transmission should be inserted as in Figure . In *NVIC Settings* enable the USART2 global interrupt. The code generate by STM32Cube automatically configures the serial interface. You must only insert the specific code for your application.

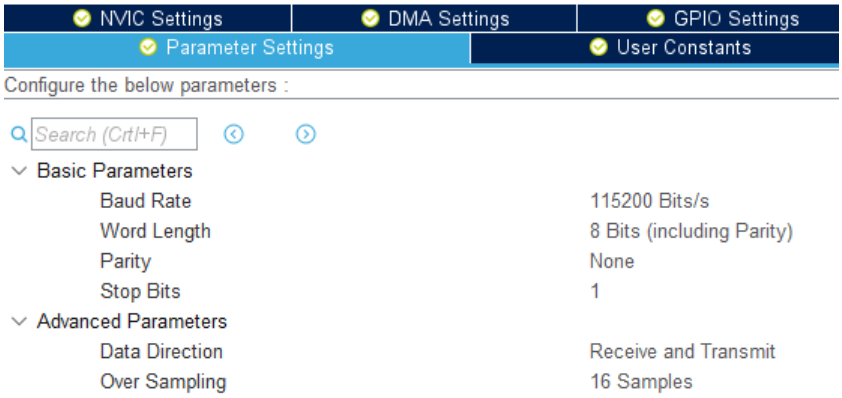


Figure 8: configuration of the USART2.

2.2. USART with the HAL programming paradigm

Communication can be performed in two modes:

1. *Blocking mode*. The communication is managed by exploiting Polling. The status of the transmission is returned by the same function after finishing the data transfer. The main program is blocked until the communication is ended.
2. *Non-blocking mode*. The communication is performed using Interrupts or DMA, these APIs return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.

In this laboratory, non-blocking communication is required. It is automatically configured by STM32Cube.

The Hardware Abstraction Layer provides some macros useful to manage the USART module and the communication. In particular, the `HAL_UART_Transmit_IT()` function allows sending a message from the board to the serial communication interface. At the end of the transmission, the operation generates an interrupt. Obviously, the interrupt must be enabled in the NVIC configuration using STM32Cube.

- `HAL_UART_Transmit_IT(__INSTANCE__,__DATA__,__SIZE__);`
    - `__INSTANCE__`: USART instance, properly pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
    - `__DATA__`: pointer to uint8_t data intended to be sent
    - `__SIZE__`: amount of data (**in bytes**) to be sent.

The opposite function is the `HAL_UART_Receive_IT()`. It configures the UART peripheral to receive a data from the serial interface. The execution of the program is continued (non-blocking interrupt mode), then the execution is blocked as soon as the data are received and an interrupt is generated.

- `HAL_UART_Receive_IT(__INSTANCE__, __DATA__,__SIZE__);`
    - `__INSTANCE__`: USART instance, properly pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
    - `__DATA__`: pointer to uint8_t data intended as RX buffer
    - `__SIZE__`: amount of data (**in bytes**) to be received.

The following user callbacks will be executed respectively at the end of the receive or transmit process.

```
void HAL_UART_RxCpltCallback (UART_HandleTypeDef *huart)
{
      // insert your code
}

void HAL_UART_TxCpltCallback (UART_HandleTypeDef *huart)
{
      // insert your code

}
```

Starting a new UART reception in the callback of the previous one could lead to problem during execution. A better approach consists in setting a flag in the callback and reading the flag in the main loop in order to trigger a new receive. Furthermore, you should minimize the code in the callbacks: it is better to perform all the requested instruction in the main loop.

For the sake of completeness, we report a commented example. Notice that this example cannot be executed as it is, use it as a mean of understanding how the UART can be managed.

```
int volatile correctlySentData = 0;
int volatile correctlyReceivedData = 0;
```

```
int main ()
{

  // Print welcome message
  HAL_UART_Transmit_IT(&huart2, "Hello world!!", sizeof("Hello world!!"));

  // Prepare UART to receive a single character
  char character[2]; // Pointer to received data
  HAL_UART_Receive_IT(&huart2, character, 1);

  while (1)
  {
         // Check TX flag
         if (correctlySentData == 1) {
             correctlySentData = 0;
             // …
         }

         // Check RX flag
         if (correctlyReceivedData == 1) {
             correctlyReceivedData = 0;
             // …
         }
  }

}

void HAL_UART_TxCpltCallback (UART_HandleTypeDef *huart)
{
       \\ Set TX flag
        correctlySentData = 1;
}

void HAL_UART_RxCpltCallback (UART_HandleTypeDef *huart)
{
       \\ Set RX flag
        correctlyReceivedData = 1;
}
```

3. Connecting the board to the PC

You will use a script written in Python to connect the PC to the NUCLEO board. **The script has all the functionalities to handle all the exercises**. In the top you will have to setup connection parameters and select the correct COM port. On the left you have a log with all the messages received from the board and a line edit in the bottom where you can input your commands (for all the exercises). On the right there is the graph that can be used to display the motor speed, see Figure 9. The script will automatically load the value and plot the trace of the motor speed, but you have to use specific text messages to send the information through the USART:

- "Speed: XX" when you need to send the speed value.
- "New target speed: XX rpm" when you update the target speed (OPTIONAL only)

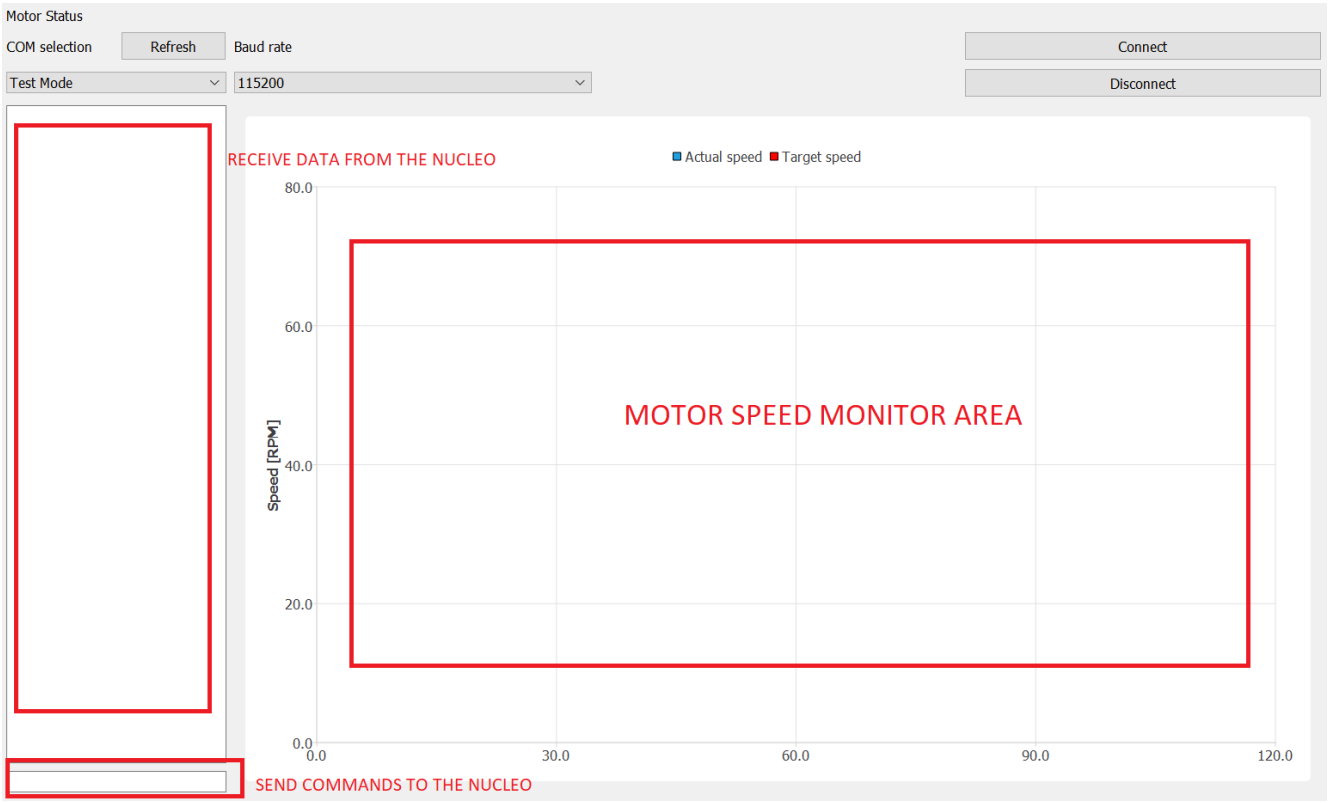


Figure 9: Script for the transmission control.

4. DC motors

Pay attention: the notation employed for the electrical quantities in DC, time domain and Laplace domain are the same introduced in the Electronic Circuits course!

4.1. Introduction

A DC motor is any of a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields when current flows in a loop of wire.

DC motor consists of two parts: rotor (or armature) and stator (Figure). The stator is the stationary part of the motor, while the rotor rotates with the respect to the stator. When both armature and field are excited by DC supply, current flows through windings and magnetic flux proportional to the current is produced. When the flux of field magnets interacts with the flux of armature, it results in motion of the rotor.

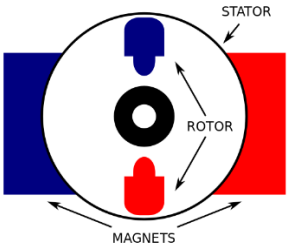


Figure 10: Basic components of a DC motor.

In this laboratory we will focus our attention on an armature-controlled DC motor, whose equivalent electrical circuit is reported in Figure. The rotation speed of the motor is modified by changing the current flowing in windings. The relation between the torque and current flowing in the armature is:

$$\tau = K_\tau i$$

where K_τ is a proportionality coefficient named torque constant. The rotational electromotive force induced in the armature because of the flowing of the current and the consequent generation of the concatenated magnetic field is proportional to the product of speed and flux. It can be written as

$$EMF = K_e \omega$$

The constant K_e is generally known as the voltage constant. For DC motors the torque constant and voltage constant have typically the same value, therefore $K_\tau = K_e = K$.

The equivalent circuit of a DC motor without load is the following.

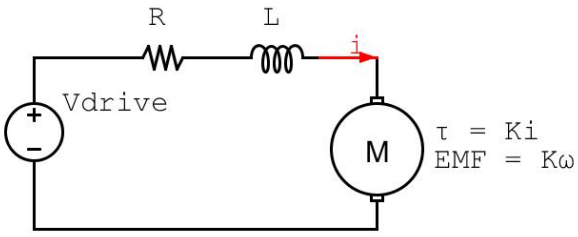


Figure 11: Armature-controlled DC motor equivalent circuit.

R and L are the equivalent resistance and inductance of the armature. It can be described by a Kirchoff voltage equation

$$v_{drive} - Ri - L\frac{di}{dt} - K\omega$$

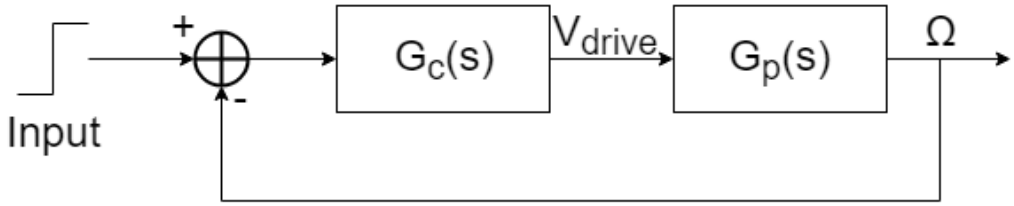
In addition to the electrical equation, a mechanical one related to the torque applied to the rotor must be written

$$\tau = J\frac{d\omega}{dt} + B\omega$$

where J is the inertia moment of the rotor and B is the viscous friction coefficient. Moving into the Laplace domain, we can derive the transfer function of the system, relating the angular frequency of the motor rotation with the input voltage.

$$G_p(s) = \frac{\Omega(s)}{V_{drive}(s)} = \frac{K}{(sJ + B)(sL + R) + K^2}$$

It is a second-order transfer function. We can finally write the block scheme of the negative-feedback control system.



4.2. DC characteristics

The electrical equation can be exploited for the derivation of the DC characteristics of the motor. Assuming a steady-state condition, so that the inductor behaves as a short-circuit, we can derive the current as

$$I = \frac{V_{DRIVE} - K\omega}{R}$$

Then the torque can be written as

$$\tau = KI = K\frac{V_{DRIVE} - K\omega}{R} = \frac{KV_{DRIVE}}{R} - \frac{K^2}{R}\omega = \tau_{stall} - S\omega$$

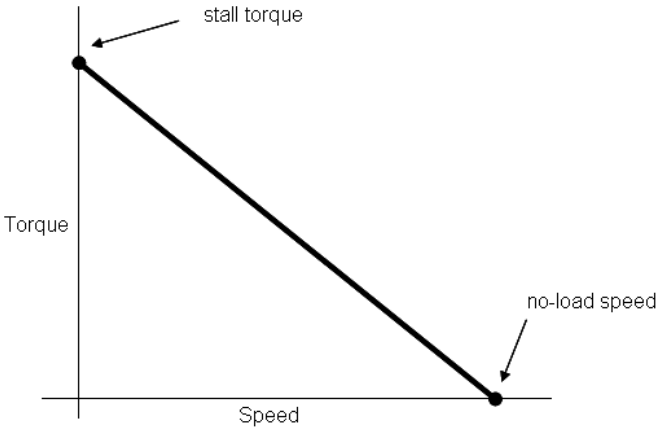
According to the previous equation, it depends linearly with the respect to the angular frequency, with a proportionality factor *S* commonly known as **steepness**. The torque at zero speed is named **stall torque**.

A parameter that is typically reported in the datasheets is the **no-load speed**, *i.e.* the angular frequency for τ = 0 Nm and *I* = 0 A. Taking the previous equation into account, it is equal to

$$\omega_0 = \frac{V_{DRIVER}}{K} = \frac{\tau_{stall}}{S}$$

We can finally write the general torque equation and the corresponding torque-speed curve.

$$\tau = S(\omega_0 - \omega)$$



In some datasheets, the **nominal torque** is indicated instead of the stall torque. Any DC motor has an efficiency that can be defined – in steady-state conditions -as

$$\eta \equiv \frac{P_{out}}{P_{in}} = \frac{\tau\omega}{V_{DRIVER}I} = \frac{\tau\omega}{RI^2 + K\omega I}$$

where the input power is the electrical and the output power is mechanical. The nominal torque is related to the operating point where the efficiency is maximum, *i.e.* it is the closest to 1.

The DC motor exploited in this laboratory is the LE149-12-43 by Micromotors. Its electrical and mechanical characteristics are exactly the same of L149-12-43 DC motor (Figure and http://www.micromotorssrl.com/gear_motor_l149.html): it requires a voltage supply of 12 V and the gear reduction ratio (related to the number of full rotations of the motor to achieve one full rotation of the output shaft) is equal to 43.3. It differs from the L149-12-43 one because of the presence of a Hall-effect encoder. The encoder is based on a six-pole magnet, so it provides three pulses for motor turn. The pinout is reported in Figure .

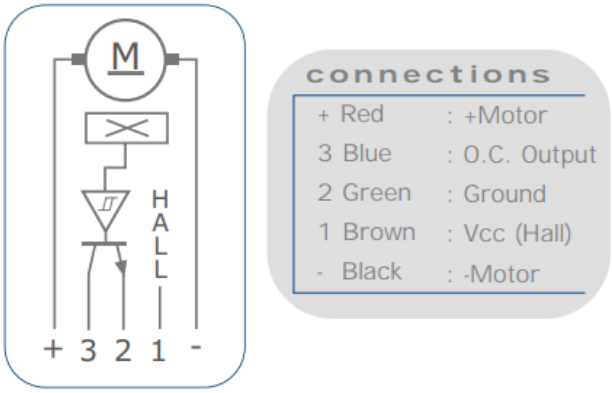


Figure 12: LE149 pinout

TECHNICAL DATA - series L149

VDR interference suppression on the collector
Precious metal brushes (Au-Ag-Cu)
Direction of rotation depending on polarity
Can be mounted in any position
Maximum radial shaft load: 10N
Maximum axial shaft load: 5N
Temperature range: -20 °C/60 °C
Approx weight: 55g

Typical values at ambient temperature +20% - Tollerance +/-10%

TYPE			NOMINAL VOLTAGE	L	RATIO TO:1	NOMINAL TORQUE	SPEED		CURRENT	
							NO LOAD	AT NOMINAL TORQUE	NO LOAD	AT NOMINAL TORQUE
			v	mm		Ncm	rpm		mA	
L149	4 6 12	10	4,5 6 12	36	10	1,5	255 215 255	165 120 165	<35 <30 <20	100 85 50
L149	4 6 12	21	4,5 6 12	36	20,8	2,5	125 105 125	80 60 80	<35 <30 <20	100 85 50
L149	4 6 12	43	4,5 6 12	41	43,3	3,8	60 52 60	40 32 40	<35 <30 <20	100 85 50
L149	4 6 12	90	4,5 6 12	41	90,3	8	30 25 30	18 13 28	<35 <30 <20	100 85 50
L149	4 6 12	188	4,5 6 12	46	188	14	14 12 14	9 7 9	<35 <30 <20	100 85 50
L149	4 6 12	392	4,5 6 12	46	391,8	20	7 6 7	5 4 5	<35 <30 <20	90 75 45

Figure 13: Electrical and mechanical characteristics of the DC motor.

It is possible to observe that the electrical characteristics of the equivalent circuit of the motor are not available in the datasheet. However, we can derive the resistance and the inductance through direct measurements, done with the HP 4192A LF Impedance Analyzer (see https://web.sonoma.edu/esee/manuals/04192-90011.pdf for the manual) available in LED laboratories. The obtained parameters are L = 79 mH and R = 89.6 Ω for the frequency f = 100 Hz.

Another approach exploitable for deriving the resistance is based on the speed and current parameters in Figure . These quantities are indicated without load and at the nominal torque, *i.e.* the torque value associated to the most efficient operating point of the DC motor, that is equal to $\tau = 0.038$ Nm. Linear regression permits to derive $\tau(\omega)$, with the consequent

steepness S = 0.0181 Nm/(rad*s^{-1}). Then we can exploit the torque range derived by the linear regression and the current values without load and at the nominal torque for obtaining the current values $i = \tau/K$ through another linear regression, thus deriving the torque constant K = 1.267 (Nm)/A. Since the steepness and the torque constant are available, we can derive the internal resistance as $R = K^2/S$ = 88.4 Ω, whose value is compatible with the measured one.

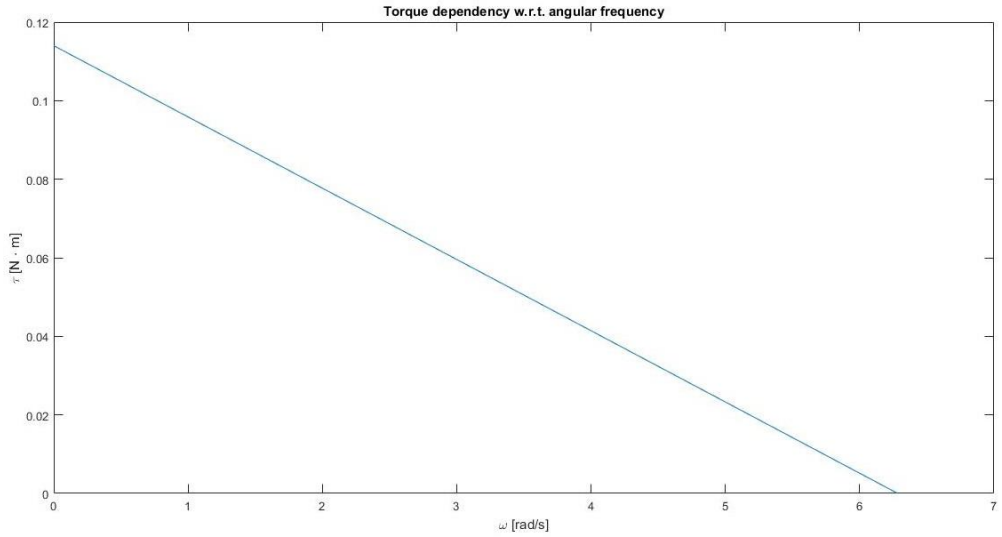


Figure 14: Derivation of torque dependency with the respect to angular frequency.

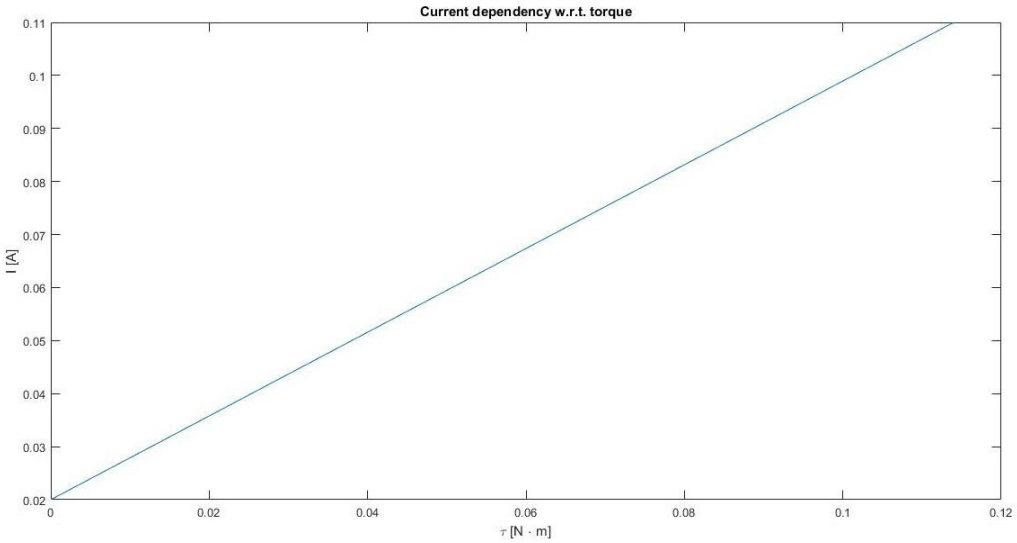


Figure 15: Derivation of current dependency with the respect to torque.

4.3. DC motor and Nucleo board

The circuit to be mounted is the following:

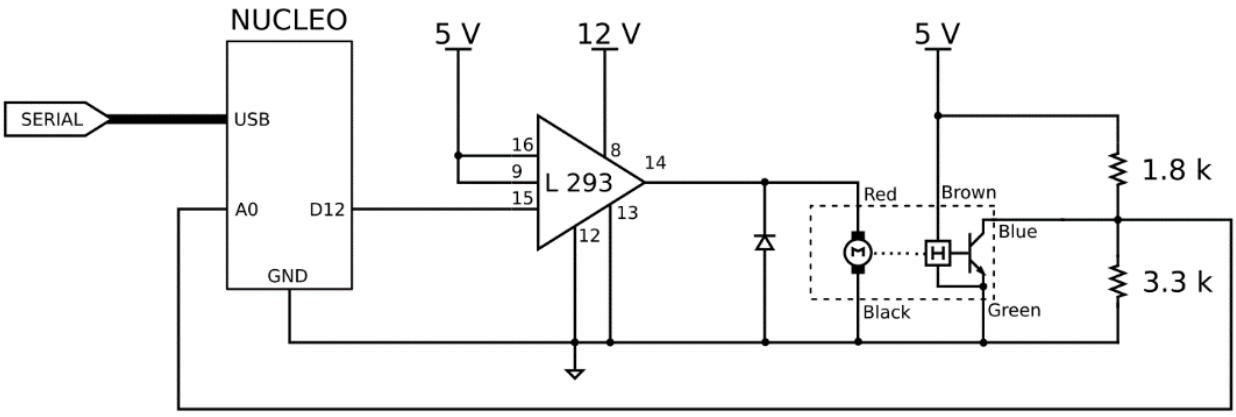


Figure 1: DC motor speed control circuit.

A L293 driver is required for providing the sufficient current to the DC motor. Pay attention that the protection diode must be always connected and that the ground is the same for all the components of the circuit. The Hall sensor is connected to an Open-Collector output, thus obtaining an output square wave between 0 V and 5 V. In order to ensure a 0-3.3 V voltage range for the square wave to be provided to the *Nucleo* board, a voltage divider is required (3.3/(1.8 + 3.3) ≈ 3.3/5). The speed can be derived by measuring the frequency of the square wave as

$$v[\text{rpm}] = \frac{60 \times f_{sw}[\text{Hz}]}{43.3 \times 3}$$

where 43.3 is the gear reduction ratio and 3 is related to the fact that three pulses are generated for a single motor turn.

For what concerns the controller, the block scheme of the circuit is the following

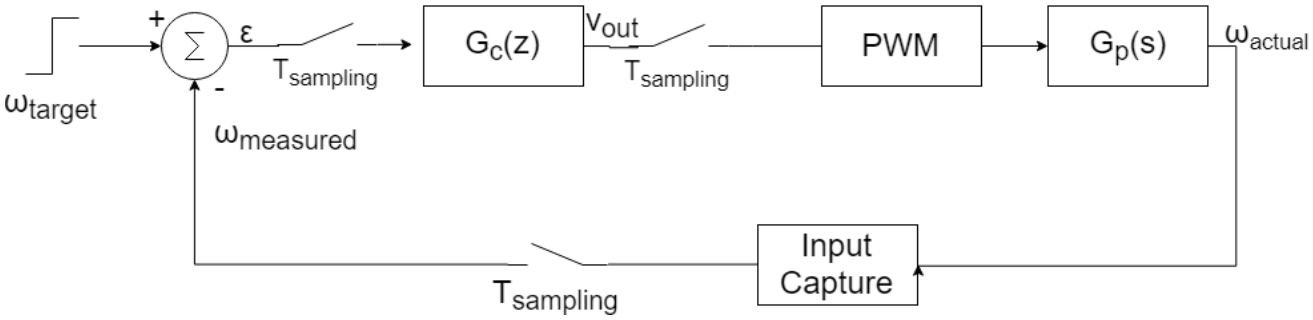


Figure 2: Block scheme of the system.

The input of the controller is the difference ϵ between the target and measured angular velocities

$$\epsilon = \omega_{target} - \omega_{measured}$$

and it provides a voltage as output. We remember that the speed must be measured in Input Capture mode, but the error computation is done periodically with sampling period $T_{sampling}$ (to be configured in Output Compare).
The output of a proportional controller is

$$v_{out} = K_p \epsilon$$

while the output of an integrative controller is

$$v_{out} = Ki \sum_{\text{all samples}} \epsilon_{sample} T_{sampling}$$

The choice of the values of K_d, K_i and $T_{sampling}$ is at your discretion. The output voltage is related to a PWM signal, so it must not be a negative number and the driving voltage must not exceed the supply voltage of the DC motor (12 V) since in all these cases the duty cycle would be 100%. From a programming point of view, we suggest the insertion of a dynamics delimiter, forcing a voltage equal to 0 V (and a null duty cycle) when the output of the controller is negative and a voltage equal to 12 V for all voltages greater or equal than 12 V.

It is important to precise that if an oversampling strategy ($T_{sampling} << \min\{T_{motor}\}$) is chosen, the probability of measuring a null time interval between the last two edges is very high, due to the fact that the values of the last and second-last edges can be the same. It is preferable to update the measured time interval for the estimation of ϵ only if the measured time is different from 0. Moreover, if the speed of the motor is null, the Hall sensor does not provide any square wave on the output, thus avoiding the Input Capture update of the speed variable. This phenomenon can have dramatic consequences on the reliability of your controller. In order to overcome this problem, you can use an edge counter variable that is incremented whenever an edge occurs. With a "watchdog" timer (output compare without output and period T = 0.5 s – equal to that of serial transmission - is sufficient) you can verify whether two edges in the watchdog time interval took place or not. If this condition is not satisfied, the speed is reset to 0.

References

[1] https://www.st.com/en/development-tools/stm32cubeide.html
[2] http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-nucleo/nucleo-f401re.html#design-scroll
[3] UM1724 User manual (STM32 Nucleo-64 board), Nov. 2016 (description of the Nucleo board).
[4] RM0368 Reference manual, May 2015 (guide to the use of memory and peripherals in the STM32).
[5] Agus Kurniawan , "Getting Started With STM32 Nucleo Development", 1st Edition, ISBN 9781329075559, 2015
[6] Carmine Noviello, "Mastering STM32", 2017, available for sale at http://leanpub.com/mastering-stm32
[7] http://www.st.com/en/ecosystems/stm32cube.html?querycriteria=productId=SC2004