



**Politecnico  
di Torino**

## **POLYTECHNIC OF TURIN**

Master's degree in Mechatronic Engineering

A.Y. 2020/2021

Graduation session: July 2021

# **“Position control of a linear axis with Arduino board and Matlab interface”**

**Tutors:**

**Prof. A. Mura**

**Prof. L. Mazza**

**Prof.ssa F. Curà**

**Ing. E. Goti**

**Candidate:**

**Raffaele Meligrana**

**Mat : 268000**

# SUMMARY

The topic of the thesis concerns the development of the position control of a linear axis, an axis composed of an electric motor controlled in position and a screw that transforms the rotary and linear motion. In particular, the motor is a DC motor; the sensor used to control the position and speed is an encoder. The type of control is a PID carried out by means of an Arduino Board interfaced with Matlab.

The thesis project develops in three parts:

1. A brief introduction of the theory on the DC motors and PID controller, its origin, its applicability. Moreover, the tuning and all the adjustment methods are described: such as manual tuning, open loop Ziegler-Nichols method and Relay method;
2. The construction of the model starting from the DC motor identification and its analysis through the theory of the PWM and its possible applications in controlling the motor;
3. Practical analysis starting from the equipment description: a system alimented by Arduino, which is an opens source electronic board that includes a microprocessor, digital and analog input/output and some interfaces such as incremental encoders.

The third part has been developed in the DIMEAS department of the Politecnico of Turin in four months. The first step was to interface Matlab with Arduino in order to manually control the motor via PC and to read the position and the speed of the incremental encoder. To ensure that, Arduino communicates directly with the PC – a Matlab interface was created through the GUI (Graphical user interface) – thus allow to control the direction of rotation and speed through some point-and-click control over software application. Two different code were written in order to perform two different tests:

1. The first one was executed with a small DC motor, a slider to adjust the speed and three pushbuttons: one to perform counter clockwise rotation and one to stop the DC motor;

2. The second one had a different implementation: sending a set of analog data and different speed already set. The implementation of Arduino and the connection with the motor via the breadboard are the same as the first test.

The final GUI is designed in order to permit to set the constants of the PID control and the target speed directly. Furthermore, it allows sending the mas a command to Arduino through a simple pushbutton.

For the model realization, the work was divided as follows:

- **Chapter 1:** The first chapter concerns the didactic purpose and hardware used, including the various datasheets and their functions within the project:
  1. DC motor;
  2. Encoder;
  3. Motor driver L298 H-bridge;
  4. Arduino.
- **Chapter 2:** This chapter analyses the low power DC motor, the equilibrium analysis, the efficiency, the transient behaviour and the DC motor modelling.
- **Chapter 3:** In the third chapter, there is a brief review on the theoretical part behind the controller, PWM theory and PID controllers, the origins, the adjustment methods, the stability and the limitations.
- **Chapter 4:** it is focuses on the model realized by means of the MATLAB tool. Firstly, it was analysed how Arduino and Matlab are interfaced. Secondly, the chapter presents in detail how to create a GUI on Matlab and the entire interface taken in consideration. Thirdly, it describes the Arduino codes for speed and position controls, and the test carried out with the anti-windup technique.

- **Chapter 5:** in this chapter, the realized model was validate in laboratory performing some measurements of actuation on the real system.
- **Chapter 6:** the sixth chapter furnishes information on the possible future implementation with the linear axis and project conclusions.



## INDEX

<b>List of figures</b>	<b>7</b>
<b>List of tables</b>	<b>10</b>
<b>1. Introduction</b>	<b>11</b>
1.1 Didactic purpose	11
1.2 Hardware	11
1.2.1 <i>DC Motor</i>	12
1.2.2 <i>Encoder</i>	14
1.2.2.1 <i>Encoder Minirod 421</i>	15
1.2.3 <i>Motor Driver Module-L298N</i>	17
1.2.3.1 <i>L298N Features &amp; Specifications</i>	18
1.2.3.2 <i>L298N Module Pin Configuration</i>	19
1.2.4 <i>Arduino</i>	19
1.2.4.1 <i>Arduino UNO</i>	20
<b>2. DC Motor identification</b>	<b>23</b>
2.1 Equilibrium analysis	23
2.2 DC Motor Stall torque and No-load velocity	25
2.3 DC Motor efficiency	26
2.4 Transient behaviour	27
2.5 DC motor Modelling	28
<b>3. Types of control</b>	<b>31</b>
3.1 PID control	31
3.1.1 <i>Mathematical form</i>	32
3.1.2 <i>Origins of the PID</i>	32
3.1.3 <i>Theory of the PID control</i>	33



3.1.3.1	<i>Proportional term</i>	34
3.1.3.2	<i>Integral term</i>	35
3.1.3.3	<i>Derivative term</i>	36
3.1.4	<i>Loop tuning</i>	36
3.1.5	<i>Stability</i>	37
3.1.6	<i>Adjustment methods</i>	37
3.1.7	<i>Limitation of PID control</i>	38
3.2	<i>PWM theory</i>	38
3.2.1	<i>Time proportioning</i>	40
3.2.2	<i>PWM sampling theorem</i>	40
3.2.3	<i>Applications</i>	41
<b>4.</b>	<b>Modelling and Simulation</b>	<b>42</b>
4.1	How to interface Matlab and Arduino	42
4.2	GUI application creation	43
4.3	PID speed control and final graphic interface	47
4.3.1	<i>Arduino Code</i>	47
4.3.2	<i>Matlab GUI</i>	51
4.4	PID position control	52
4.5	Anti-Windup, speed and position control	55
<b>5.</b>	<b>Model validation and results</b>	<b>57</b>
5.1	DC Motor Identification	58
5.2	Experimental results of speed control with Matlab graphic interface	59
5.3	Experimental results of speed control with Matlab graphic interface at different speeds	62
5.4	Experimental results of the position control on Arduino	65



5.4.1	<i>Experimental result of the position control with fixed target position</i>	66
5.4.2	<i>Experimental result of the position control with sinusoidal position</i>	67
5.5	Experimental result of the speed and position control on Arduino, Anti-windup	68
<b>6</b>	<b>Conclusion and future applications</b>	<b>70</b>
6.1	Final implementation on the linear axis	70
6.2	Conclusion	73
	<b>Bibliography</b>	<b>74</b>



## List of figures

1.1	Test bench diagram	11
1.2	Test bench mode	12
1.3	DC Motor with built-in Encoder	12
1.4	DC Motor with built-in Encoder	12
1.5	Encoder	13
1.6	Phases of the Encoder	15
1.7	Encoder Minirod 421	15
1.8	Motor Driver H-bridge L298N	17
1.9	Circuit Motor Driver L298N	18
1.10	Arduino UNO	20
2.1	Power versus angular speed in DC motor	24
2.2	Speed versus torque DC motor	25
2.3	Current and velocity versus time	27
2.4	Electrical diagram of a DC motor circuit with torque and rotor angle	28
3.1	PID control scheme	31
3.2	First PID controller	32
3.3	PID controller circuit	33
3.4	Response of the system to the proportional action	34
3.5	Response of the system to the integral action	35
3.6	Response of the system to the integral action	36
3.7	Duty cycle with different PWM (0-255)	39
4.1	Matlab serial communication with Arduino	42





4.2	First step Matlab GUI	43
4.3	First Matlab GUI with start and stop	44
4.4	Initialization of the parameters for speed control	47
4.5	Void setup speed control	48
4.6	Void loop speed control	49
4.7	PID speed control	49
4.8	PWM regulation PID speed control	50
4.9	Communication with Matlab	50
4.10	Parameters and target speed to be sent to Arduino	51
4.11	Graph to visualize the response of the system	51
4.12	Initialization of the parameters for position control	53
4.13	Void loop position control	53
4.14	Arduino code to read the encoder	54
4.15	Initialization of the parameters, Anti-windup	55
4.16	Void loop Anti-windup	56
5.1	Parameters of the step Response	57
5.2	DC Motor identification	58
5.3	Simulink simulation PID speed control	59
5.4	Beginning of oscillation with $K_{p0} = 0.9$	60
5.5	Behaviour at 100 rpm	61
5.6	Behaviour at 50 rpm	62
5.7	Behaviour at 100 rpm	63
5.8	Behaviour at 150 rpm	64
5.9	Behaviour at 210 rpm	65
5.10	Position control on the Arduino serial plotter	66
5.11	Position control, sinusoidal function on the Arduino serial plotter	67
5.12	Data from arduino	68
5.13	Anti WINDUP path	69
5.14	PID speed control with Anti-windup technique	69



6.1	Linear axis test bench	70
6.2	Linear axis test bench	71
6.3	Linear axis test bench	71



## **List of tables**

1.1 Pin specification	14
1.2 Encoder datasheet	16
1.3 Encoder Pin layout	16
1.4 Motor driver L298N pins specification	19
1.5 Arduino UNO features	21
1.6 Functions of digital pins	21
1.7 Functions of others pins	22
5.1 Effects of PID constants on output behavior	58
5.2 Ziegler-Nichols method	66

# 1. Introduction

## 1.1 Didactic purpose

This thesis intends to create a test bench to control a test bench to control a linear axis in position. It explains how the Arduino code must connect and implement the DC motor and how it has started from the Matlab graphical interface. The aim is to realize speed control for the motor and position control for the linear axis.

## 1.2 Hardware

The figure 1.1 shows the scheme. It is composed of the following components:

- DC motor: to operate the bench and to transform an input voltage into rotation speed;
- Encoder: sensor required for speed, to close the control loop;
- L298N H-bridge: allows to electronically control both the speed and the direction of rotation of a DC motor;
- Arduino: Hardware platform consisting of a series of electronic boards equipped with a microcontroller.

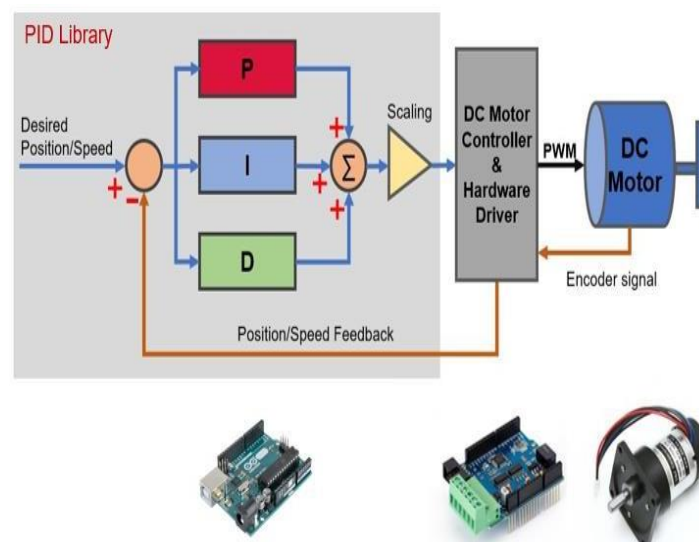


Figure 1.1 Test bench diagram

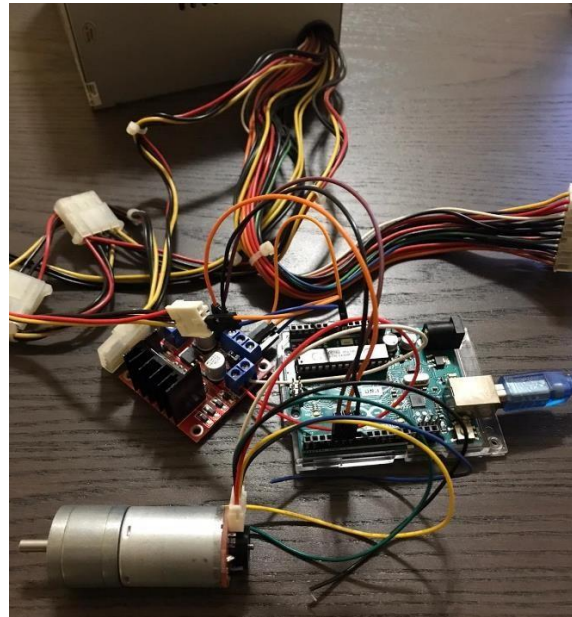


Figure 1.2 test bench made

### 1.2.1 DC motor

The DC motor features large torque, low speed and low noise. The gear motor is primarily designed to reduce the speed in a series of gears, which in turn creates more moment of force. This motor is adopted with pure copper wire coil, low temperature and low loss. The motor is made of metal gears, wear-resistant, in order to prevent the teeth from breaking. By consequence, it has a long service life.



Figure 1.3 DC motor with built-in encoder

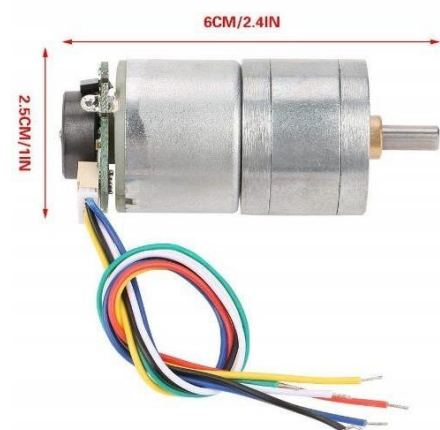


Figure 1.4 DC motor with built-in



Figure 1.5 Encoder

The following table presents the meaning of the cable colours:

RED	Motor + (positive and negative switching can control CW/CCW)
BLACK	Encoder – (voltage range is 3.3-5 [V])
YELLOW	Encoder A phase (The motor turns one turn output, 11 signals)
GREEN	Encoder B phase (The motor turns one turn output, 11 signals)
BLUE	Encoder + (voltage range is 3.3-5 [V])
WHITE	Motor - (positive and negative switching can control CW/CCW)

**Specifications:**

Voltage	DC 12[V]
Speed	230 rpm
Encoder motor end:	11 signals
Rated voltage	DC 12[V]
No-load speed	230 rpm 0.13[A]
Max efficiency	2[Kg][cm]/170 rpm/2[W]/0.6[A]
Max power	5.2[kg][cm]/110rpm/3.1 W/1.10A
Stall torque	10[kg][cm] 3.2[A]
Reduction ratio	1:34
Hall resolution	Hall x ratio 34.02 = 341.2 PPR

Table 1.1 specifications of the DC motor

**1.2.2 Encoder**

In the relative mode, the encoder provides an incremental indication referred to the previous position. This indication is easy to manage as it is sufficient to associate the resolution unit of the incremental encoder to the displacement unit or the precision ratio.

The incremental information consists of a two-bit Gray code (phases A and B) in quadrature (duty cycle 50%). The phase shifts between the two bits in 90 electrical degrees and is necessary to check the direction of rotation. By convention it is possible to assume that the increase is positive when phase B anticipates phase A with rotation of the shaft clockwise, with a view from it. There is also a reference (Zero or Marker) which indicates the completion of the lap. The size of this pulse can be supplied on request from 90 electrical degrees up to some periods (in reference to the duration of the pulses).

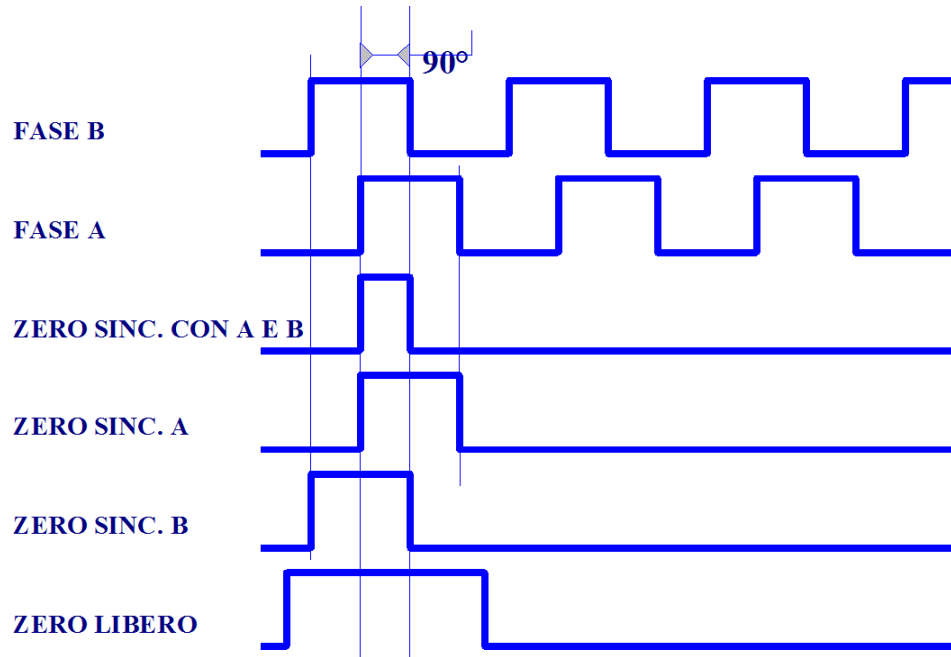


Figure 1.6 Phases of the Encoder

### 1.2.2.1 Encoder Minirod 421

The model used for this work is the 'HEIDENHAIN' 'Minirod 421' which presents the following technical specifications. This is also used as a substitute of the one already applied to the DC motor to have further confirmation.



Figure 1.7 Encoder Minirod 421





<b>Line counts</b>	100/200/250/360/400/500/600/720/900/1000/1024/1080/3600
<b>Accuracy</b>	$\pm 1/20$ grating period
<b>Resolution</b>	0.025° with 3600 lines and 4-fold evaluation in the subsequent electronics
<b>Speed</b>	Max. 10000 rpm
<b>Moment of inertia of rotor</b>	$0.17 \cdot 10^{-6} \text{ kgm}^2$
<b>Torque at 20°C (68°F)</b>	$\leq 0.001 \text{ Nm}$
<b>Shaft load</b>	Axial max 5N, Radial max 10N (at shaft end)
<b>Weight</b>	Approx 0.09kg (0.198lb)
<b>Protection</b>	IP 50 according to IEC 529
<b>Operating temperature and Storage temp.</b>	0° to 70°C -32° to 80°C
<b>Vibration (50 to 2000 Hz)</b>	$\leq 100 \text{ m/s}^2$
<b>Shock (11 ms)</b>	$\leq 300 \text{ m/s}^2$

Table 1.2 Encoder datasheet

### Pin Layout

Pin	1	2	3	4	5	6	7	8	9	10	11	12
Signal	$\overline{U}$	Sensor +5V	$U_{a0}$	$\overline{U}$	$U_{a1}$	$\overline{U}$	free	$U_{a2}$	Shield	0V	Sensor 0V	+5V
Color	Pink	Blue	Red	Black	Brown	Green	/	Gray	/	White/ Green	White	Brown/ Green

Table 1.3 Encoder Pin layout

### 1.2.3 Motor Driver Module-L298N

The L298N is an integrated monolithic circuit in a 15-lead Multiwatt and power SO20 packages. It is a high voltage, high current dual full-bridge driver de-signed to accept standard TTL logic level sand drive inductive loads such as relays, solenoids, DC and stepping motors. Two inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. It is provided an additional supply input in order that the logic works at a lower votage.

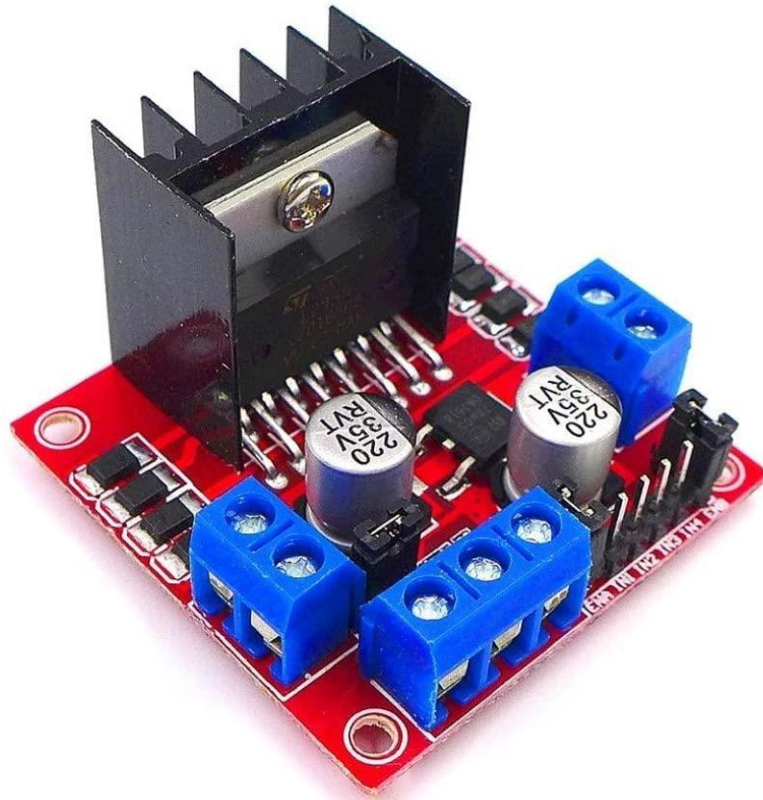


Figure 1.8 Motor driver H-bridge L298N

The figure 1.9 shows the L298N schematic circuit :

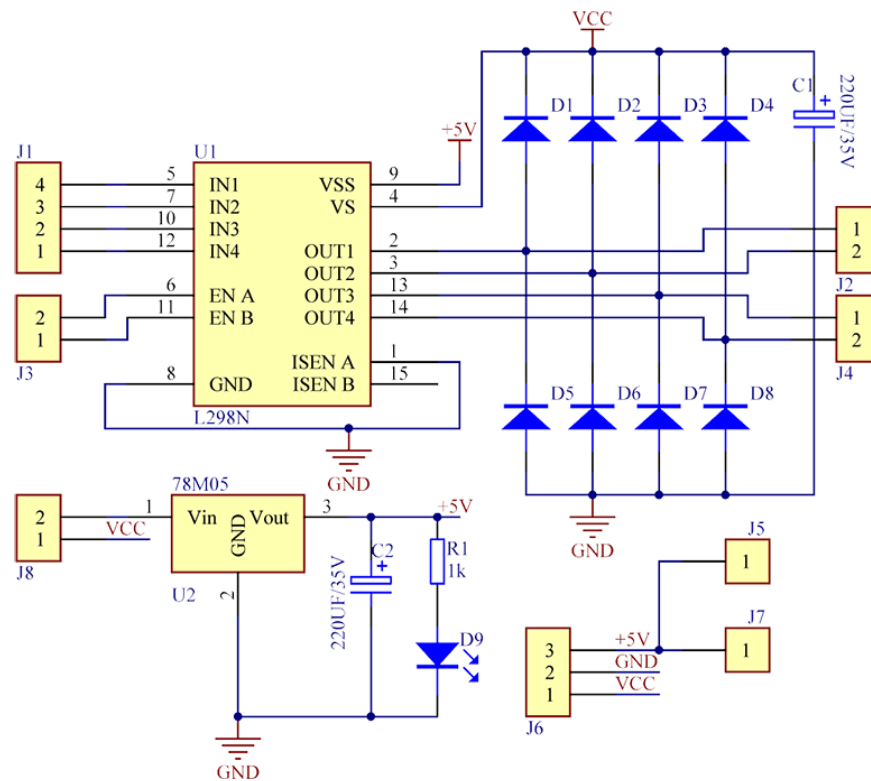


Figure 1.9 Circuit motor driver L298N

### 1.2.3.1 L298N Features & Specifications:

- Driver Model: L298N 2[A];
- Driver Chip: Double H Bridge L298N;
- Motor Suplly Voltage (maximum): 46[V];
- Motor Suplly Current (maximum): 2[A];
- Logic Voltage: 5[V];
- Driver Voltage: 5-35[V];
- Driver Current: 2[A];
- Logical Current: 0-36[mA];
- Maximum Power (W): 25[W];



- Current Sense for each motor;
- Heatsink for better performance;
- Power-On LED indicator.

### 1.2.3.2 L298N Module Pin Configuration

PIN Name	Description
IN 1 & IN 2	Motor A input pins. Used to control the spinning direction of Motor A.
IN 3 & IN 4	Motor B input pins. Used to control the spinning direction of Motor A.
ENA	Enables PWM signals for Motor A.
ENB	Enables PWM signals for Motor B.
OUT1 & OUT2	Output pins of Motor A.
OUT3 & OUT4	Output pins of Motor B.
12 V	12V input from DC power Source.
5 V	Supplies power for the switching logic circuitry inside L298N IC.
GND	Ground pin.

Table 1.4 Motor driver L298N pin specification

### 1.2.4 Arduino

Arduino was born in 2003 in Ivrea, with the idea of developing a low-cost board that could interface with sensors and actuators and is an open source electronic board that includes a microprocessor and various inputs and outputs which can be digital or analog, the number and type depends on the card considerate. These inputs/outputs can be interfaced with cards or other circuits. The card is easily programmable via USB interface from which can be loaded programs written with the appropriate software. The

programming language can be said to be a mixture of C and C++ with functions added and dedicated to the purposes for which the card was created.

### 1.2.4.1 Arduino UNO

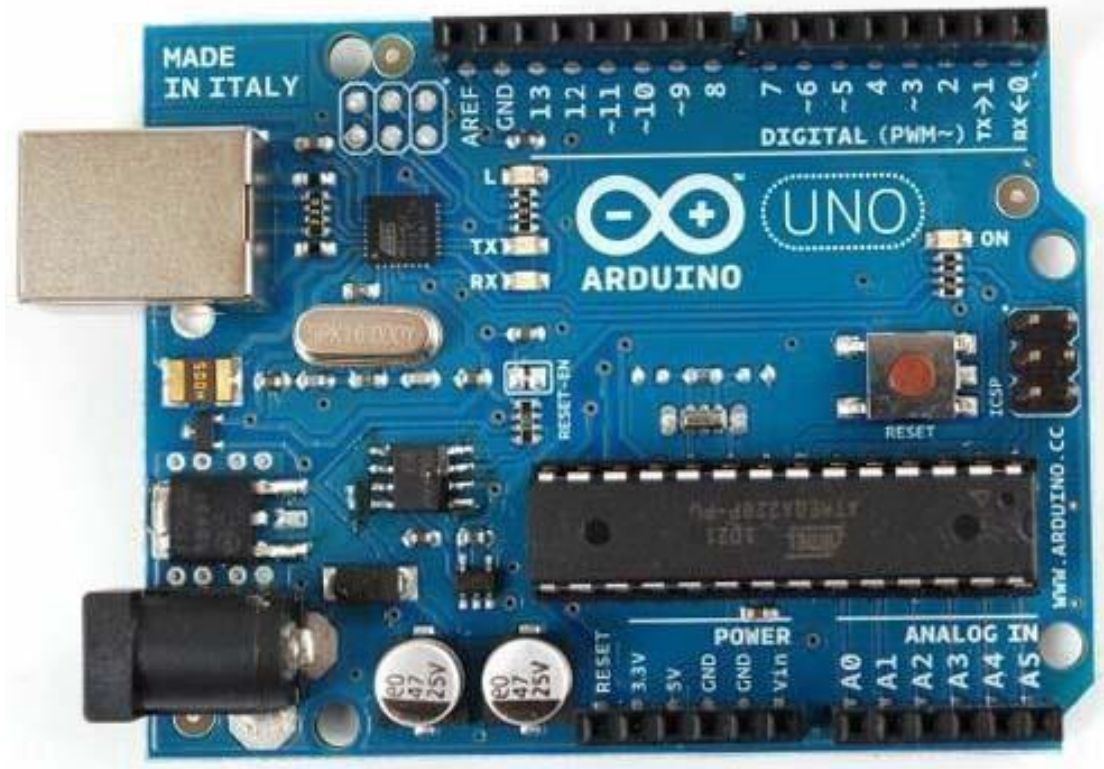


Figure 1.10 Arduino UNO

Arduino Uno is a microcontroller device based that allow you to create different types of electronic circuits. It has 14 programmable digital pins as inputs or outputs (which also have the ability to be used for dedicated functions such as PWM signal generation or UART communication) and 6 input for the acquisition and processing of analog signals. The microcontroller is the ATmega328 produced by Atmel with the following characteristics:

- speed of 16 [MHz]
- 32KB of flash memory
- 2KB sram



- 1KB EEPROM memory.

The board is powered via the USB port or via the appropriate connector. If both the USB cable and the power connector are connected, the card is able to automatically choose the external power source.

The following table lists the Arduino UNO features:

Type of Microcontroller	Atmel ATmega328
Working Voltage	5Vdc
Reccomanded power supply voltage	7V-12V
Digital pins	14 configurable as inputs or outputs
Analog pins	6 entrances
Maximum current per digital pin	40mA max
Flash memory	32KB
Sram memory	2KB
EEPROM memory	1KB
Clock speed of the microcontroller	16MHz

Table 1.5 Arduino UNO features

Description of additional functions of the digital pins:

Pin 0	UART RX
Pin 1	UART TX
Pin 2	External interrupt
Pin 3	External interrupt or PWM
Pin 5	8 bit PWM
Pin 6	8 bit PWM
Pin 9	8 bit PWM
Pin 10	8 bit PWM or SPI(SS)
Pin 11	8 bit PWM or SPI(MOSI)
Pin 12	SPI(MISO)
Pin 12	SPI(SCK)

Table 1.6 Functions of the digital pins



Description of additional functions of the analog pins:

Pin 4	I2C(SDA)
Pin 5	I2C(SCL)

Table 1.7 functions of others pins

## 2. DC motor identification

The equation of the motion for DC motors are the following:

$$V = L \frac{di}{dt} + RI + k_b \theta \quad (2.1)$$

$$J\ddot{\theta} = k_T I - \beta \dot{\theta} - \tau \quad (2.2)$$

Where:

- $V$  is the voltage applied to the motor (from 12V battery); [V]
- $L$  is the motor inductance; [H]
- $I$  is the current through the motor windings; [A]
- $R$  is the motor winding resistance; [ $\Omega$ ]
- $k_b$  is the motor's back electromagnetic force constant; [Vs/rad]
- $\dot{\theta}$  is the rotor's angular velocity; [rad/s]
- $J$  is the rotor's moment of inertia; [Kgm<sup>2</sup>]
- $k_T$  is the motor's torque constant; [Nm/A]
- $\beta$  is the motor's viscous friction constant; [Nm/(rad/s)]
- $\tau$  is the torque applied to the rotor by the load. [Nm]

### 2.1 Equilibrium analysis

When a voltage source is applied to the terminals of the DC motor and a mechanical load is applied to its rotor, a transitory behaviour, lasting a transitory time interval, is followed by a regime behaviour: the angular speeds increments to a stabilized value after a transition time and, at regime, the time derivatives of the current and velocity are null, the regime equilibrium equations are then:

$$V = RI + k_b \dot{\theta} \quad (2.3)$$

$$\tau = k_T I - \beta \dot{\theta} \quad (2.4)$$

It means that:





$$V = \frac{R}{k_T} \tau + \frac{R\beta}{k_T} \theta + k_b \theta \quad (2.5)$$

It is now possible to get the equations of velocity and torque for the equilibrium:

$$\theta = \left( \frac{R\beta}{k_T} + k_b \right)^{-1} \left( V - \frac{R}{k_T} \tau \right) \quad (2.6)$$

$$\tau = V \frac{k_T}{R} - \left( \beta + \frac{k_T k_b}{R} \right) \theta \quad (2.7)$$

From equation 2.6 and 2.7 it is possible to get the equation for the mechanical power  $P$  delivered by the motor.

$$P = \tau \theta \quad (2.8)$$

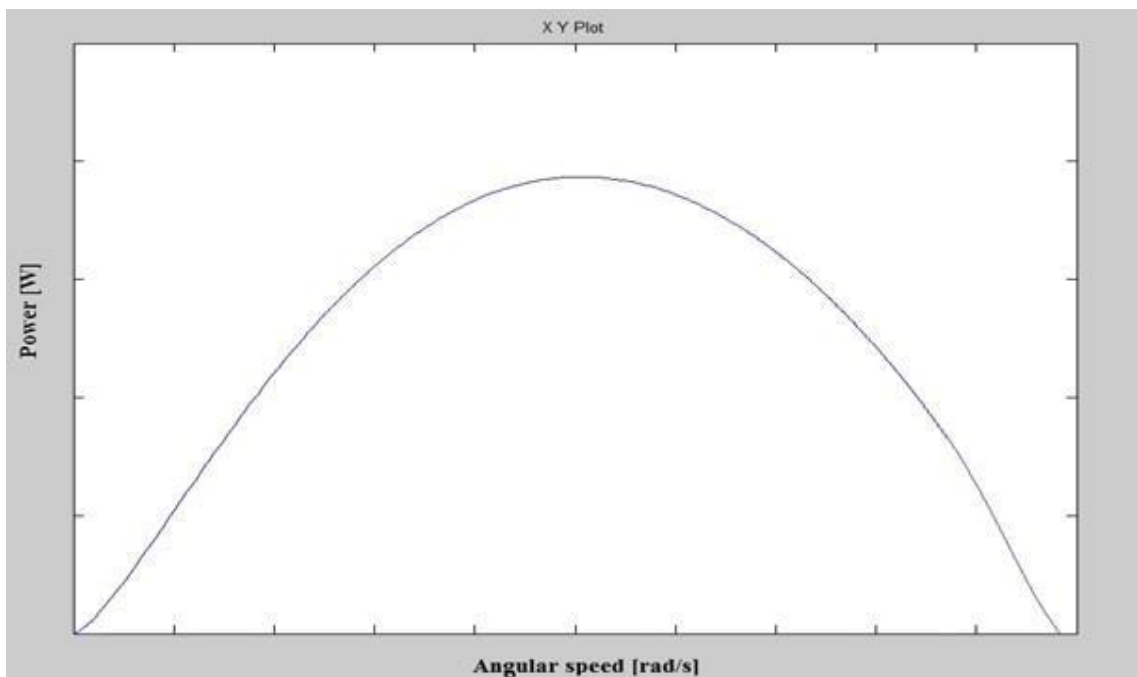


Figure 2.1 Power versus Angular speed in DC motor

When the motor is not braked, it will turn at its maximum speed and it will deliver no mechanical power (left part of the graph). It will deliver the highest amount of mechanical power when the braking is such that the motor turns at one half its maximum speed. In

fig. 2.2 angular velocity, efficiency, current and power curves with respect to the torque for the DC motor running at 12V are displayed.

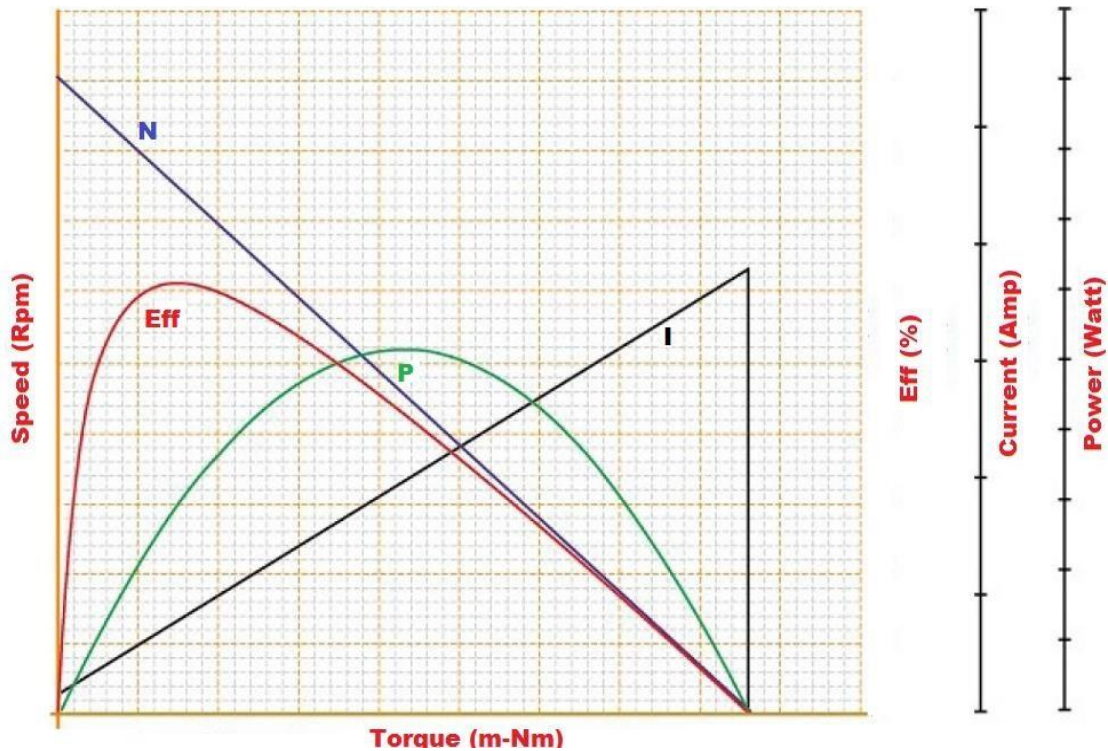


Figure 2.2 (Blue) ÷ Speed with respect to the Torque;

(Green) ÷ Power with respect to the Torque;

(Black) ÷ Current with respect to the Torque;

(Red) ÷ Efficiency with respect to the Torque.

## 2.2 DC motor stall torque and no-load velocity

When the load is such that the DC motor does not move at all, the maximum torque is achieved and this is so called *stall torque*:

$$\tau_s = V \frac{k_T}{R} \quad (2.9)$$

Under stall conditions ( $\theta = 0$ ), since the current is proportional to the torque, it follows that the *stall current* is:

$$I_s = \frac{\tau_s}{k_T} \quad (2.10)$$

It is now possible to compute  $R$  in stall conditions  $R_{\text{stall}}$ , the value of the motor's torque constant  $k_T$ , and the value of the motor's back electromagnetic force  $k_b$ .

$$R_{\text{stall}} = \frac{V}{I_s} \quad (2.11)$$

$$k_T = \tau_s \frac{R_{\text{stall}}}{V} \quad (2.12)$$

$$k_b \cong k_T \quad (2.13)$$

It is now necessary to consider the nominal speed of the DC motor. It can be achieved when no load is applied to the motor shaft and so it is also called *no-load velocity*  $\theta_n$ . The corresponding formulas are reported:

$$\theta_n = V \frac{k_T}{R} \left( \beta + \frac{k_b k_T^{-1}}{R} \right) \quad (2.14)$$

From that equation is it possible to get the value of the motor's viscous friction coefficient:

$$\beta = \frac{k_T}{R} \left( \frac{V}{\theta_n} - k_b \right) \quad (2.15)$$

## 2.3 DC motor efficiency

The efficiency of a motor is defined as the ratio between the output mechanical power and the input electrical power:

$$\eta = \frac{P}{VI} \quad (2.16)$$

To get the efficiency as a function of the velocity, it is possible to express the current and the torque as a function of the velocity as follows:

$$I = \frac{V - k_b \theta}{R} \quad (2.17)$$

$$\tau = \tau_s - \left( \beta + \frac{k_b k_T}{R} \right) \theta \quad (2.18)$$

It is now possible to get the efficiency as:

$$\eta = \frac{\tau \theta}{VI} = \frac{\tau_s \theta - \left( \beta + \frac{k_b k_T}{R} \right) \theta^2}{\frac{V^2}{R} - \frac{V k_{bs} \theta}{R}} \quad (2.19)$$

## 2.4 Transient behaviour

When a voltage is applied to the DC motor starting from a rest condition, the current increases according to the equation 2.1. The current increment is faster than the rotor velocity increment. The time constant with which the current grows is  $L/R$  which is called *electrical time constant*. In general, the electrical time constant is larger than the time constant and so, when analysing the speed dynamics, it is possible to assume the current value as being at regime value. It means that:

$$V \cong RI + k_r \theta \quad (2.20)$$

The above equation means that, generally speaking, it is possible to assume a null electrical time constant. In the following figure, it is possible to understand the differences between using the approximation or not:

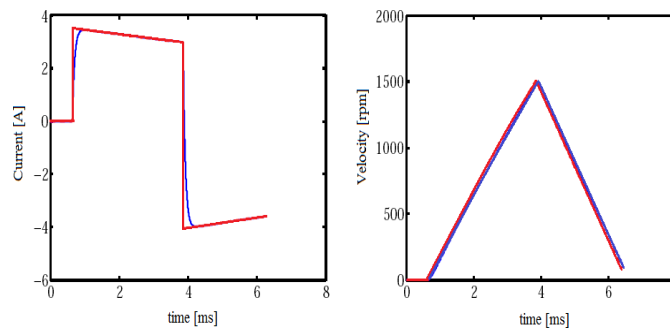


Fig 2.3 Blue: Current and Velocity using the differential equations;

Red: Current and velocity using the zero electrical time constant approximation.

Supposing the DC motor in a stationary condition and applying a voltage  $V$ , the current suddenly assumes a peak value called *start-up* value:

$$I_{\text{start-up}} = \frac{V - k_b \theta}{R} \cong \frac{V}{R} \quad (2.21)$$

After this peak, the current slowly decreases according to the motor velocity. For a given voltage range  $[-V; V]$ , the largest current spike occurs when the motor runs at maximum speed (no load speed), and the voltage is reversed. Because of the large current draws, which may occur when a sudden change in the supply voltage takes place, it is important to have hardwares capable of handling these current spikes. Another approach is to avoid abrupt changes in voltage. For example, to accelerate and decelerate by slowly changing the voltage.

## 2.5 DC motor modelling

For a proper modelling of the DC motor, it is important to know its electrical diagram:

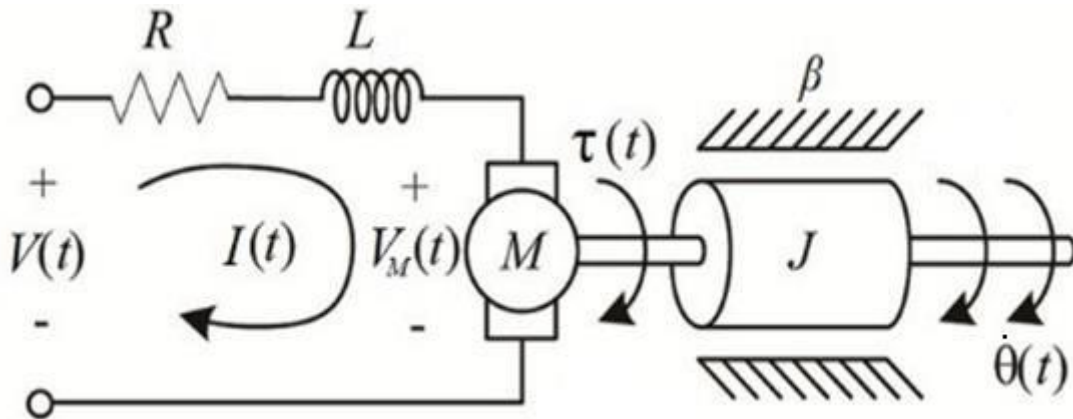


Figure 2.4 Electrical diagram of a DC motor circuit with torque and rotor angle.

The motor torque  $\tau$  is related to the armature current  $I$  by the torque constant  $k_T$ :



$$\tau = I k_T \quad (2.22)$$

The voltage generated on the motor,  $V_M$ , is related to the angular velocity by the following equation:

$$V_M = k_T \frac{d\theta}{dt} \quad (2.23)$$

Basing on figure 4, neglecting electrical and mechanical dissipations, it is possible to write the following equations based on the Newton's law combined with the Kirchhoff's law:

$$J \frac{d^2\theta}{dt^2} + \beta \frac{d\theta}{dt} = k_T I \quad (2.24)$$

$$L \frac{dI}{dt} + RI = V - k_T \frac{d\theta}{dt} \quad (2.25)$$

Using the Laplace transform, the above equations could be written as follows:

$$Js^2\theta(s) + \beta s \theta(s) = k_T I(s) \quad (2.26)$$

$$Ls I(s) + RI(s) = V(s) - k_T s \theta(s) \quad (2.27)$$

From the equation 2.27 it is possible to express  $I(s)$ :

$$I(s) = \frac{V(s) - k_T s \theta(s)}{Ls + R} \quad (2.28)$$

and substitute it into the equation 2.26:

$$Js^2\theta(s) + \beta s \theta(s) = k_T \left( \frac{V(s) - k_T s \theta(s)}{Ls + R} \right) \quad (2.29)$$

From the above equation it is possible to get the expression for  $\theta(s)$ :

$$\theta(s) = \frac{k_T V(s)}{(Ls + R)(Js^2 + \beta s + k_T^2 s)} \quad (2.30)$$

It is now possible to get the transfer function from the input voltage  $V$  to the output angle  $\theta$ :



$$\frac{\theta(s)}{V(s)} = \frac{k_T}{s[(Ls + R)(Js + \beta) + k_T^2]} \quad (2.31)$$

In the same way, considering the equation 2.23 it is possible to get the transfer function from the input voltage  $V$  to the output angular velocity  $\theta$  :

$$\frac{\theta(s)}{V(s)} = \frac{k_T}{(Ls + R)(Js + \beta) + k_T^2} \quad (2.32)$$

The above transfer functions are related to the following block scheme, considering  $k_b \cong k_T$  .

## 3. Types of control

### 3.1 PID control

A PID is a proportional-integrative-derivative control used in control system, both in industrial environments and where continuous and modulated controls are required. The operation of this system is based on the calculation of an error in time which is the difference between the setpoint and the feedback, that is the measured process variable  $e(t) = \text{Set} - \text{Feedback}$ . It is a negative feedback system widely used in control systems. It reacts thanks to an input to any positive or negative errors.

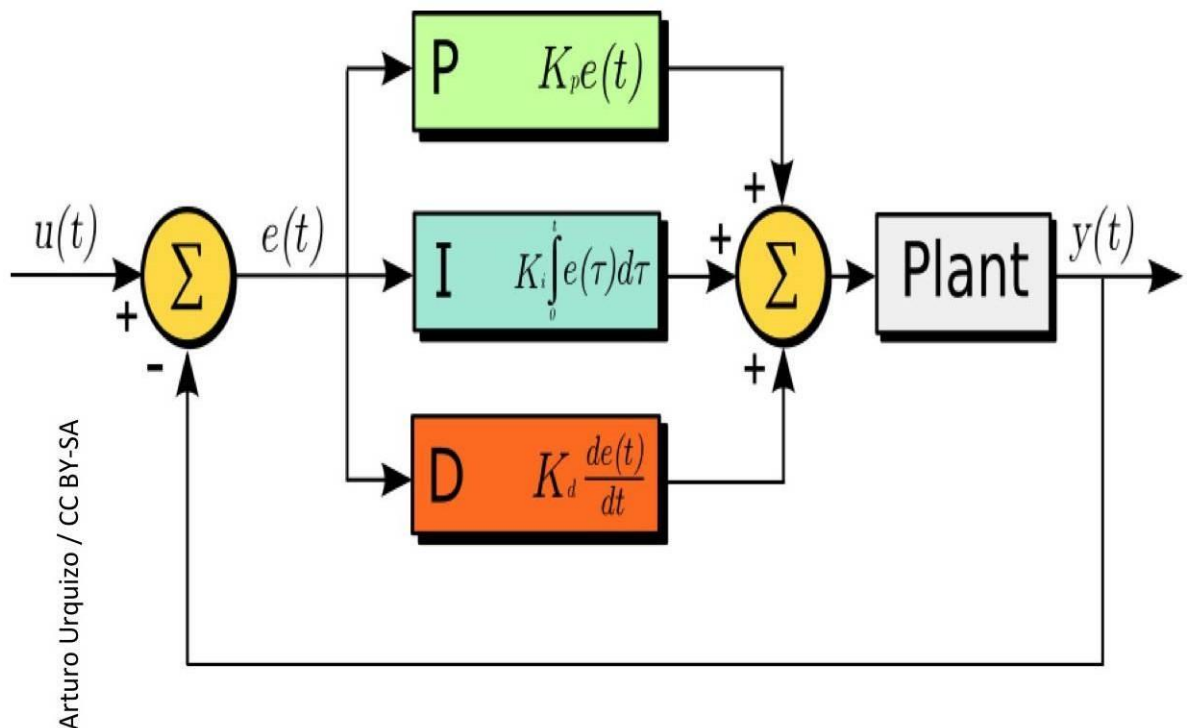


Figure 3.1 PID control scheme

In the figure 3.1 it is possible to see how the process variable  $y(t)$  is subtracted from the setpoint  $r(t)$  to generate the error  $e(t)$ . The controller acquires a value as input and



compares it with a reference value and the difference, the error signal, is used to determine the value of the controller's output variable.

The PID controller adjusts based on:

- proportional action;
- integral action (past values);
- derivative action (how fast the signal varies).

### 3.1.1 Mathematical form

The output can be interpreted as:

$$\begin{aligned} u(t) &= K_p \cdot e(t) + K_i \int_0^t e(t') dt' \\ &\quad + K_d \frac{de(t)}{dt} \end{aligned} \quad (3.1)$$

Where  $K_p$ ,  $K_i$  and  $K_d$  are non-negative values corresponding to the terms proportional, integrative and derivative.

There are other types of control based on the PID such as the PI where the derivative action is null.

### 3.1.2 Origins of the PID

In the image 3.2 it is possible to see one of the first pneumatic PID controllers

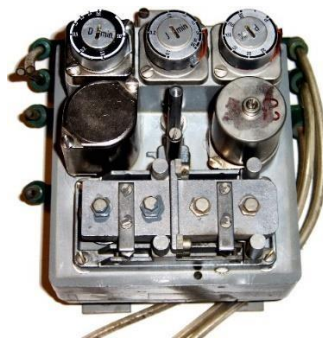


Figure 3.2 first PID controller

With the advent of electronic, PID controllers become cheaper and more accessible, made through Operational Amplifiers. In the fig 3.3, the three operational amplifiers represent the three types of correction.

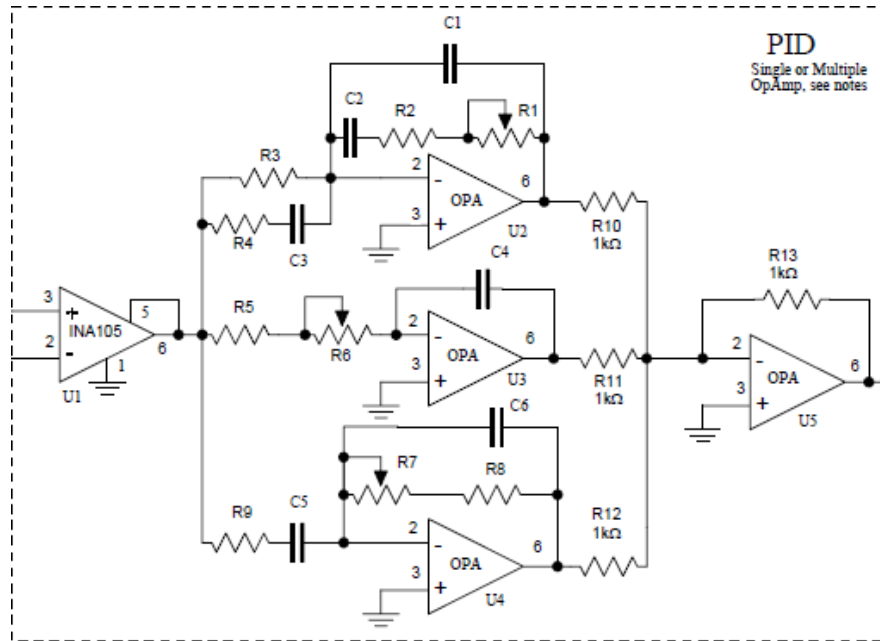


Figure 3.3 PID controller circuit

### 3.1.3 Theory of the PID control

As already mentioned before, the PID has 3 terms, which added together form the output  $u(t)$ . (3.1).

$e(t) = SP - PV(t)$  (3.2) is the error (SP is the setpoint and PV(t) is the process variable, the feedback).

In the Laplace domain, this results in:

$$L(s) = K_p + \frac{K_i}{s} + K_d \cdot s \quad (3.3)$$

Where 's' indicates the Laplace variable.

### 3.1.3.1 Proportional term

The proportional term produces an output value which is proportional to the current value of the error. The proportional contribution can be adjusted multiplying it by the proportional gain  $K_p$ , which is, in fact, constant. There is, therefore,

$$P_{out} = K_p \cdot e(t) \quad (3.4)$$

A high proportional gain results a large variation of the output also as result of a small variation of the error. If the gain proportional is too high, the system can become unstable, viceversa with a small gain proportional, large variations of error and small variations on the output, which make the insensitive contribution to the error. If the proportional gain is too small, the system can be slow to respond to disturbances.

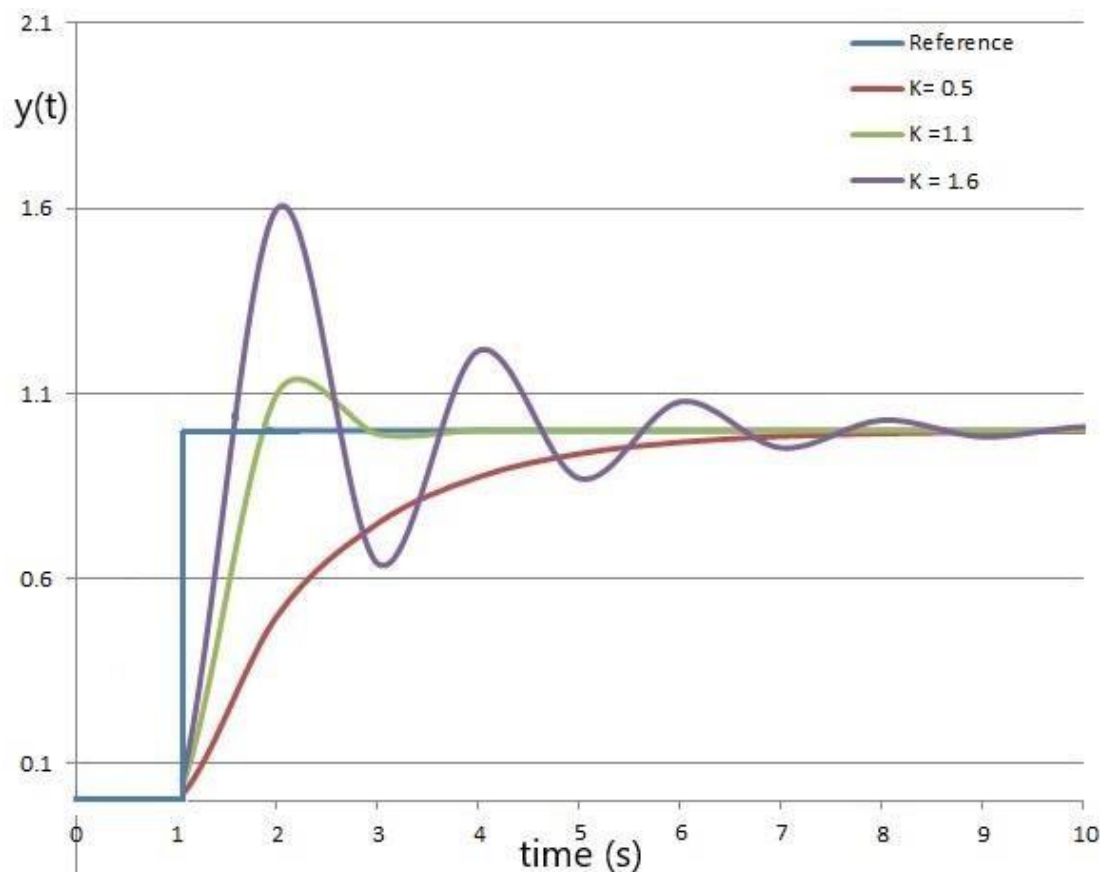


Figure 3.4 response of the system to the proportional action

### 3.1.3.2 Integral term

The contribution of the integral part can amplify the error and its duration, the integral term in PID is the sum of the instantaneous error over time and gives an accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain and the whole comes added to the control output.

To recap:

$$L_{out} = K \cdot \int_0^t e(t) dt \quad (3.5)$$

The integral term is used to eliminate the error steady state residue, which is generally presente with a purely proportional control. It necessary to take into account that even, if the integral term counts the accumulated value of the error from the past, it can cause overshoots around the setpoint value.

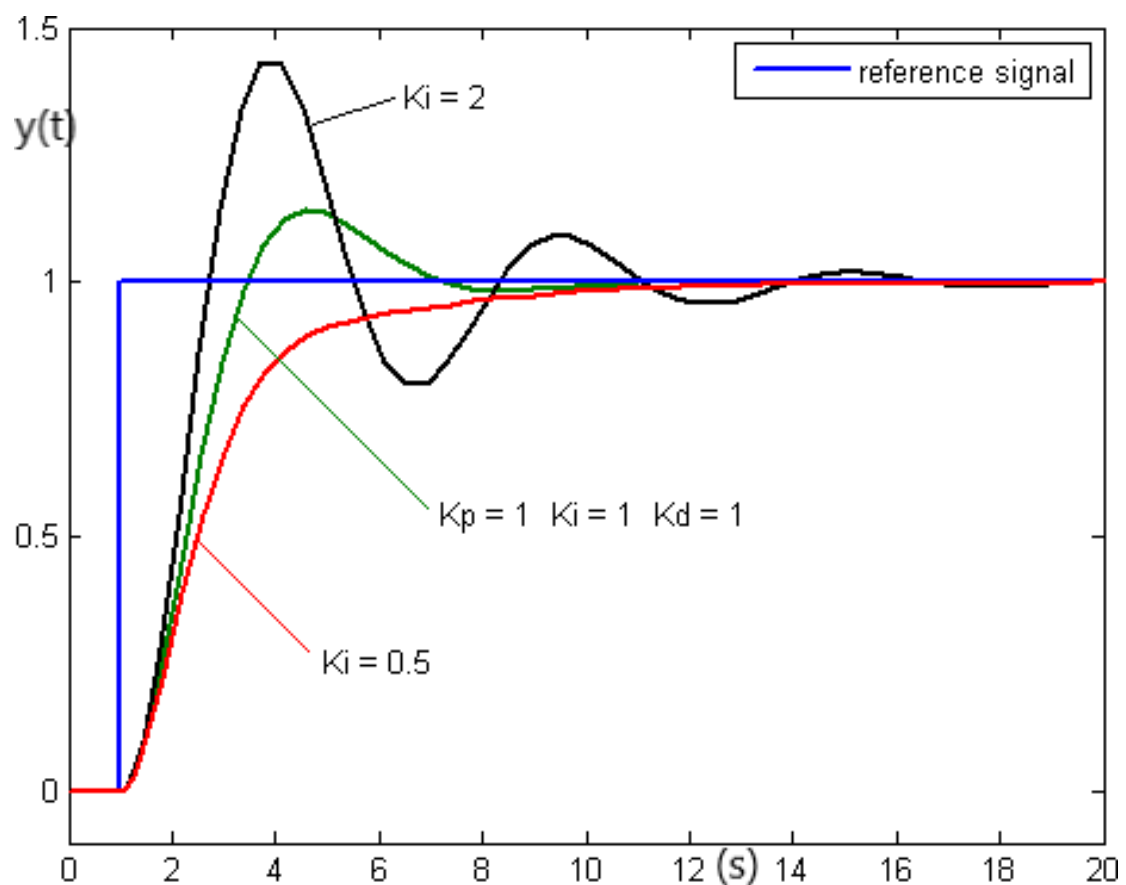


Figure 3.5 Response of the system to the integral action

### 3.1.3.3 Derivative term

The derivative term is calculated by determining the variation of the error over a certain period, this variation is finally multiplied by the gain derivative  $K_d$

$$D_{out} = K_d \cdot \frac{de(t)}{dt} \quad (3.6)$$

This contribution predicts the behavior of the system and increases system stability.

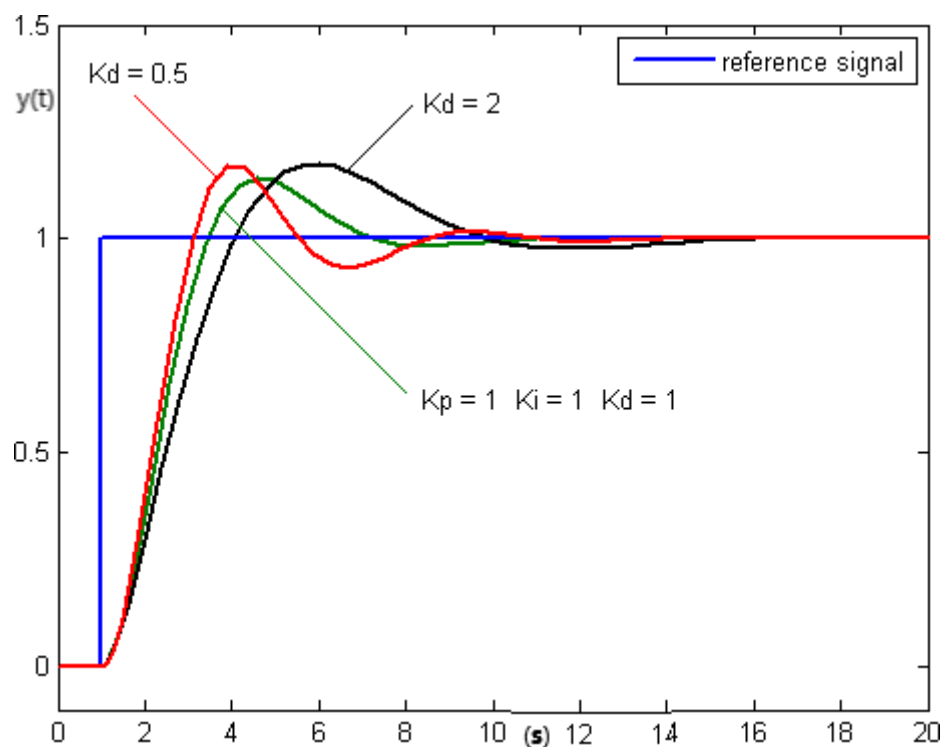


Figure 3.6 Response of the system to the derivative action

### 3.1.4 Loop tuning

With tuning we want to indicate the changes to the control parameters (proportional, integrative and derivative gains) which lead to a system with optimal response. Stability is a basic requirement, but depending on the system, there may be different behaviors and sometimes the requirements they may be in conflict.

Many processes can have degrees of non-linearity and it can happen that in certain fields, the parameters used are fine, while in other fields these tuning values are incorrect and do not maintain the system stability; this can be corrected by doing ‘gain scheduling’ or mapping different parameters in different regions where the system operates.

### 3.1.5 Stability

If the PID parameters ( $K_p$ ,  $K_i$ ,  $K_d$ ) are not correct, the system can be unstable and very often this instability is caused by excessive gain, particularly if the system has delays significant. Theoretically, the system must reach the set value as soon as possible and not oscillate. Mathematically, considering a classic ring, the following transfer function can be obtained:

$$H(s) = \frac{K(s) \cdot G(s)}{1 + K(s) \cdot G(s)} \quad (3.7)$$

Where  $K(s)$  is the transfer function of the PID, while  $G(s)$  is the feedback transfer function. The system is unstable if the closed loop function diverges, this happens when  $K(s) \cdot G(s) = -1$  but typically this happens when  $|K(s) \cdot G(s)| = 1$  with a phase of  $180^\circ$ .

Optimal behavior can be established in two conditions:

- A fixed set is imposed and disturbs the system with an external input, the system has to go back to set as fast as possible and without wobbling. In practice, a system faster to reach the set even if there are small oscillations around it, however, they die down pretty fast rather than having a system that doesn't wobble at all at the set but inevitably remains slower in reaching it;
- A variable set is imposed and notice how fast and stable the system is in following the set.

### 3.1.6 Adjustment methods

There are several methods for tuning a PID, the most used, in general, involves development of models, then the values of  $K_p$ ,  $K_i$  and  $K_d$  are chosen. However, the most

practical way is to do a manual tuning by seeing how the system reacts, but this can lead to setup times long. The choice of method is also influenced by how the system is complex.

- Manual tuning;
- Ziegler-Nichols method;
- ‘Relay’ method;
- Software tools;
- Cohen-coon (only for first order processes);
- Åström-Hägglund.

### **3.1.7 Limitation of PID control**

PID control can be applicable in many situations, in many situations it works well but in others it may have unsatisfactory performance. The main difficulty with simple PID control is to have a controlled system with feedback with constant parameters, therefore there is no direct knowledge of the process that involves delays, compromising performance in certain cases. PID controllers, when used alone, can lead to poor performance if the gains are reduced in such a way that the system does not have overshoot and oscillations around the setpoint. PID controllers have difficulty with the presence of non-linearity of the system, because they always respond in the same way. An improvement can be done through feed-forward with system knowledge, using the PID to check only the error. PID can also be used by doing ‘gain scheduling’ that is, using different parameters in based on the work area and performance. A PID control can also be improved by acting on the reading of the values, on the time of sampling, increasing precision and accuracy or using multiple PID controls in cascade.

## **3.2 PWM theory**

PWM is the acronym for Pulse-Width-Modulation technique used to control digital devices. It is mainly used for the control of electrical devices, such as motors, digital valves etc.

The average value of voltage and current provided is checked by connecting the load through pulses repeated ON and OFF in which the duration is controlled through the ratio of  $T_{on}$  and  $T$ . Ideally, if the frequency is high, the result is to an analog control, but we must consider the application.

The term duty cycle is proportional to the ON period, in fact we have that:  $duty\ cycle = \frac{T_{on}}{T}$ , for example, a low duty cycle corresponds to a low supplied power, because the power supply is on OFF most of the time. The duty cycle is expressed as a percentage, 100% means that the load is always powered.

A square wave is therefore created, given precisely by the continuous change between ON and OFF during the period. By varying the duty cycle, the portion of the ON time with respect to the period of the carrier varies.

With Arduino the duty cycle can be done through the `analogWrite()` function and the values of output voltage are between 0 volts and 5 volts.

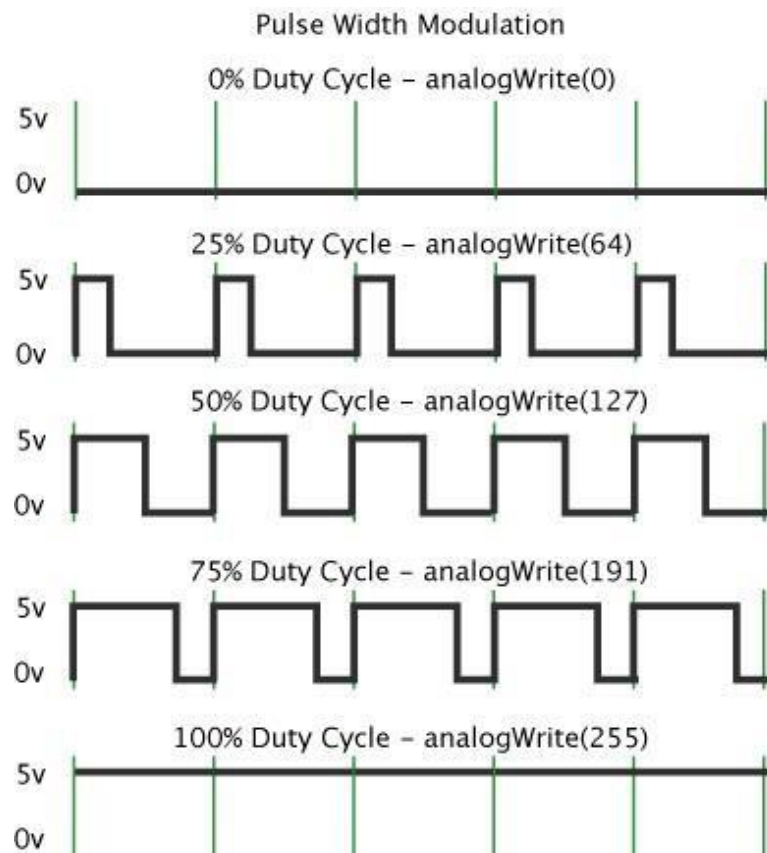


Figure 3.7 Duty cycle with different PWM(0-255)



Considering figure 3.17 it is possible to see that a 50% duty cycle corresponds to a square wave which assume a high value for the 50% of the period; a 75% duty cycle corresponds to a square wave which assume a high value for the 75% of the period and so on. The output square wave is the signal that drives the switch of the power converter. Anyway, this kind of control is not able to run the motor in both directions of rotation. In order to run the motor also in the opposite direction it is necessary to invert the sign of the current which flows in the motor. To invert the sign of the current a circuit called H-bridge realized with 4 switches and 4 recirculation diodes must be implemented (in our case motor driver H-bridge L298N).

### **3.2.1 Time proportioning**

Many digital circuits can generate PWM signals. They normally use a counter that increments periodically and is resetted at the end of every period of the PWM. When the counter value is more than the reference value, the PWM output changes state from high to low (or low to high).

The incremented and periodically reset counter is the discrete version of the intersecting method's sawtooth. The analog comparator of the intersecting method becomes a simple integer comparison between the current counter value and the digital (possibly digitized) reference value. The duty cycle can only be varied in discrete steps, as a function of the counter resolution.

### **3.2.2 PWM sampling theorem**

The process of PWM conversion is non-linear and generally supposed that low pass filter signal recovery is imperfect for PWM. The PWM sampling theorem shows that PWM conversion can be perfect. The theorem states that “Any bandlimited baseband signal within  $\pm 0.637$  can be represented by a pulsewidth modulation (PWM) waveform with unit amplitude. The number of pulses in the waveform is equal to the number of Nyquist samples and the peak constraint is independent of whether the waveform is two-level or three-level”.

- Nyquist-Shannon Sampling Theorem: “If you have a signal that is perfectly band limited to a bandwidth of  $f_0$  the you can collect all the information there is in that signal by sampling it at discrete times, as long as your sample rate is greater than  $2f_0$ .”

### 3.2.3 Applications

Pwm is used to vary the voltage and therefore the power to a generic load. With a duty cycle equal to zero the transferred power is zero, while at 100% the power corresponds to the maximum transferred value if the modulation circuit is not present. Each intermediate value determines a corresponding power supply. The advantage of this technique is to reduce drastically the power dissipated by the limiting circuit compared to the use of analogically controlled transistors and MOSFETs. PWM is also used in efficient voltage regulators. By switching voltage to the load with the appropriate duty cycle, the output will approximate a voltage at the desired level. The switching noise is usually filtered with an inductor and a capacitor. One method measures the output voltage. When it is lower than the desired voltage, it turns on the switch. When the output voltage is above the desired voltage, it turns off the switch.

## 4. Modelling and simulation

The practical part begins by checking the behavior of the motor in open loop, simply by checking it with a simple graphic interface that allows it to rotate clockwise and counterclockwise, stop it and adjust its speed with the PWM theory.

### 4.1 How to interface Matlab and Arduino

The first step was to interface Matlab with Arduino in order to manually control the motor via PC. The Matlab support package for Arduino allows to write Matlab programs that read and write data on the Arduino device and on connected devices such as Adafruit motor shields, I2C and SPI devices. Since Matlab is a high-level interpreted language, programming with it is easier than with C/C++ and other compiled languages. Moreover, it is immediately to see the results from the I/O instructions – no compilation. Matlab includes thousands of built-in math, engineering, and graphing functions to quickly analyze and visualize the data collected by Arduino device.

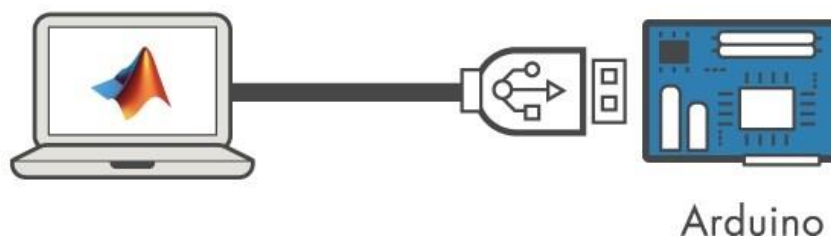


Figure 4.1 Matlab serial connection with Arduino

The graphical command interface was the first application that it was considered appropriate to create.

## 4.2 GUI application creation

The first step to do is the guide instruction in the Matlab command window in order to create a GUI. As a result, a window appears that allows you to create a new project. By choosing to create a new job, it will begin to work in the window shown in the figure 4.2

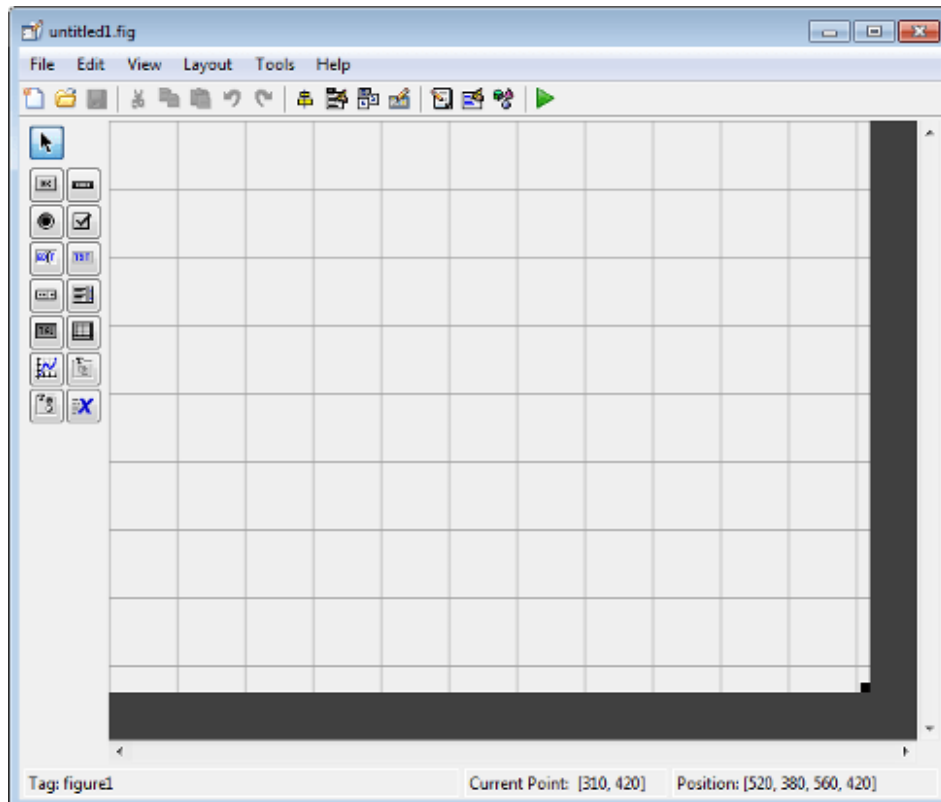


Figure 4.2 first step Matlab GUI

On the left there is a menu from which it is possible to select which components to insert in the graphic interface.

For the first control, the open loop one, the following buttons are necessary:

- Pushbuttons: to choose the type of movement (clockwise or counterclockwise);
- Slider: to increase or decrease the speed;
- Edit text: to view the selected speed.

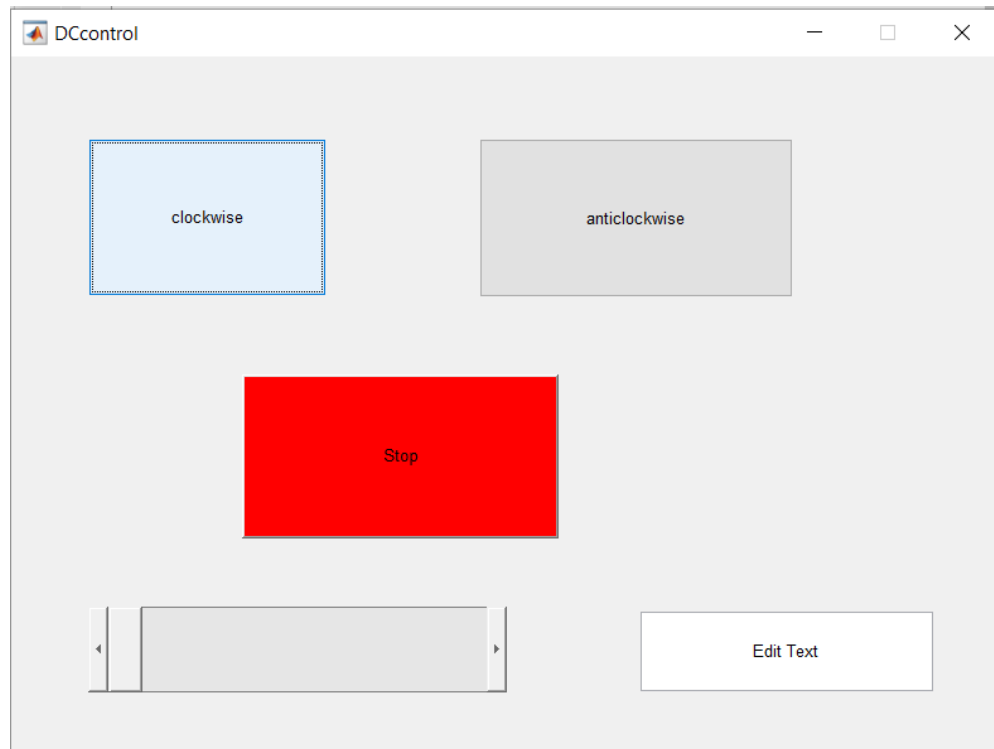


Figure 4.3 first Matlab GUI with start and stop

It is possible to see the matlab code below, automatically generated after the creation of the matlab GUI and implemented to be able to communicate with Arduino. (Comments also generated automatically have been eliminated to make reading more fluid).

```
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton,
                  ...
                  'gui_OpeningFcn',   @DCcontrol_OpeningFcn, ...
                  'gui_OutputFcn',    @DCcontrol_OutputFcn, ...
                  'gui_LayoutFcn',    [] , ...
                  'gui_Callback',     []);
```



```
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before DCcontrol is made visible.
function DCcontrol_OpeningFcn(hObject, eventdata,
handles, varargin)

% Choose default command line output for DCcontrol
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

function varargout = DCcontrol_OutputFcn(hObject,
eventdata, handles)

% Get default command line output from handles
structure
varargout{1} = handles.output;
clear all;
global a;
a = arduino('COM3');
a.pinMode(6, 'output');
a.pinMode(7, 'output');

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata,
handles)
global a;
a.digitalWrite(6,0);
a.digitalWrite(7,1);

% --- Executes on button press in pushbutton2.
```



```
function pushbutton2_Callback(hObject, eventdata, handles)
global a;
a.digitalWrite(6,1);
a.digitalWrite(7,0);

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
global a;
a.digitalWrite(6,0);
a.digitalWrite(7,0);

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
global a;
slider = get(hObject, 'Value')
slider1 = slider*20;
set(handles.edit1, 'String', num2str(slider1));
a.digitalWritePWMVoltage(8, slider);
guidata(hObject, handles);
function slider1_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', [.9 .9 .9]);
end

function edit1_Callback(hObject, eventdata, handles)
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function edit1_Callback(hObject, eventdata, handles)
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end
```

## 4.3 PID speed control and final graphic interface

The PID control of this test bench is summarized in the Arduino and Matlab GUI codes (4.4 and 4.5). The closed loop speed control adds the value obtained with the control open loop with the result achieved from the PID control. The control part allows us to compensate for disturbances.

### 4.3.1 Arduino Code

```
String mySt = "";
char myChar;
boolean string = false; // whether the string is complete
boolean motor_start = false;
const byte pin_a = 2; //encoder pulse A
const byte pin_b = 3; //encoder pulse B
const byte pin_fwd = 4; //run motor fw H-bridge L298N
const byte pin_bwd = 5; //run motor bw H-bridge L298N
const byte pin_pwm = 6; //motor speed in H-bridge L298N, EnA to regulate the PWM
int encoder = 0; //encoder start to count
int m_direction = 0;
int sv_speed = 100; //PWM (0~255)
double pv_speed = 0;
double set_speed = 0;
double e_speed = 0; //error= set_speed - pv_speed
double e_speed_pre = 0; //precedent error of speed
double final_error = 0; //final error
double pwm_pulse = 0;
double kp = 0;
double ki = 0;
double kd = 0;
int timer1_counter;
int i=0;
```

Figure 4.4 initialization of the parameters for speed control

In the initialization part of the variables, a string and a character are defined. They allow us to read the values sent by Matlab. Arduino digital pins, 2 and 3 to read the encoder, 4 5 and 6 to communicate with the H-bridge. Lastly, the PID parameters, errors and constants, initialized to zero are defined.





```
void setup()
{
    pinMode(pin_a, INPUT_PULLUP);
    pinMode(pin_b, INPUT_PULLUP);
    delay(100);

    pinMode(pin_fwd, OUTPUT);
    pinMode(pin_bwd, OUTPUT);
    pinMode(pin_pwm, OUTPUT);
    delay(100);

    attachInterrupt(digitalPinToInterrupt(pin_a), set_point, RISING);
    delay(100);

    noInterrupts();
    TCCR1A = 0;
    TCCR1B = 0;
    TCNT1 = 59000;

    TCCR1B |= (1 << CS12);
    TIMSK1 |= (1 << TOIE1);
    interrupts();

    |
    analogWrite(pin_pwm, 0);
    digitalWrite(pin_fwd, 0);
    digitalWrite(pin_bwd, 0);

}
```

Figure 4.5 Void setup() speed control

In the Void setup(), all the necessary informations are given before the execution of the program.

In particular:

- *pin\_a* and *pin\_b* are INPUT\_PULLUP to force the input pin to be HIGH by default, it sets the pin as an Input and also activates the internal resistor, which keeps the pin at HIGH level;
- *pin\_fwd*, *pin\_bwd* and *pin\_PWM* are OUTPUT, this means that a digital signal can go out from these pins and therefore can be high or low.
- *TCCR1A*: is timer/counter 1 control register A;

- **TCCR1B**: is timer/counter 1 control register B;
- **TCNT1**: is timer/counter 1's counter value;
- **CS12**: is the 3<sup>rd</sup> clock select bit for timer/counter 1;
- **TIMSK1**: is timer/counter 1's interrupt mask register;
- **TOIE1**: is timer/counter 1 overflow interrupt enable;
- Initialize Motor speed and position.
- **Attach interrupt: Rising**, the interrupt is executed when passing from a LOW level to a HIGH level.

### ***The void loop:***

```
if (stringComplete) {  
    // clear the string when COM receiving is completed  
    mySt = "";  
    stringComplete = false;  
}  
  
if (motor_check == true) {  
    digitalWrite(pin_fwd, HIGH);    //run motor fwd or bwd invert pin  
    digitalWrite(pin_bwd, LOW);  
    motor_start = true;  
}  
  
else if (motor_check == false) {  
    digitalWrite(pin_fwd, LOW);  
    digitalWrite(pin_bwd, LOW);    //stop motor  
    motor_start = false;  
}
```

Figure 4.6 Void loop() speed control

In the void loop the cleanliness of the string is checked and if the motor is on or off according to the matlab GUI.

### ***PID control:***

```
e_speed = set_speed - e_speed_pre; //error  
kp_corr = kp * error;  
  
accum_error += error;  
ki_corr = ki * accum_error;  
  
slope = error - lastError;  
kd_corr = kd * slope;  
lastError = error;  
  
corrfinal = kp_corr + ki_corr + kd_corr;
```

Figure 4.7 speed PID control

The errors are calculated and thanks to the action of the proportional, integrative and derivative constants, the stability of the system is obtained. When the motor is stopped the variables are reset to 0.

***PWM regulation:***

```
if (pwm_pulse <255 & pwm_pulse >0){  
    analogWrite(pin_pwm,pwm_pulse); //set motor speed  
}  
else{  
    if (pwm_pulse>255){  
        analogWrite(pin_pwm,255);  
    }  
    else{  
        analogWrite(pin_pwm,0);  
    }  
}
```

Figure 4.8 PWM regulation PID speed control

If the pulsations exceed 255 the speed is set to maximum otherwise to 0

***Communication with Matlab:***

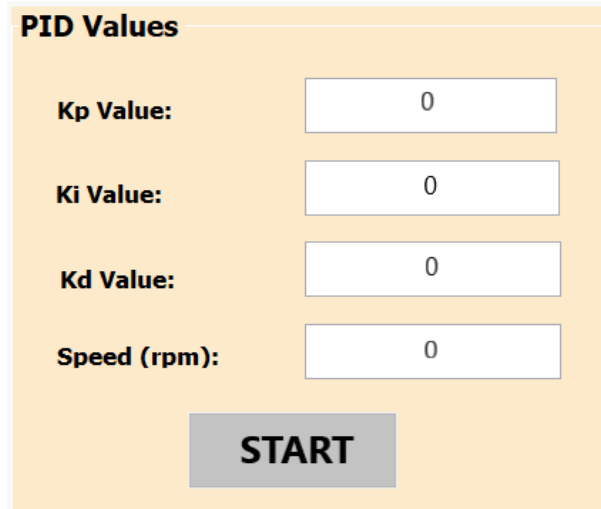
```
void serialtransmission()  
{  
    motor_check = Serial.parseInt();  
    set_speed = Serial.parseInt();  
    kp = Serial.parseFloat();  
    ki = Serial.parseFloat();  
    kd = Serial.parseFloat();  
}
```

Figure 4.9 Communication with Matlab

The values sent by the Matlab GUI are received.

### 4.3.2 Matlab GUI

The graphical interface of Matlab consists of the setting of the PID parameters and the choice of the target speed (fig 4.10)



The figure shows a Matlab GUI window titled "PID Values". It contains four input fields for parameters: "Kp Value:", "Ki Value:", "Kd Value:", and "Speed (rpm):". Each field currently displays the value "0". Below these fields is a large grey button labeled "START".

Figure 4.10 Parameters and target speed to be sent to Arduino

In detail it is composed of:

- Four edit text: allow us to write the set value which will be read as a string and then transformed into a number;
- 'START' pushbutton: allow us to send values to Arduino.

The behavior of the system is displayed in this graph 4.11 , with the speed versus time, always created on the Matlab GUI:

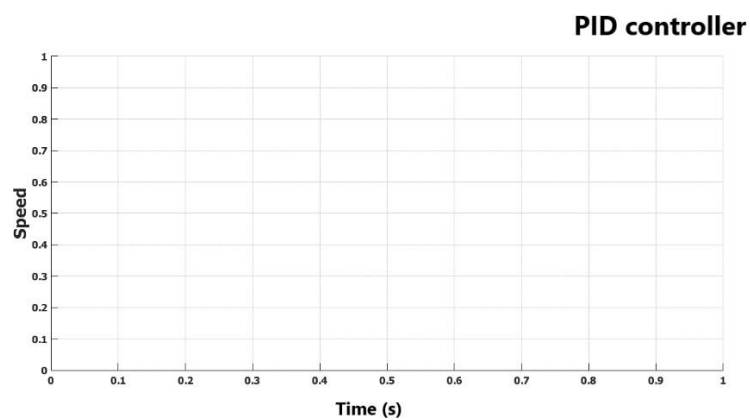


Figure 4.11 Graph to visualize the response of the system



After creating the matlab GUI, the code is implemented to make it possible to communicate with Arduino. In particular:

```
handles.s = serial('COM3');  
set(handles.s, 'BaudRate', 9600);  
set(handles.s, 'DataBits', 8);  
set(handles.s, 'StopBits', 1);  
fopen(handles.s);  
  
guidata(hObject, handles);  
return;
```

- Initialization of the serial port;
- Communication speed;
- Data bits;
- Stop bit.

## **4.4 PID position control**

Arduino simplifies the position control. Indeed, it tries it both with a fixed position and with a variable position over time, for example using sin and cos functions. The PID position control was then implemented with a graphical user interface, it is designed only as a test so that it can be implemented with the linear axis.



```
#define pin_a 2 //enc1
#define pin_b 3 //enc2
#define pin_fw //input H-bridge 1
#define pin_bw 5 //input H-bridge 2
#define pin_pwm 6

int actualposition = 0;
long T = 0;
float err = 0;
float errintegral = 0;

void setup() {
    Serial.begin(9600);
    pinMode(pin_a, INPUT);
    pinMode(pin_b, INPUT);
    attachInterrupt(digitalPinToInterrupt(ENCA), readEncoder, RISING);
    Serial.println("TARGET POSITION");
}
```

#### 4.12 Initialization of the parameters for position control

As for speed control, errors and pins are defined. Subsequently, the communication speed of the serial port and the characteristics of the pins are determined in the void setup().

##### *In the void Loop:*

The initialized variables are used to save the values between the time phases. Thus, they are able to adjust the control signal. The error is calculated by subtracting the current position from the target position. The maximum value of PWM=255 means that we are at 100% of the duty cycle.

```
// target position
int target = 100; //for example
//initialization of the constants
float kp = 0.01;
float kd = 0.2;
float ki = 0.001;
// time difference
long currT = micros();
float deltaT = ((float) (currT - prevT)) / (1.0e6);
prevT = currT;
// error
int e = actualpos - target;
// derivative
float dedt = (e - eprev) / (deltaT);
// integral
eintegral = eintegral + e * deltaT;
// control signal
float u = kp * e + kd * dedt + ki * eintegral;
// motor power
float pwr = fabs(u);
if (pwr > 255) {
    pwr = 255;
}
// motor direction
int dir = 1;
if (u < 0) {
    dir = -1;
}
```

#### 4.13 Void loop() position control

The values of the controller constants are those entered for the final test, which leads to the stability of the system. You have the operations for the control and regulation of the maximum power (regulated thanks to the PWM output and the H-bridge) by following the reading of the code.

### ***Encoder reading:***

In the following figure, you have the final piece of the code. Here, you can see the function that controls the direction of the motor. Moreover, the figure shows if it is OFF. The increase or decrease o

```
void setMotor(int dir, int pwmVal, int pwm, int in1, int in2){
    analogWrite(pwm,pwmVal);
    if(dir == 1){
        digitalWrite(pin_a,HIGH);
        digitalWrite(pin_b,LOW);
    }
    else if(dir == -1){
        digitalWrite(pin_a,LOW);
        digitalWrite(pin_b,HIGH);
    }
    else{
        digitalWrite(pin_fw,LOW);
        digitalWrite(pin_bw,LOW);
    }
}

void readEncoder(){
    int REG = digitalRead(pin_b);
    if(REG > 0){
        pos++;
    }
    else{
        pos--;
    }
}
```

#### 4.14 Arduino code to read the encoder

## 4.5 Anti-Windup, speed and position control.

When using a regulator with integral action, it is possible that the controller output reaches these limits; in this case the action of the actuator cannot grow, even if the error of regulation is not null. Consequently, the integral term continues to grow, but this increase does not produce no effect on the plant control variable. This situation, in addition to not making the regulator work properly, makes the regulator inactive even when the error decreases or reverses its sign; in fact, the regulation system can reactivate only when the  $u(t)$  signal falls within the linearity zone of the actuator characteristic (discharge of the integral term). This phenomenon is called integral wind-up.

The code starts with the same initialization of the pins of the controls seen previously. A classic control that increases or decreases a counter in order to check if the motor rotates clockwise or anticlockwise.

```
unsigned long t;
unsigned long t_prev = 0;

const byte pin_a = 2;
const byte pin_b = 3;
const byte pin_PWM = 6;
const byte In2 = 5;
const byte In1 = 4;
volatile long EncoderCount = 0;

float kp = 0.01;
float ki = 0.2;
float kd = 0.001;

volatile unsigned long count = 0;
unsigned long count_prev = 0;

float Theta, RPM, RPM_d;
float Theta_prev = 0;
int dt;
float RPM_max = 230;

float Vmax = 6;
float Vmin = -6;
float V = 0.1;
float e, e_prev = 0, inte, inte_prev = 0;
```

Figure 4.15 inizialization of the parameters Anti-windup



The key part of the code is this with the trajectory that the system must follow, we calculate the actual angular speed and the position. We also define the trajectory, minimize the error using the PID and the wind-up formulation.

```
void loop() {
  if (count > count_prev) {
    t = millis();
    Theta = EncoderCount / 900.0;
    dt = (t - t_prev);
    RPM_d = RPM_max * (sin(2 * pi * 0.005 * t / 1000.0)) * sign(sin(2 * pi * 0.05 * t / 1000.0));
    if (t / 1000.0 > 100) {
      RPM_d = 0;
    }
    RPM = (Theta - Theta_prev) / (dt / 1000.0) * 60;
    e = RPM_d - RPM;
    inte = inte_prev + (dt * (e + e_prev) / 2);
    V = kp * e + ki * inte + (kd * (e - e_prev) / dt);
    if (V > Vmax) {
      V = Vmax;
      inte = inte_prev;
    }
    if (V < Vmin) {
      V = Vmin;
      inte = inte_prev;
    }
  }
}
```

#### 4.16 void loop Anti-windup

The encoder function measure rpm and position of the motor. The power supply is finally set to 12[V].

## 5. Model validation and results

In this part, the key points of system behavior are analyzed in fig. 5.1:

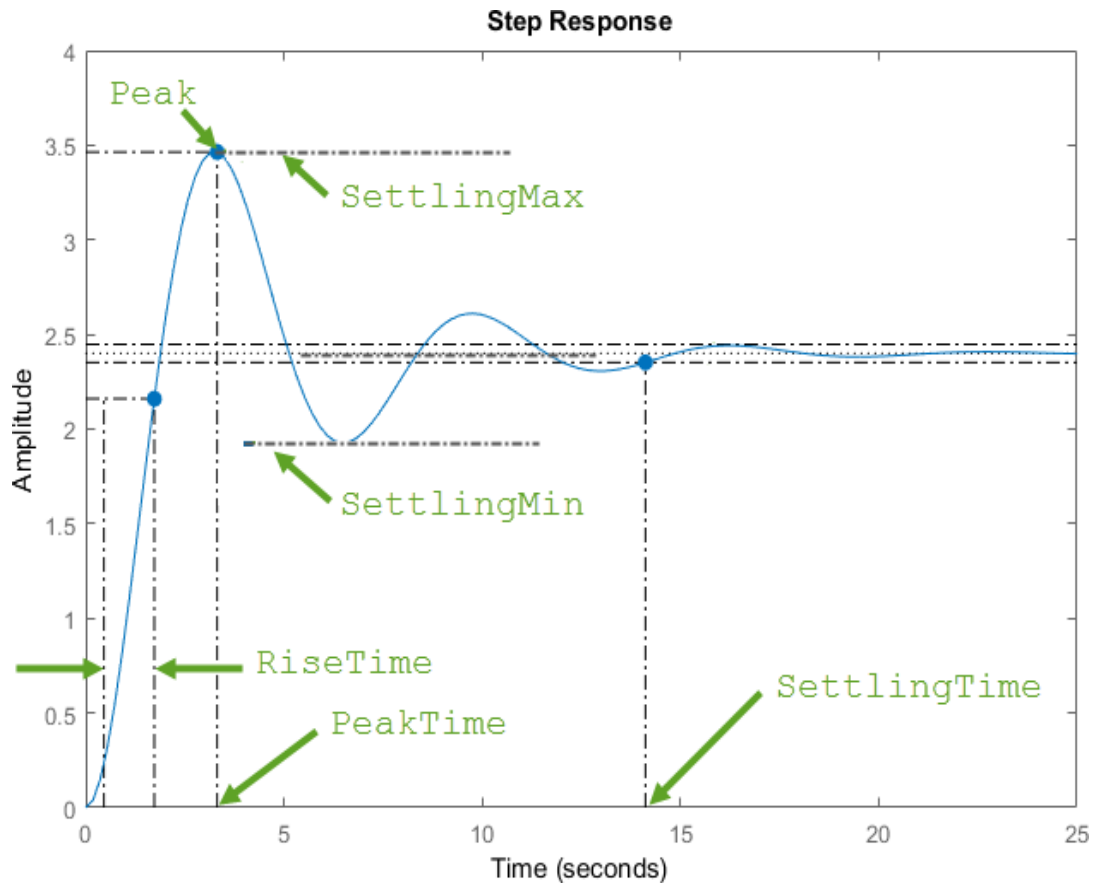


Figure 5.1 parameters of the step Response

- **Average speed:** the average speed maintained during the period considered;
- **Peak value:** the highest speed value during the period considered;
- **Rise time:** the time taken by a signal to change from a specified low value to a specified high value.

- **Settling time:** the time elapsed from the application of an ideal instantaneous step input to the time which the amplifier output has entered and remained within a specified error band.
- **Overshoot:** the occurrence of a signal or function exceeding its target.
- **Steady-state error:** the difference between the desired value and the actual value.

The constants of the PID controllers affect the behavior of the system in this way:

PID	Rise Time	Overshoot	Settling Time	Steady-state err
$K_p$	decrease	increase	Small change	decrease
$K_i$	decrease	increase	increase	eliminate
$K_d$	Small-change	decrease	decrease	No change

Table 5.1 effects of PID constants on output behavior

## 5.1 DC Motor identification

To identify the motor, tests were carried out to measure the I/O behaviour voltage and current:

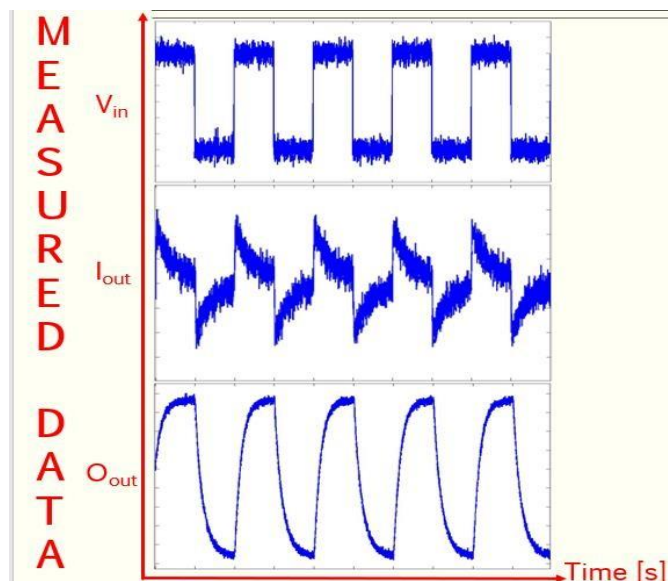


Fig 5.2 DC Motor Identification

## 5.2 Experimental results of speed control with matlab graphic interface

In this formulation, the PID part allows to compensate the disturbances. However, it is linked to the accuracy of the encoder. The system is powered by an external 12V power supply

Controller constants are calibrated with the Ziegler-Nichols method wich consists of:

- The loop is closed by inserting a constant of low proportionality until the system begins to exhibit stable oscillations, so as to close the loop;
- The chosen value of  $K_p$  is what allows us to trigger a permanent oscillation;

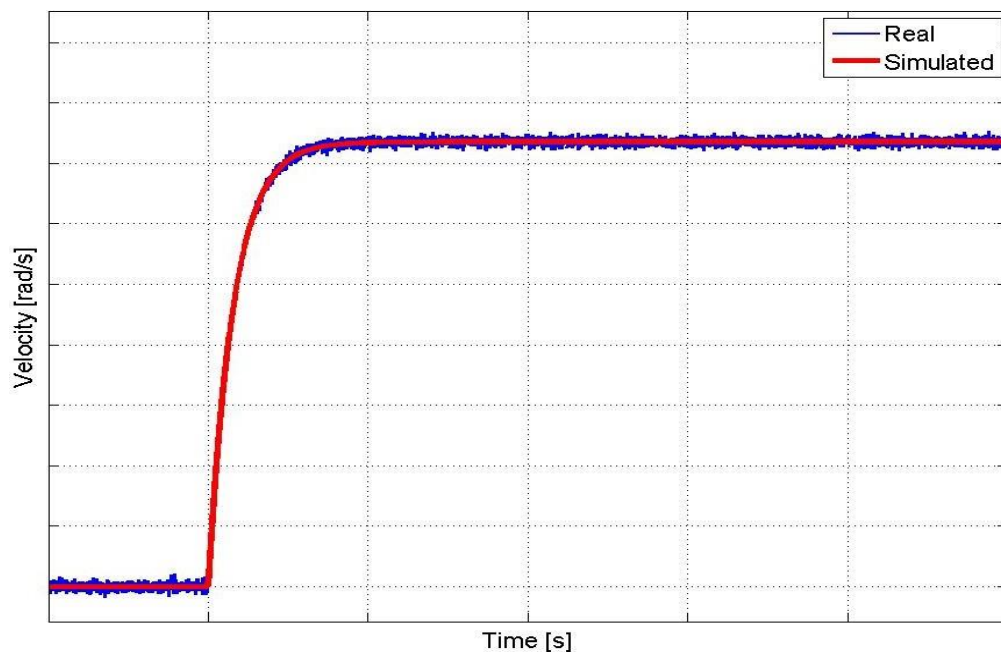


Fig. 5.3 Simulink simulation PID speed control

The first value of  $K_p$  that allows us to see stable oscillations is  $K_p = 0.9$ .

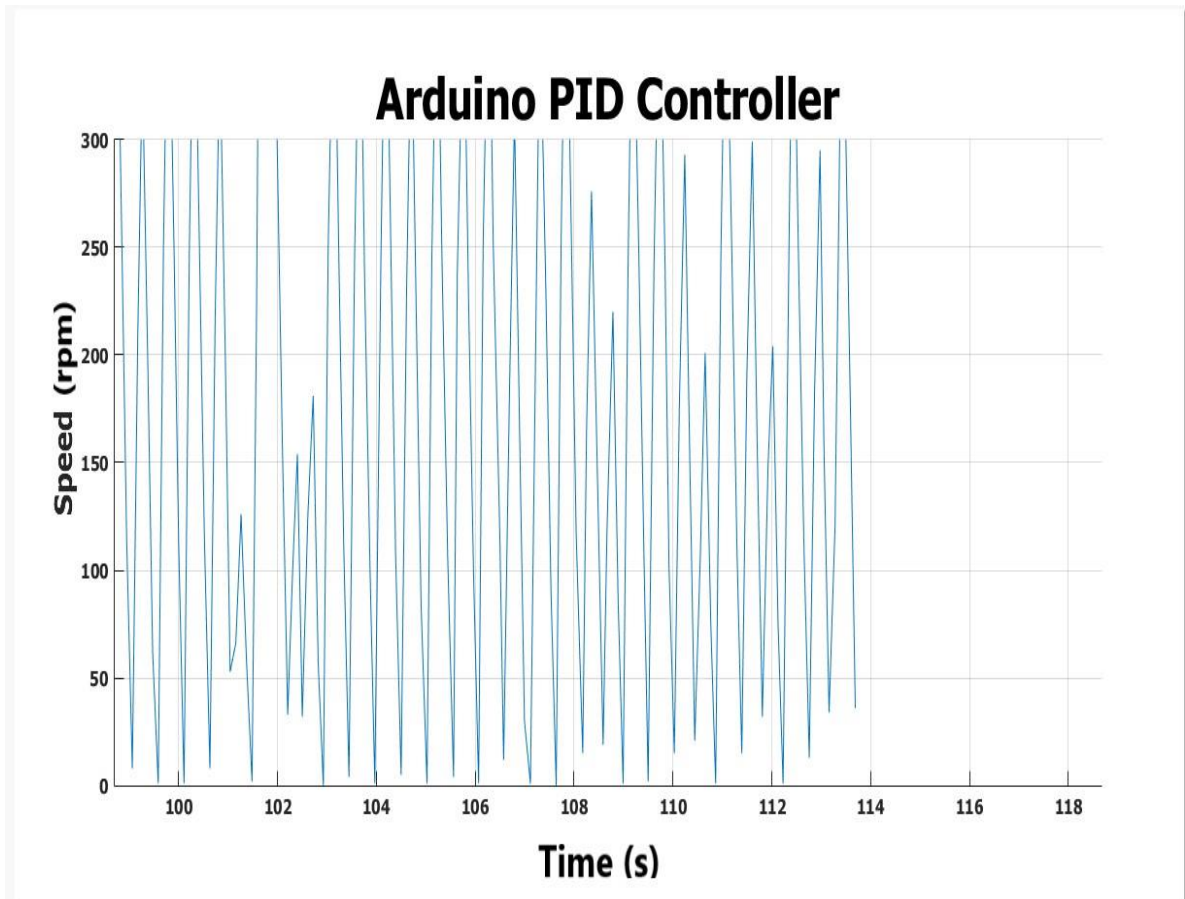


Figure 5.4 Beginning of oscillation with  $K_{p0} = 0.9$

- The chosen  $K_p$  becomes the reference  $K_{p0}$ ;
- The period of oscillation  $T_0$  is measured;
- $K_i$  is increased until maximum performance is reached;

Experimentally the following results are found:

- $K_p = 0.48$ ;
- $K_i = 9.6$ ;
- $K_d = 0.06$ ;
- $T_0 = 0.2$  [ms];

Inserting these values in the Matlab GUI the system response is that of the figure 5.5

## PID controller

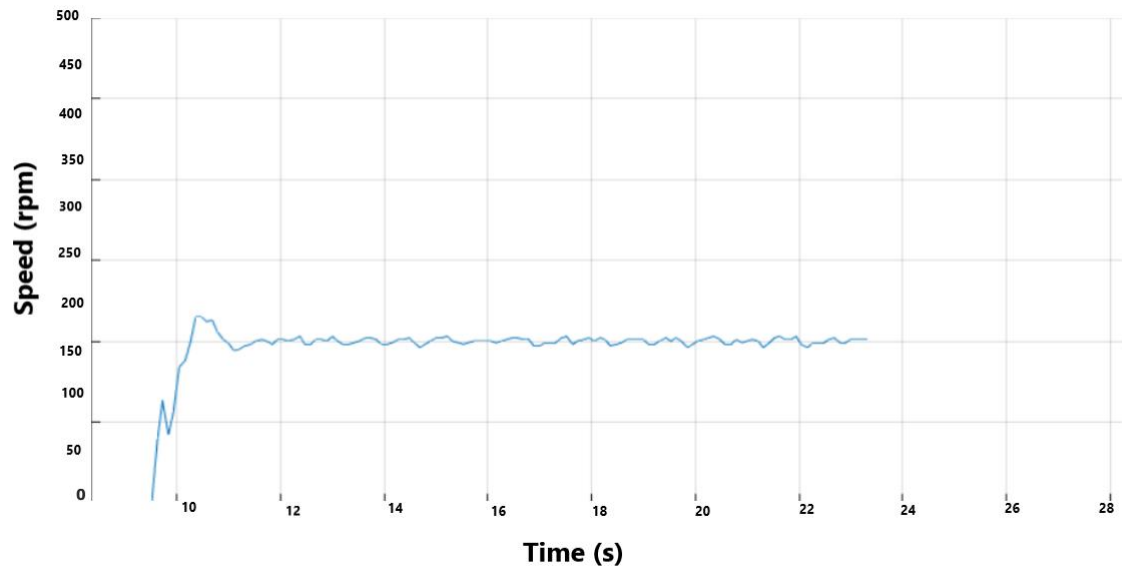


Figure 5.5 behaviour at 150 rpm

- Peak value: 171 rpm;
- Rise time: 0.93 [s];
- Settling time: 3.41 [s];
- Overshoot: 14%;
- Steady-state error: 2.4%.

### 5.3 Experimental results of speed control with matlab graphic interface at different speeds.

The behaviours are evaluated by looking at the different speed using the same parameters of the PID used previously and calculate with the method of Ziegler-Nichols, always powering the motor at 12[V]:

- 50 [rpm];
- 100 [rpm];
- 150 [rpm];
- 210 [rpm].

<b>Speed</b>	<b>50 rpm</b>
<b>Peak value</b>	<b>76 rpm</b>
<b>Overshoot</b>	<b>52 %</b>
<b>Steady-State error</b>	<b>1.8 %</b>

Figure 5.6 behaviour at 50 rpm



<b>Speed</b>	<b>100 rpm</b>
<b>Peak value</b>	<b>120 rpm</b>
<b>Overshoot</b>	<b>20 %</b>
<b>Steady-State error</b>	<b>2 %</b>

Figure 5.7 response at 100 rpm



<b>Speed</b>	<b>150 rpm</b>
<b>Peak value</b>	<b>171 rpm</b>
<b>Overshoot</b>	<b>14 %</b>
<b>Steady-State error</b>	<b>2.4 %</b>

Figure 5.8 behaviour at 150 rpm

<b>Speed</b>	<b>210 rpm</b>
<b>Peak value</b>	<b>220 rpm</b>
<b>Overshoot</b>	<b>4.7 %</b>
<b>Steady-State error</b>	<b>2.8 %</b>

Figure 5.9 behaviour at 210 rpm

The overshoot for low speeds is higher and the steady state increase with increasing speed. At different speed levels, the read system responses are not very oscillating while they oscillate more at low speeds, when PWM decreases. This could be interesting to test in motors with higher power so as to notice the differences in a much more marked way.

## 5.4 Experimental results of the the position control on Arduino

The PID position control speed, made using the Arduino IDE, is the basis of what the position control will be for the linear axis. The results are analyzed on the Arduino serial plotter

### 5.4.1 Experimental result of the position control with fixed target position.

PID parameters used: using the method of Ziegler-Nichols based on the following table:

Control type	$K_p$	$T_i$	$T_d$
P	$0.5K_{p0}$		
PI	$0.45K_{p0}$	$0.8T_0$	
PID	$0.6K_{p0}$	$0.5T_0$	$0.125T_0$

Table 5.2 Ziegler-Nichols method

- $K_p = 0.66$ ;
- $K_i = 6.6$ ;
- $K_d = 0.0165$ ;
- $T_0 = 0.2$  [ms].

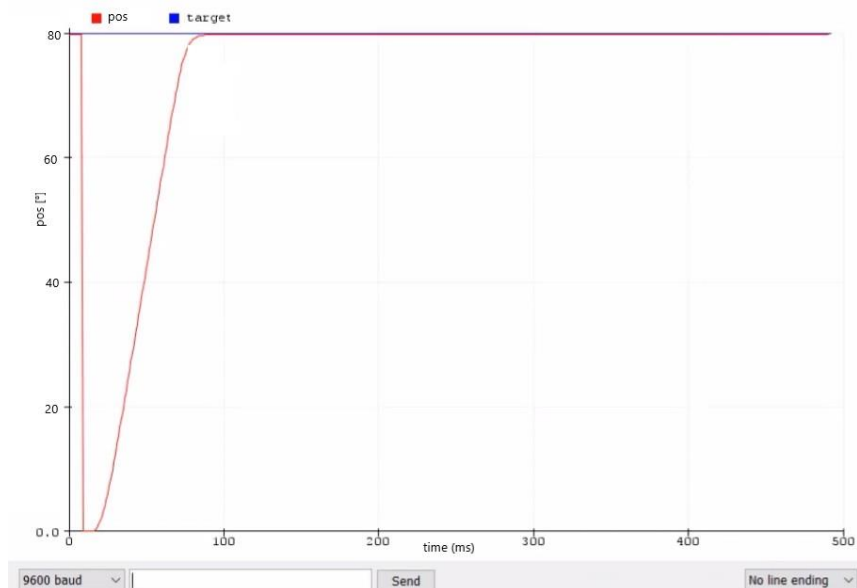


Figure 5.10 Position control on the Arduino serial plotter

## 5.4.2 Experimental result of the position control with sinusoidal test.

To perform this test, unlike the previous one, just replace the int target variable in the code with an int sin( ). For example `target=130sin(prevT)`.

- $K_p = 0.66$ ;
- $K_i = 6.6$ ;
- $K_d = 0.0165$ ;
- $T_0 = 0.2$  [ms].

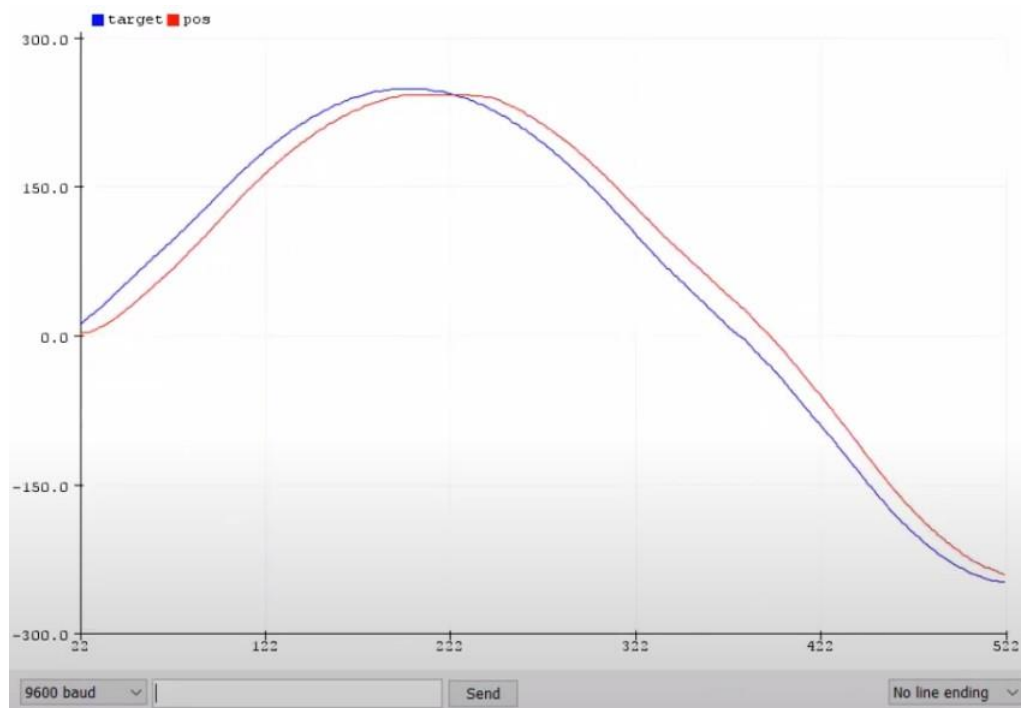
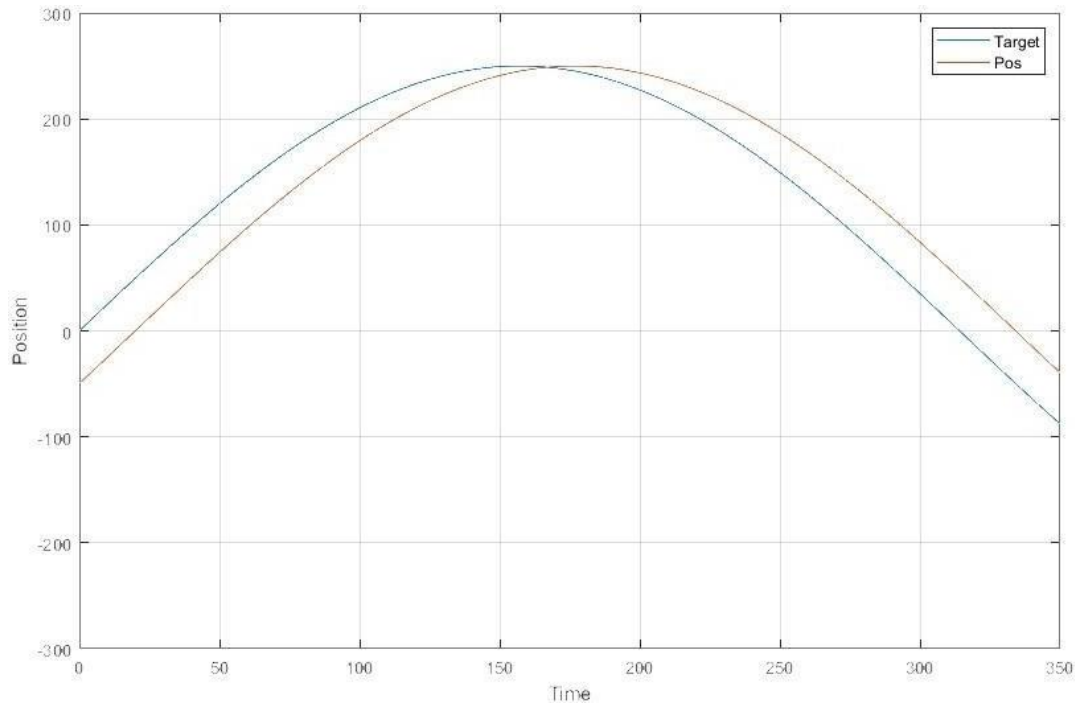


Figure 5.11 Position control, sinusoidal function on the Arduino serial plotter



5.12 Data from Arduino serial plotter to Matlab

## 5.5 Experimental results of the the speed and position control on Arduino, Anti-windup.

First test done by increasing  $K_p$  until displaying oscillations,  $K_d$  and  $K_i$  set to zero.

PID parameters used:

- $K_p = 1$ ;
- $K_i = 0.03$ ;
- $K_d = 0.0015$ ;
- $T_0 = 0.2$  ms;

The motor is rotated in both directions following a sinusoidal path. The data obtained in the Arduino serial plotter were entered on Matlab to obtain a clearer and more readable graph. The maximum speed is setted and it is 230 [rpm]. The system it tends to stabilize

and then to follow the desired profile with a minimum of delay. This is due to the time in which the signal is communicated.

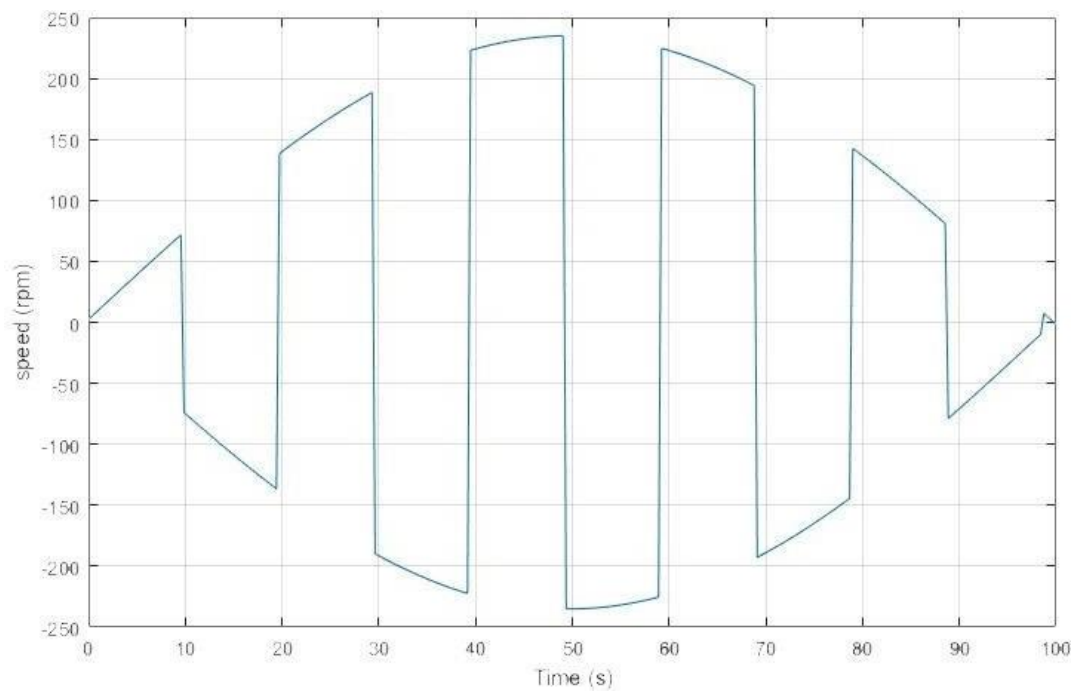


Fig 5.13 Anti WINDUP path

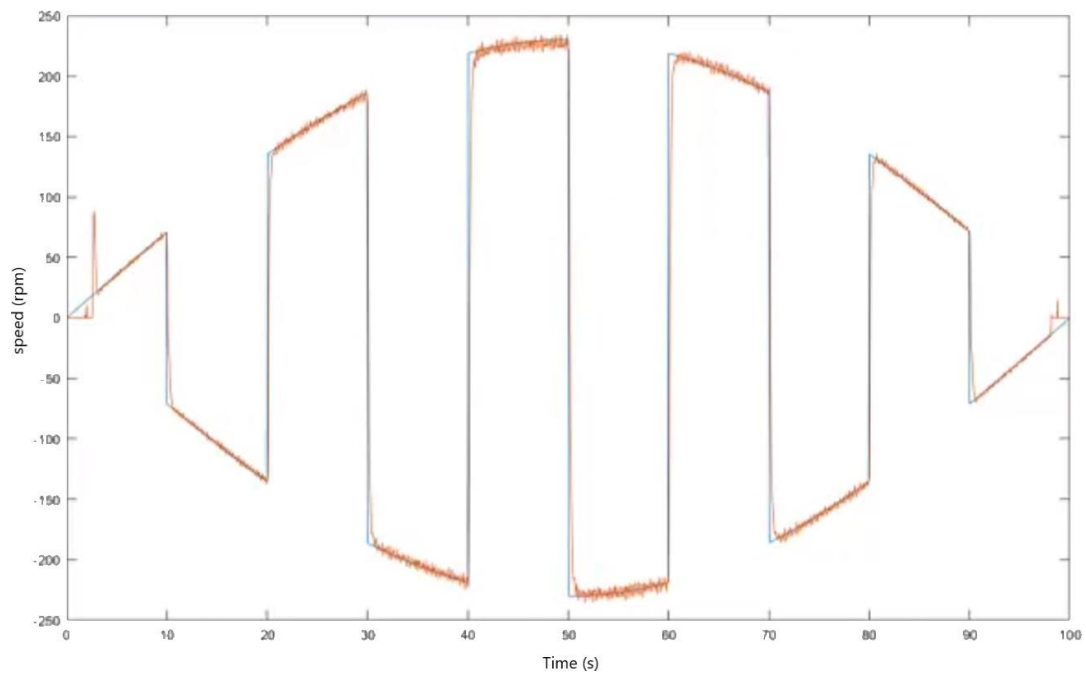


Figure 5.14 PID speed control with Anti-windup technique

## 6. Conclusion and future applications

### 6.1 Final implementation on the linear axis.

After checking the motor both in position and speed, the future application will be to connect it with the linear axis in the fig. 6.1, 6.2 and 6.3

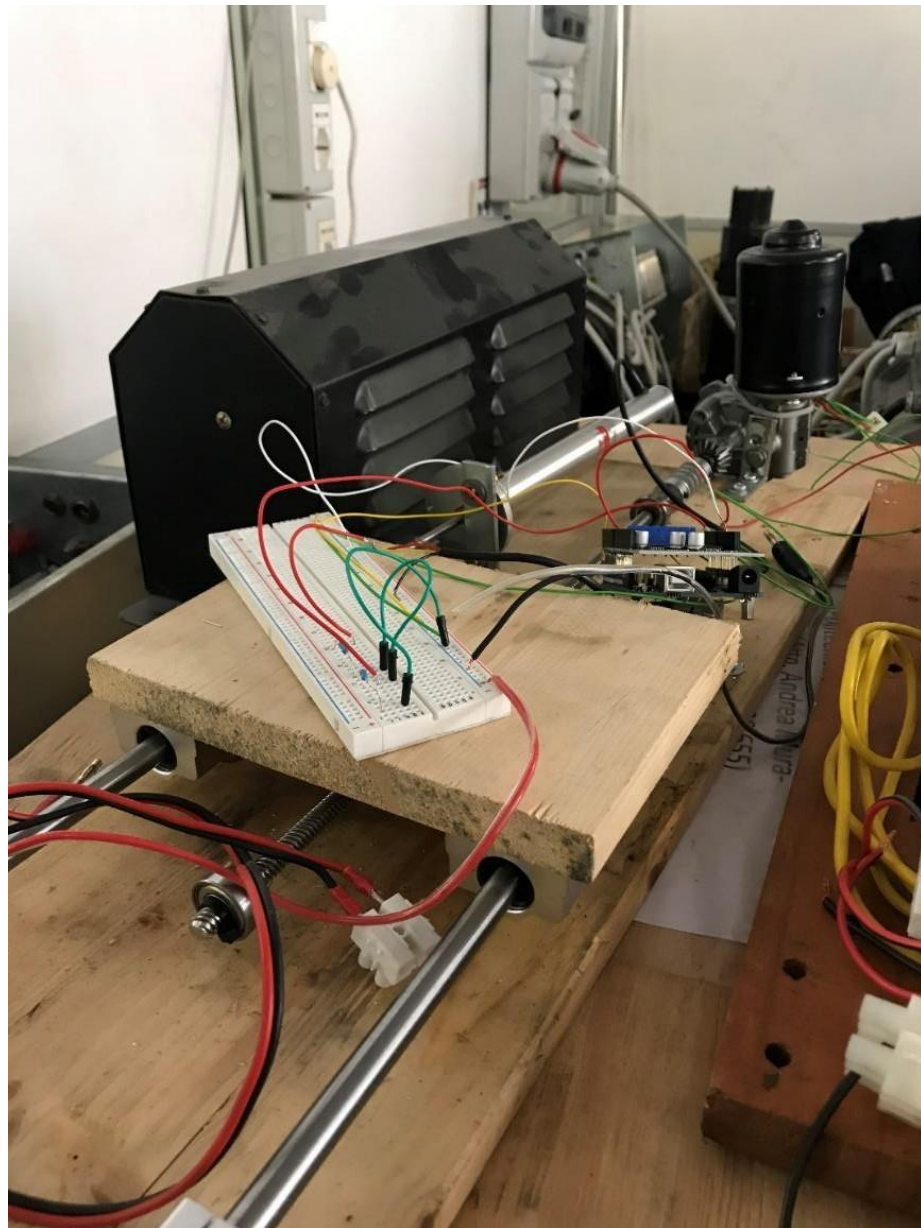


Figure 6.1 Linear axis test bench



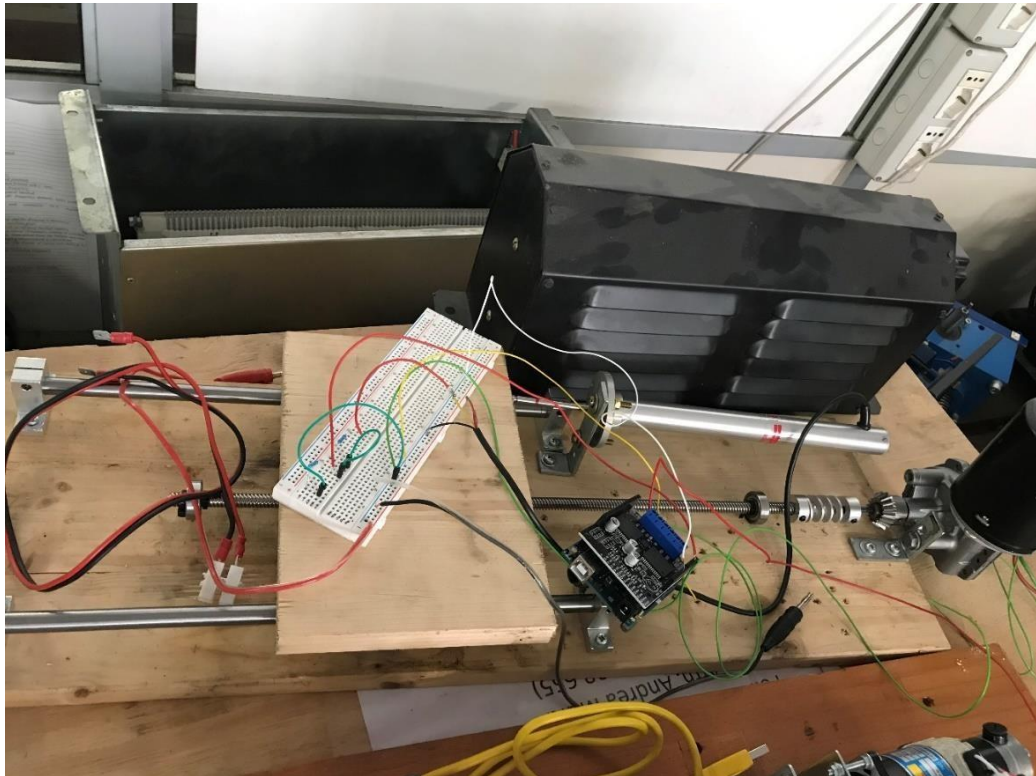


Figure 6.2 linear axis test bench

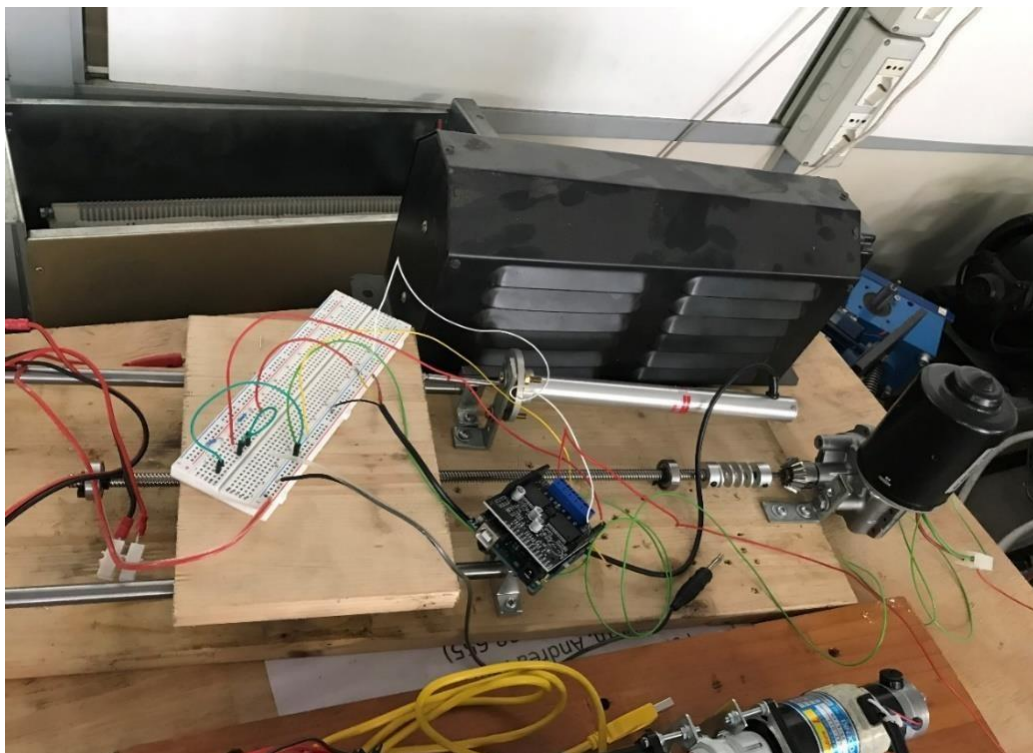


Figure 6.3 linear axis test bench



The screw applied to the DC motor has a pitch of 1[mm] and it will allow the table to be moved forward or backward by 1[mm] at each revolution of the motor. As a result, the rotary motion is transformed into linear motion. The DC motor, with its rotary movement, transmits motion to the reducer which it turn rotates the screw. Being coupled to the screw, they produce a linear movement as being constrained they do not rotate but they 'unscrew or screw'.

## 6.2 Conclusion

The main idea of the work was the creation of a PID controller with a Matlab graphic user interface to stabilise a DC motor. In order to do that, it was crucial to analyse the whole system and its functioning.

The realisation of the Matlab GUI has given satisfactory results. Through the Matlab Support package for Arduino Hardware it was possible to proceed to implement the Arduino controllers and to found clear advantages. First of all, because the functions given by Matlab are better exploited with respect to the only one of the Arduino IDE that would require many lines of code, difficult to implement with C or C++ languages and it would take enormous knowledge in the field of programming. Secondly, it is also possible to control the variables in real time and, thirdly to have the possibility to directly analyse the data without having to move them to Matlab. The experimental data made it possible to tune the parameters of the model with a higher precision than before.

Since the final aim of the thesis would be the control in the position of the linear axis, we can assume that a possible future control can be done with a greater power supply, so as to be able to have better response behaviors especially with larger speed differences, compared to a small size motor.

## Bibliography

- C. Bonivento - C. Melchiorri - R. Zanasi, “Sistemi di controllo digitale”,1995.
- MathWorks, “SystemIdentificationToolbox”
- MathWorks, “Arduino support from Matlab”
- MathWorks, “Arduino support from Simulink”
- Law, A. M., and W. D. Kelton. 1991. *Simulation Modeling and Analysis*, Second Edition,McGraw-Hill.
- Wikipedia, PID controlArduino, Arduino.cc
- Banks, J., J. S. Carson, II, and B. L. Nelson. 1996. *Discrete-Event System Simulation*, Second Edition,Prentice Hall.
- Mathworks, GitHub. “Communication Matlab and Arduino”
- Wikipedia, PWM
- Wikipedia, Motor Driver H-bridge
- Lastminuteengineers.com, Motor Driver H-bridge

- Luca Zaccarian, “Motori CC ed encoders incrementali”.MathWorks, “System Identification Toolbox”.
- Montgomery, D. C. 1997. *Design and Analysis of Experiments*, Third Edition, JohnWiley.
- Law, A. M., and M. G. McComas. 1991. Secrets of Successful Simulation Studies, *Proceedings of the 1991 Winter Simulation Conference*, ed. J. M.

## ***Thanks***

*I would like to thank my tutors: Professors A. Mura and L. Mazza for excellently guiding and supporting me during these months and for what they taught me in the past years;*

*I would also like to thank all the people which in the years of university studies have supported and tolerated me: they know who they are.*

*To 'Zia Rosa and Nonna Melina' who celebrate with us from the sky.*