Politecnico di Torino

Master of Science in Mechatronics Engineering

Master Degree Thesis

Torque estimation in closed-loop position control with DC motors

Supervisor:
Prof. Marcello Chiaberge

Candidate:
Sebastiano Marmo

Co-supervisor:
Ing. Alessandro Buscicchio

Academic Year 2020/21
Torino

Abstract

Design and control with efficiency and performance are one of the most goals to achieve in the robotics field, cranes or aerospace automation. The stability of the system is essential and the non-linear effects create by motor reduce stability, like maintain speed or position at set points. The position control of DC motor allows it to move to a precise position and remain there even if an external force tries to move it. Different techniques have been developed to perform the position control of DC Motor, the most used are Proportional-Integral-Derivative(PID), Fuzzy Logical Model(FLM) and Artificial Neural Network(ANN).

This thesis discusses the implementation and tuning steps of the Proportional Integral Derivative (PID) controller for the position control of Brushed DC Motor. Analysing three principal feedback (Rotation, Speed and Current), using STM-Nucleo-G431RB microcontroller and monitoring the data in a GUI-Matlab, specially designed for this purpose based on UART communication protocol. Since the system is stable, the performance of the system is analyzed and validation is done in terms of robustness, time response and percentage of error.

In conclusion, Torque estimation has been carried out through a load cell, in order to realize a function Torque-Current and monitoring how much Torque DC Motor is producing at any given moment.

Acknowledgements

Priama di inizare la presentazione, vorrei dedicare questo spazio alle persone che mi hanno supportato nella realizzazione del progetto di tesi.

Ringrazio il mio correlatore Ingegnere della NASA JPL Alessandro Buscicchio che, in questi sei mesi di lavoro, mi ha guidato nel progetto dandomi consigli utili al fine di migliorare la ricerca e lo sviluppo della tesi. Grazie a lui ho migliorato l'approccio alla realizzazione di progetto, mettendo in pratica le conoscenze acquiste durante il percorso di studi, le quali saranno utili per la mia carriera professionale. Ringrazio il mio relatore Dr. Marcello Chiaberge per la sua disponibilità e consigli utili nello sviluppo di tesi.

Un ringraziamento speciale alla mia famiglia che mi hanno sempre supportato nelle scelte e che, nonostante la distanza, sono stati sempre al mio fianco anche durante le difficoltà.
Un grazie a mia madre per la sua tenacia a non arrendersi mai e avermi insegnato che ogni ostacolo si può superare.
Un grazie a mio padre per la fiducia che mi ha sempre dato nella realizzazione dei miei desideri.
Un grazie a mia sorella, per me esempio di determinazione e grinta, che mi ha aiutato nelle scelte ed è sempre al mio fianco e che mi ha sempre guardato con ammirazione.

Un ringraziamento a tutte le persone che ho incontrato in questo percorso, con le quali ho potuto avere un confronto costruttivo e di condivisione. Un grazie ai miei coinquilini, a quelli passati e quelli presenti, che mi hanno sempre ascoltato e spronato ad andare avanti. A tutti i miei amici di sempre, grazie per essermi stati vicini nel darmi supporto nella realizzazione dei miei progetti.

Contents

List of Figures

Listings

Chapter 1

Introduction

Nowadays, industrial system and commercial system the DC Motors are widely used, for example in cars, trains, sensor placement, elevator and robotics field. This last field is used for the arm, changing the angle of the joint.
The robot field in the last decade is linked with the medical or educational application. But to substitute the human work the robot behavior must be precise(high accuracy), it must guarantee stability and robustness.
However, modeling a system which requires high accuracy some techniques are taken into account to control the behavior, but the most used is the PID controller.

In this dissertation has been implemented a PI position controller, the tuning and the stability have been performed with a STM32Nucleo Board for Brushed DC Motor. The torque estimation has been developed in order to monitor the complete behavior of the motor and how much torque it is generating.
The goal of this thesis is to present a torque PI controller and a graphical interface design. The system is developed for robotic field or for sensor placement, using the torque of DC Motor, in order to guarantee the correct pressure on the sensor.

1.1 Thesis overview

In the second chapter, an overview of the DC Motor is made in order to understand the components that take part in the project and how it works. For example, the functioning of the encoder or the calculation of the torque with mathematical equations.

After that, the third chapter will be illustrated the components that will take part in the implementation of the project, both hardware and software. The IDE and the components for the torque estimation.

In the fourth chapter the description of the system, the implementation of three PI controller in cascade, the code used for the correct behavior of the system, following the mathematical equations and the design of GUI-Matlab with UART protocol communication.

Then, the next chapter is reserved for the tuning and stability of the system. Firstly, the tuning part have been developed for the correct value of proportional and integral gain of the three PI position controller. After that, the stability and robustness of the system has been analyzed, implementing a motion profile.

In the last chapter of the thesis, a torque estimation was realized through a load cell and the complete model of the system was monitored using a graphical interface, plotting data in real-time.

Chapter 2

Brushed DC Motor

2.1 Introduction

The Brushed DC motor was the first commercially important application of electric power to driving mechanical energy, and the system was used for more than 100 years to operate motor in commercial and industrial buildings.

Thanks to the contribution played in the industrial and technological advance, DC motor theory and practice were extensively analyzed by luminaries such as Charles Steinmetz. This included detailed circuit, electromotive-force and magnetic-field equations, performance (speed, torque, control and efficiency) and design. It has been overshadowed by the brushless DC motor, but it remains the best choice in terms of adequate performance, low cost and sufficient reliability.

2.2 Brushed DC Motor Basic

All motors depend on the interaction between an electromagnetic (EM) field on the fixed body (the stator), the EM field on the rotating armature (the rotor) and how these fields are controlled and changed induce magnetic attractive/repulsion and thus motion.

The rotating field across the armature reacts with the stator magnetic field creating a force on the rotor winding ($F = l \times \vec{L} \wedge \vec{B}$). Using Fleming's left-hand rule is possible to determine the direction of the force and the rotation of the motor (clockwise or counterclockwise).

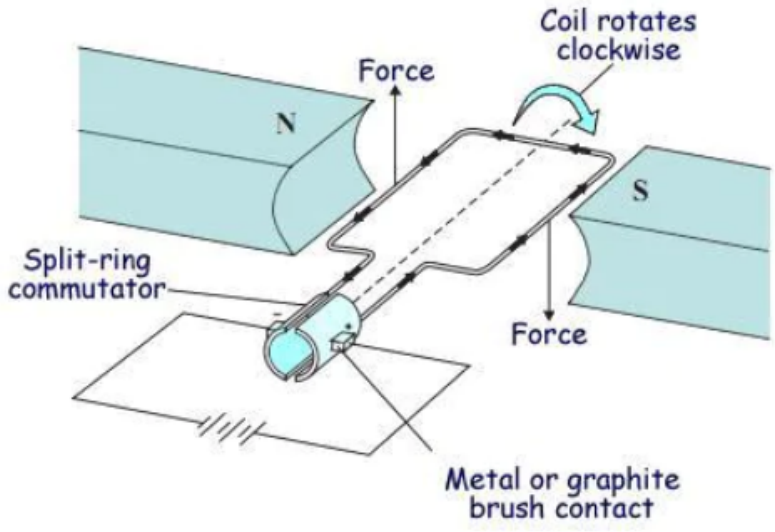

Figure 2.1. Scheme of DC motor.

2.3 Components of Brushed DC Motor

2.3.1 Stator/Inductor

The stator generates a stationary magnetic field that surrounds the rotor. This field is generated by either permanent magnets or electromagnetic windings. The different types of BDC motors are distinguished by the construction of the stator or the way the electromagnetic windings are connected to the power source.

2.3.2 Rotor/Armature

The rotor is made up of one or more windings. When these windings are energized they produce a magnetic field. The magnetic poles of this rotor field will be attracted to the opposite poles generated by the stator, causing the rotor to turn. As the motor turns, the windings are constantly being energized in a different sequence so that the magnetic poles generated by the rotor do not overrun the poles generated in the stator. This switching of the field in the rotor windings is called commutation.

2.3.3 Brushes and Commutator

Unlike other electric motor types (i.e., brushless DC, AC induction), BDC motors do not require a controller to switch current in the motor windings. Instead, the commutation of the windings of a BDC motor is done mechanically. A segmented copper sleeve, called a commutator, resides on the axle of a BDC motor.

As the motor turns, carbon brushes slide over the commutator, coming in contact with different segments of the commutator. The segments are attached to different rotor windings, therefore, a dynamic magnetic field is generated inside the motor when a voltage is applied across the brushes of the motor. It is important to note that the brushes and commutator are the parts of a BDC motor that are most prone to wear because they are sliding past each other.

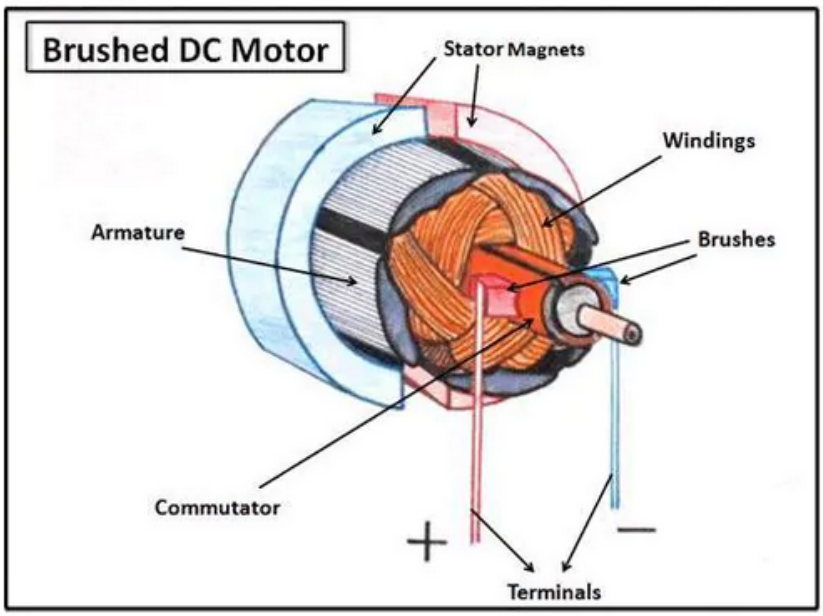

Figure 2.2. Components of DC Motor.

2.4 Hall-effect sensor for Brushed DC Motor

A Hall-effect sensor (or simply Hall sensor) is a device to measure the magnitude of a magnetic field. Its output voltage is directly proportional to the magnetic field strength through it. The sensor type is named after the American physicist Edwin Hall. In a Hall-effect sensor, a thin strip of metal has a current applied along it. In the presence of a magnetic field, the electrons in the metal strip are deflected toward one edge, producing a voltage gradient across the short side of the strip (perpendicular to the feed current). Hall-effect sensors have an advantage over inductive sensors in that, while inductive sensors respond to a changing magnetic field that induces current in a coil of wire and produces voltage at its output, Hall-effect sensors can detect static (non-changing) magnetic fields.

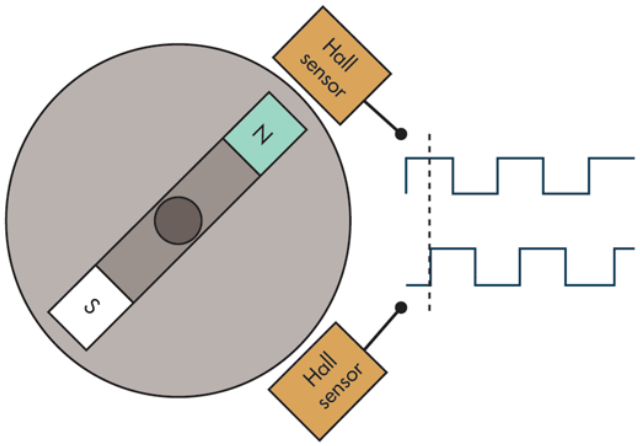

Figure 2.3. Magnetic encoder for DC Motor with the corresponding logic states of the A and B signals.

2.5 Gearbox

Gearboxes are used to increase torque while reducing the speed of a prime mover output shaft (e.g. a motor crankshaft). This means that the output shaft of a gearbox rotates at a slower rate than the input shaft, and this reduction in speed produces a mechanical advantage, increasing torque.

$$\mathrm{GR} = \frac{N1}{N2} = \frac{d2}{d1}$$

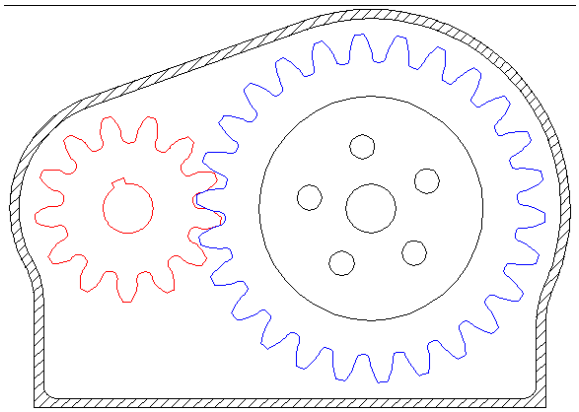

Figure 2.4. Single stage gear reducer.

2.6 Torque

A DC motor's speed and torque characteristics vary according to three different magnetization sources, separately excited field, self-excited field or permanent-field, which are used selectively to control the motor over the mechanical load's range. Self-excited field motors can be series, shunt, or a compound wound connected to the armature.

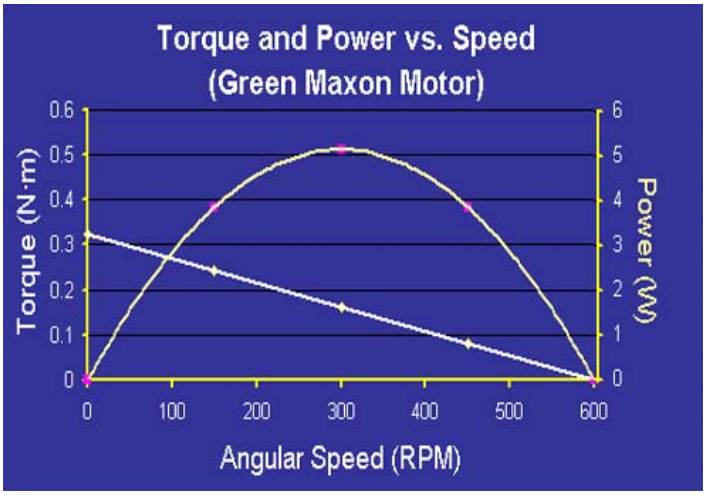

Figure 2.5. Brushed DC motor characteristics.

2.6.1 Counter EMF

The DC motor's counter emf is proportional to the product of the machine's total flux strength and armature speed:

$$E_b[V] = k_b \Phi[Wb] n[rpm]$$

where, E_b is the induced EMF, Φ is the machine's total flux, n is the armature frequency and k_b is the counter EMF equation constant.

2.6.2 Voltage balance

The DC motor's input voltage must overcome the counter emf as well as the voltage drop created by the armature current across the motor resistance, that is, the combined resistance across the brushes, armature winding and series field winding, if any:

$$V_m[V] = E_b[V] + R_m[\Omega] I_a[A]$$

where the V_m is the motor input voltage, R_m is the motor resistance and the I_a is the armature current.

2.6.3 Torque equation

The DC motor's torque is proportional to the product of the armature current and the machine's total flux strength:

$$T[Nm] = \frac{1}{2\pi} k_b I_a[A] \Phi[Wb] = k_T I_a[A] \Phi[Wb]$$

where k_T is the torque constant equation.

2.6.4 Speed

The speed rotation is calculated through the following equation:

$$n[rpm] = \frac{E_b[V]}{k_b \Phi[Wb]}$$

$$n[rpm] = \frac{V_m[V] - R_m \Omega I_a[A]}{k_b \Phi[Wb]} = k_n \frac{V_m[V] - R_m \Omega I_a[A]}{\Phi[Wb]}$$

where k_n is the speed equation constant.

Chapter 3

Software and Hardware requirements

3.1 Introduction

This chapter aims to show which hardware and software are required in order to perform the project, explaining the STM32 evaluation board, the shield used to control DC Motor and the graphical interface.

3.2 Software

The Software section analysis all environment used for modeling, simulation and measurement estimation.

- STM32CubeIDE: Software from STMicroeletronics (version 1.4.0). It is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors.
- Matlab/GUI: Software from MathWorks(2018a):Simulink, an add-on product to MATLAB, provides an interactive, graphical environment for modeling, simulating, and analyzing of dynamic systems. It enables rapid construction of virtual prototypes to explore design concepts at any level of detail with minimal effort.
- GUI-Matlab: Graphical user interfaces (GUIs), also known as apps, provide point-and-click control of your software applications, eliminating the need for others to learn a language or type commands in order to run the application.

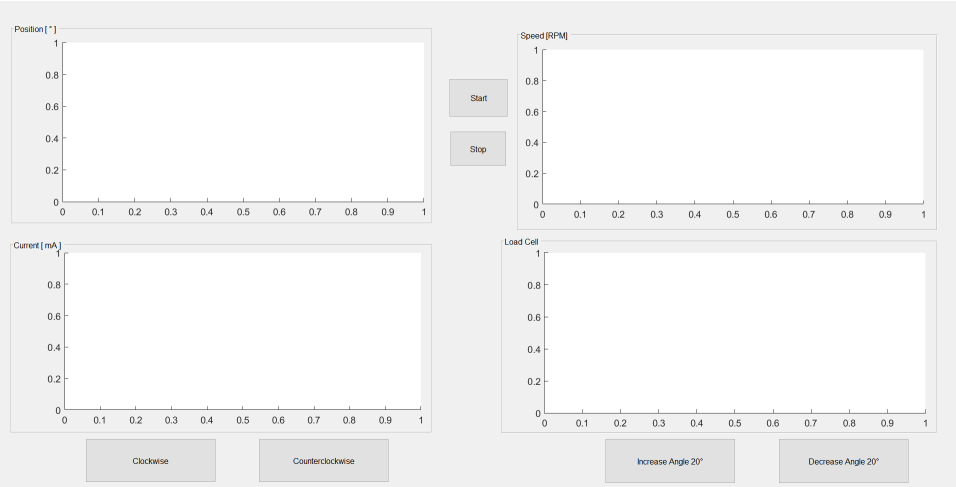

Figure 3.1. GUI-Matlab

3.3 Hardware

The Hardware section analyses DC motor, board and Micro used for this project.

- STM-NucleoG431RB: Hardware from STMicroelectronics. The main features of this microcontroller are:
  - Arm® Cortex®-M4 at 170 MHz
  - 128 KBytes of Flash memory and 32 Kbytes of SRAM
  - 4 Channels for ADC with 12-bit
  - 16 Timers for PWM
  - 3 Operational Amplifier
  - Uart communication. Baud Rate 150 Mbps

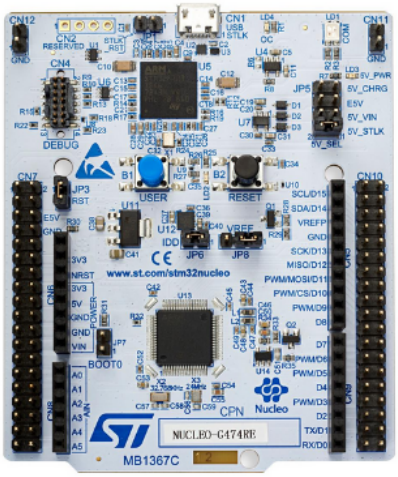

Figure 3.2. NUCLEO-G431RB

- X-NUCLEO-IHM13A1: The X-NUCLEO-IHM13A1 expansion board for STM32 Nucleo is based on the STSPIN250 low voltage brush DC motor driver. The main features are:
  - Low voltage range from 1.8 V to 10 V;
  - Current up to 2.6 A r.m.s.;
  - Full over-current and short circuit protection;

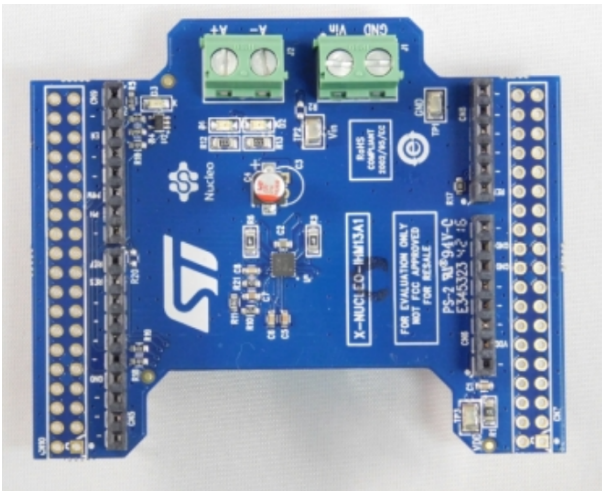

Figure 3.3. Shield X-NUCLEO-IHM13A1

- DealMux DLM-B0792RPRD2 Motor: Brushed DC Motor with Optical Encoder DC 12V 38RPM.
  - Rated voltage 12 V
  - Voltage range 6-24 V
  - Speed 38 rpm
  - Rated Torque 9 kg.cm
  - Reduction ratio 1:131
  - Rated Current 0.45 mA

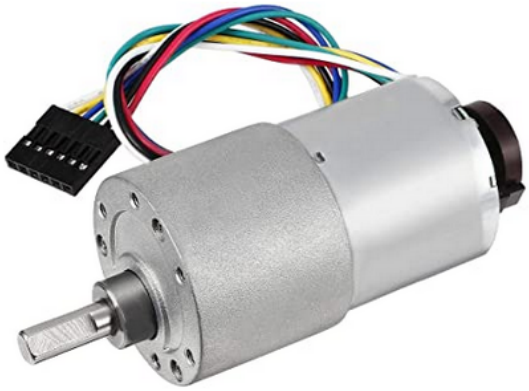

Figure 3.4. DealMux DC Motor with encoder

- ARCELI Aluminum Alloy Scale Weighing Sensor Load Cell: This straight bar load cell (sometimes called a strain gauge) can translate up to 5kg of pressure (force) into an electrical signal.
  - Measurement range 0-5 kg
  - Voltage excitation 5-10 V
  - Four strain gauges that are hooked up in a Wheatstone bridge formation

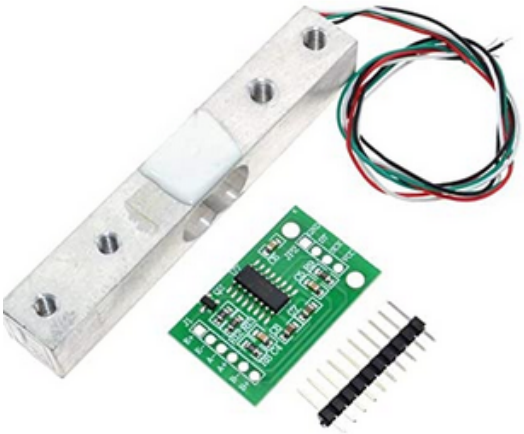

Figure 3.5. ARCELI Load Cell.

- Beam made with 3D-Printer for calculating the torque.
  - Length 10 *cm*
  - 100% Infill density

Figure 3.6. Beam using a 3D-Printer

Chapter 4

System Design

4.1 Introduction

A typical structure of a PI is shown in Figure 4.1. The equation of a PI controller is:

$$u(t) = K_p e(t) + K_i \int e(t)dt$$

where the $e(t)$ is the error $e(t) = y(t) - u(t)$ between output $y(t)$ and the input $u(t)$ of the system, but for microcontroller this formula is write in discrete time.

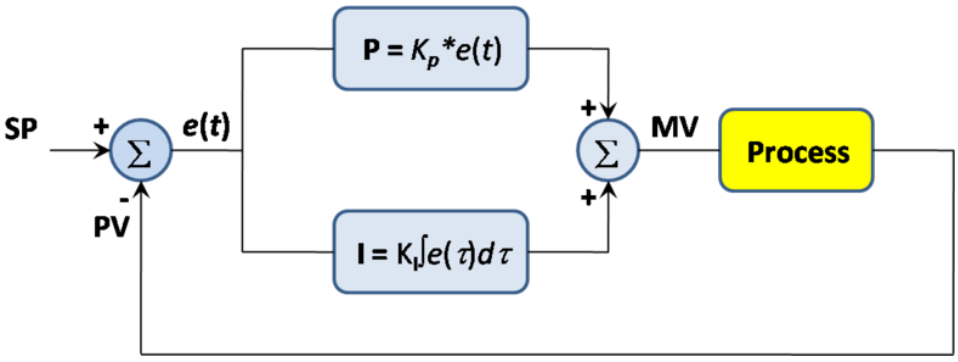

Figure 4.1. PI Controller

To increase the accuracy and the control of DC Motor, this equation was developed for three PI controller, as shown in Figure 4.2. The first was a PI current loop, the second was a PI velocity loop and the third a PI position loop.

To achieve the complete PI position control with three feedback, the system design was split into three main part, in order to analyze each feedback separately.

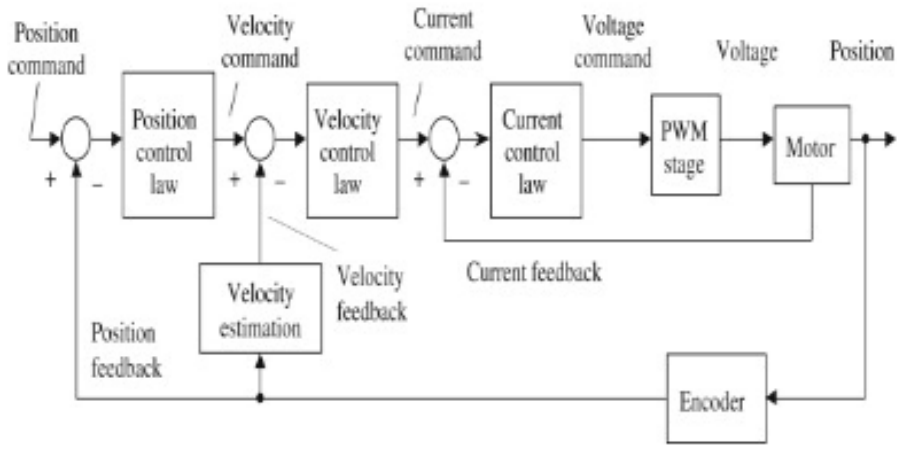

Figure 4.2. Position controller: three loop cascade

4.2 PI current controller

4.2.1 Implantation

To control a DC Motor the PWM frequency has been set equal to $70kHz$ and to measure the current a jumper wire was welded to the R_{sense} which is connected between ground and lower p-Mosfets of the H-Bridge, as shown in Figure 4.3

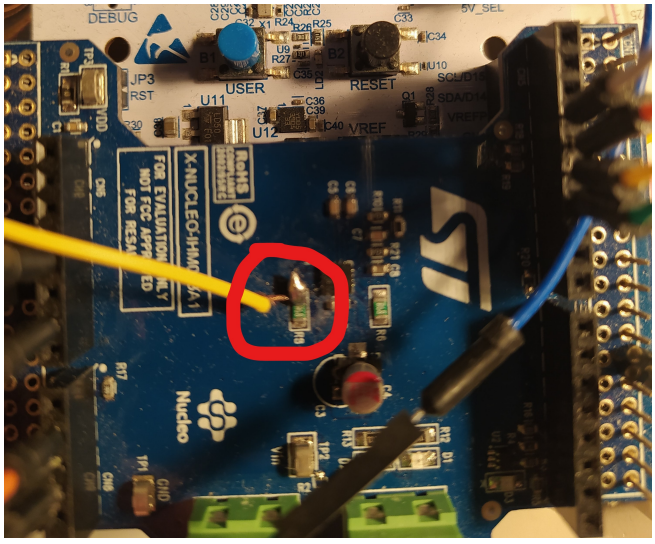

Figure 4.3. Welding

The slow decay created a ripple voltage, because when the PWM is off, the lower Mosfets of the H-Bridge were ON and no current flow into R_{sense}.

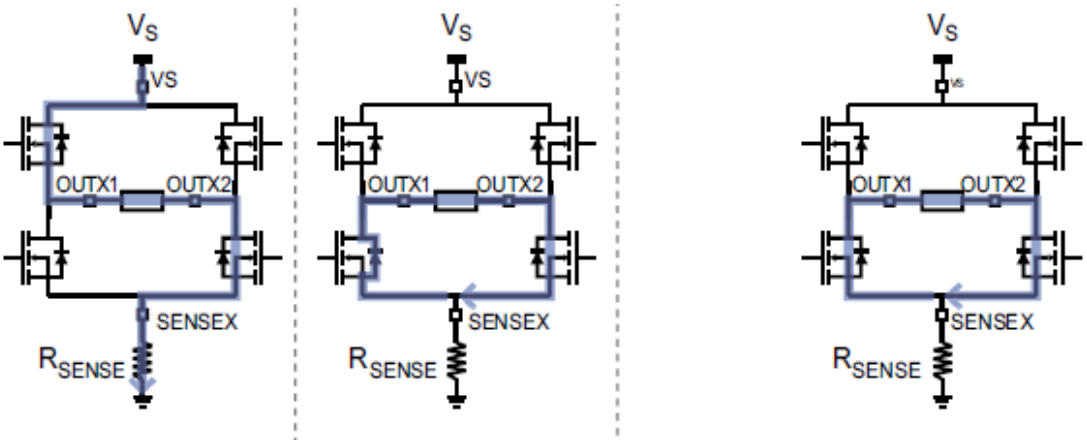

Figure 4.4. Phase of slow decay: Normal Operation, Dead Time, Slow Decay

The voltage on R_{sense} has been measured using an ADC2 with Synchronous clock mode at $60MHz$, 640,5 cycle for sample time and $12bits$ for high resolution. The error in terms of voltage was calculated by multiplying the number of LSBs by the voltage corresponding to 1LSB ($1LSB = V_{REF} \div 2^{12}$), where $V_{REF} = 3.3V$. However, the ADC did not measure the Root Mean Square, but the instantaneous voltage and the ripple were functions of the Duty Cycle. To obtain the Root Mean Square voltage an active low pass filter has been designed (Figure 4.5), in order to cut di high frequencies.

$$f_{cut} = \frac{1}{2\pi CR} = \frac{1}{2\pi 5k\Omega 100nF} = 318{,}47Hz$$

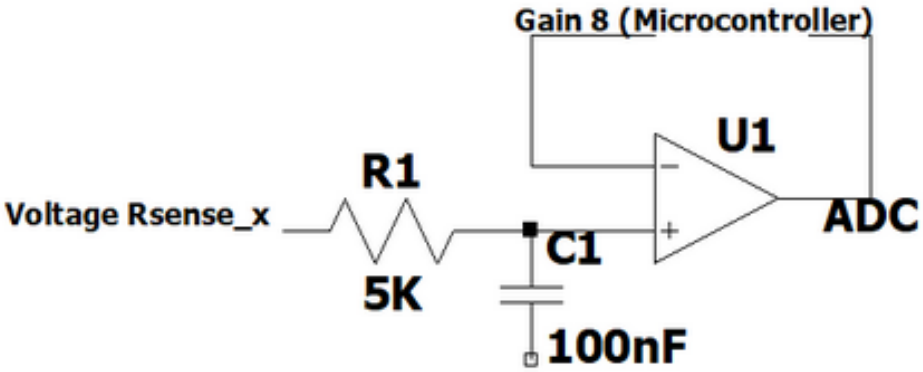

Figure 4.5. Active Low Pass Filter with $f_c = 318Hz$

Figure 4.6. Voltage measured using oscilloscope with low pass filter [Yellow] and without low pass filter [Blue] $f_c = 318Hz$

After that, due to maximum voltage on R_{sense} which is equal to $300mV$ Operation Amplifier was used in mode PGA internally Connected with PGA gain equal to 8, in order to maximize the resolution of the ADC($0V \div 3.3V$). The equations below show the calculation for the current measurement.

$$V_i[V] = \frac{3.3V}{4096[LSB]} ADC[LSB]$$

$$V_{rms}[V] = \sqrt{\frac{1}{N} \sum V_i^2[V]}$$

$$I_{DC\_Motor}[mA] = \frac{V_{rms}[V]}{330m\Omega} \frac{1000[mA]}{[A]}$$

4.2.2 Code

The code implemented in the Microcontroller show the code of current measurement, using ADC and generate an RMS voltage by software.

```
/* Get ADC value */
/* Slow decay create 0 V */
for(uint8_t j=0; j<100; j++)
{
  /*ADC Start the conversation*/
  HAL_ADC_Start(&hadc2);
  HAL_ADC_PollForConversion(&hadc2, HAL_MAX_DELAY);
  value = HAL_ADC_GetValue(&hadc2);

  /* The value of ADC is converted in Volt */
  /* 3.3V/4096 */
  value = (value)/1240;
  value = value*value;
  rms = rms + value;
}
voltage = sqrt(rms)/10;
/* 375 = 3.03 * 1000 / 8 */
current_read = (voltage * 375);
```

Listing 4.1. Current measurement in C code

The PI current controller design for the microcontroller, as shown below, was in discrete time.

```
/* Current error between current target and real current of the
   system */
error_current = current_target - current_read;

/*Error for integral part*/
Integral_error_current += error_current*elapsed_time;

/* Sum proportional and integral part*/
pulse += Kp_current*error_current + Ki_current*
   Integral_error_current;

/* Saturation for duty cycle */
if(pulse>2150)  {
```

```
12   pulse = 2200;
13 }
14   if(pulse<-2150) {
15   pulse = -2200;
16 }
```

Listing 4.2. PI controller in C code

4.3 PI speed controller

The PI speed controller was the second feedback developed to control the DC Motor in position control.

4.3.1 Implementation

A magnetic rotary encoder on the motor shaft was used to read the correct DC motor speed. One phase of the encoder has been connected to pin TIM3 of the microcontroller, which is enabled as a direct input acquisition mode. Input capture is used to capture the input signal given to the microcontroller and measures its frequency and pulse width. The frequency of the Micro is $170MHz$ and by setting the prescaler to 170 the frequency of TIM3 in the interrupt capture was $1MHz$ and the counter was set in hexadecimal $0xFFFF(65536)$. The Interrupt was enabled on both the high and low edges. The highest frequency increased the resolution to detect the speed of DC Motor.
These equations explain how the DC motor speed in RPM could be read and implemented for the PI speed controller.

$$f_{TIM3}[Hz] = 1MHz$$

$$f_{hall}[Hz] = \frac{f_{TIM3}[Hz]}{(RISING_{edge} - FALLING_{edge})}$$

$$v_{read}[RPM] = \frac{f_{hall}[Hz]}{Hall_{encoder} Ratio_{GearBox}} \frac{60[RPM]}{[Hz]}$$

The frequency can be calculated by first finding the difference between these two captured values $RISING_{edge}$ and $FALLING_{edge}$ and then dividing the timer clock by this difference. $Hall_{encoder}$ is a dimensionless number equal to 32, it represents how many edges are present into the encoder. While the $Ratio_{GearBox}$ is equal to 131.

Thanks to the gearbox the error is minimum because the error read by the encoder is divided by 131.

4.3.2 Code

The code implemented in the Microcontroller for speed measurement is shown below.

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
  ....
  uwFrequency = 1000000/ uwDiffCapture;
  RPM_read = (uwFrequency*60)/(32*131);
  /* This if change the sign of RPM_read for real-time data in GUI-
   Matlab*/
  if(negative_flag == 1)
  {
    RPM_read = RPM_read*(-1);
  }
  ...
}
```

Listing 4.3. Speed measurement in C code

The PI speed controller design for the microcontroller, as shown below, was in discrete time.

```
/* Speed error between speed target and real speed of the system */
error_RPM = RPM_target - RPM_read;

/* Error for ingeral part */
Integral_error_RPM += error_RPM*elapsed_time;

/* Sum proportional and integral part */
current_target += Kp_RPM*error_RPM + Ki_RPM*Integral_error_RPM;
```

Listing 4.4. PI controller in C code

4.4 PI position controller

The last feedback was a PI position controller. This feedback is the most important feedback to develop because the project is in position control and it monitors the degrees of rotation.

4.4.1 Implementation

To measure the position of the DC Motor, the two phases of the encoder was were taken in input. Phase A was analyzed above for the PI speed controller, while phase B was set as GPIO_PIN_0 input. The two halls are out of phase by 90 degrees. When an interrupt function created for the speed controller has invoked the encoder position variable increase or decrease the value. Phase A and phase B were analyzed to determine the correct rotation of the Encoder, when phase A is high if phase B is low the motor rotates clockwise (Figure 4.7), otherwise it rotates counterclockwise(Figure 4.8).
The gearbox creates an opposite rotation between Encoder and the output shaft, in the code, it was taken into account, so the logic is reversed.

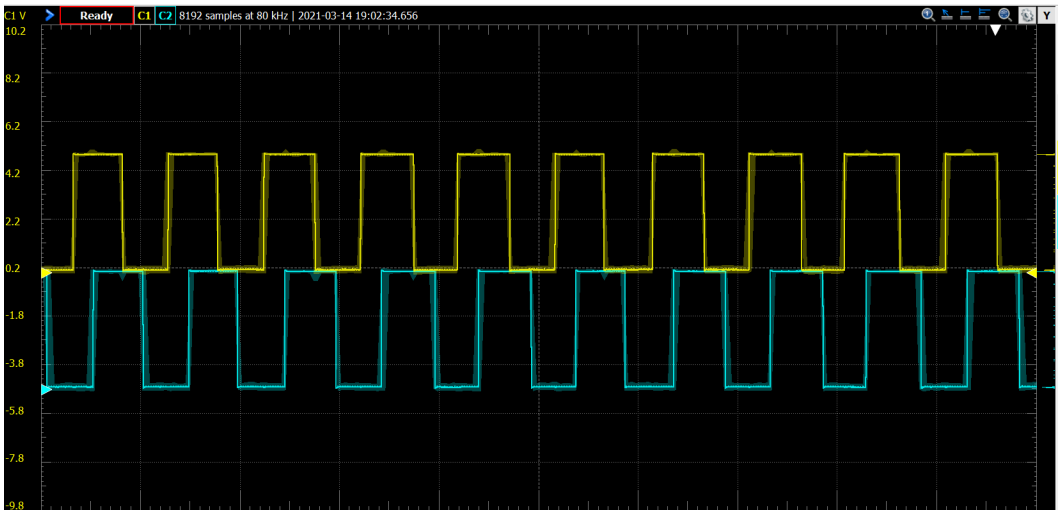

Figure 4.7. Enocder rotating in Clockwise

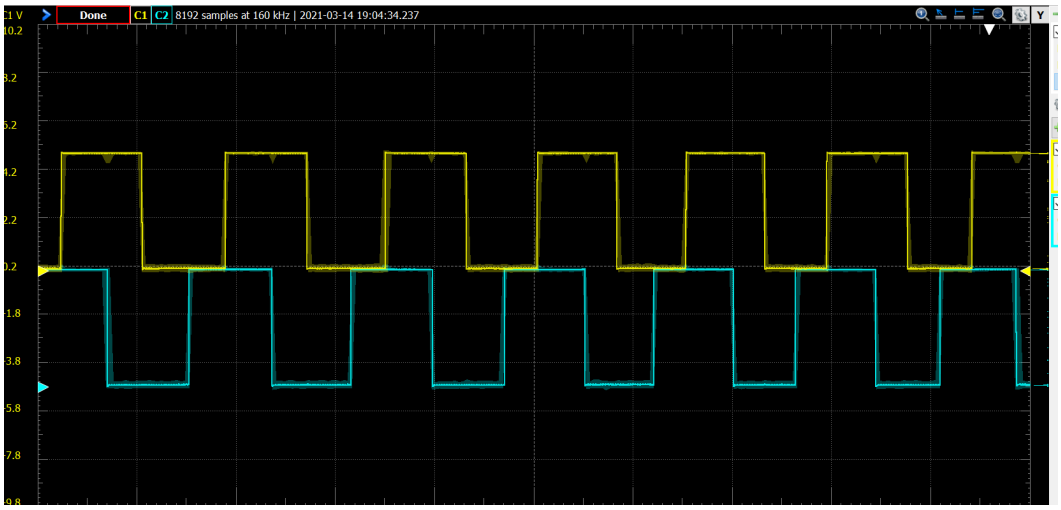

Figure 4.8. Enocder rotating in Counter-Clockwise

4.4.2 Code

The PI position controller design for the microcontroller, as shown below, was in discrete time. The function is the same for speed because the comparison occurs when the interrupt is invoked.

```
if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_0) == HAL_GPIO_ReadPin(GPIOA,
    GPIO_PIN_4))
{
  /* Clockwise */
  Encoder_position ++;
}
else
{
  /* Counterclockwise */
  Encoder_position --;
  negative_flag = 1;
}
```

Listing 4.5. PI position in C code

4.5 UART communication by STM32G431RB

4.5.1 Implementation

The channels used for Low Power Universal Asynchronous Receiver-Transmitter(LPUART) were the pin PA2 for the transmitter communication and the pin PA3 for the receiver communication using a wizard interface integrated into STM32CubeIDE, as shown in the Figure 4.9. The parameters selected for a robust and speed communication were the following:

- Baud Rate 115200$Bits/s$
- Word Length 9$Bits$ (including Parity)
- Parity Even
- Stop bit 1

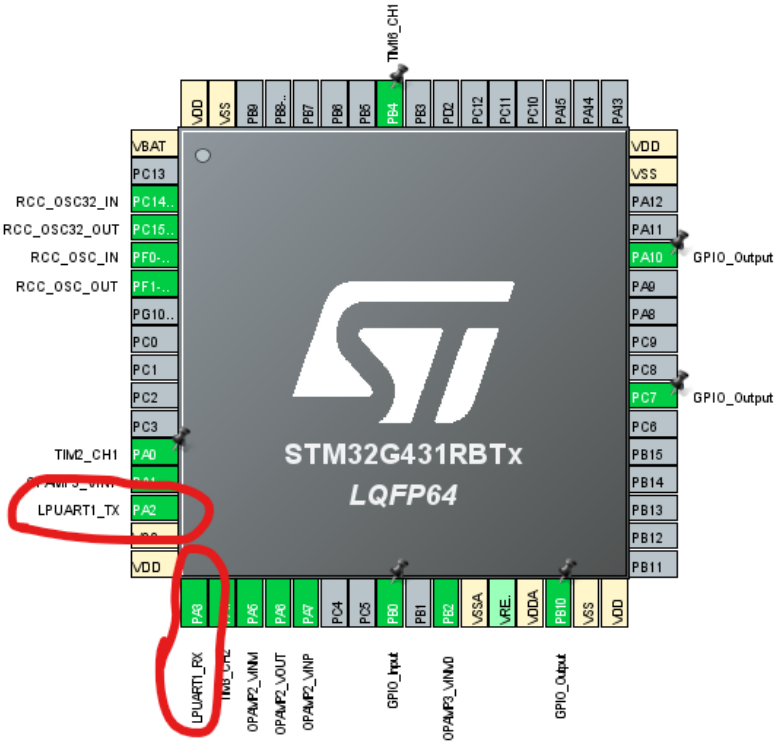

Figure 4.9. Pin used for UART communication

The communication of the micro has been set with an interrupt for the reception, so as not to remain to wait by subtracting time from the processor to execute more important instructions, such as controlling the PI. While the transmission takes place in polling because every time the data is updated in the cycle, such as speed or current, it will be transmitted to the GUI-Matlab for real-time control.

4.5.2 Code

Transmission

The code below shows the data sent from the Microcontroller in a polling mode to Matlab, using two control characters "b" and "e", in order to guarantee a correct transmission.

```
while(1)

/* Transmission begin sending a char b */
if(HAL_UART_Transmit(&hlpuart1, &b, 1, 1000)!= HAL_OK){
/* Transfer error in transmission process */
Error_Handler();
}

/* Send Current of DC Motor */
if(HAL_UART_Transmit(&hlpuart1, &current_read, 2, 1000)!= HAL_OK){
/* Transfer error in transmission process */
Error_Handler();
}

/* Send Chosen Position */
if(HAL_UART_Transmit(&hlpuart1, &Position_target, 2, 1000)!= HAL_OK
   ){
/* Transfer error in transmission process */
Error_Handler();
}

/* Send the Real Position */
if(HAL_UART_Transmit(&hlpuart1, &Position_read, 2, 1000)!= HAL_OK){
/* Transfer error in transmission process */
Error_Handler();
}
```

```
26
27 /* Send the Speed Read through Encoder */
28 if(HAL_UART_Transmit(&hlpuart1, &RPM_read, 2, 1000)!= HAL_OK){
29 /* Transfer error in transmission process */
30 Error_Handler();
31 }
32
33 /* Transmission ends up sending a char e */
34 if(HAL_UART_Transmit(&hlpuart1, &e, 1, 1000)!= HAL_OK){
35 /* Transfer error in transmission process */
36 Error_Handler();
37 }
38 }
```

Listing 4.6. Polling mode for UART TX

Reception

The code below shows the data received by the micro in interrupt mode.

```
1 void UART_Error_Callback(void)
2 {
3 __IO uint32_t isr_reg;
4
5 /* Disable USARTx_IRQn */
6 NVIC_DisableIRQ(LPUART1_IRQn);
7
8 /* Error handling example :
9 - Read USART ISR register to identify flag that leads to IT raising
10 - Perform corresponding error handling treatment according to flag
    */
11 isr_reg = LL_USART_ReadReg(LPUART1, ISR);
12 if (isr_reg & LL_USART_ISR_NE)
13 {
14 /* Turn LED2 on: Transfer error in reception/transmission process
    */
15 BSP_LED_On(LED2);
16 }
17 else
18 {
19 /* Turn LED2 on: Transfer error in reception/transmission process
    */
```

```
20 BSP_LED_On(LED2);
21 }
22 }
```

Listing 4.7. Interrupt Function for UART RX

4.6 UART communication by GUI-Matlab

4.6.1 Implementation

The parameters that have been used for the Matlab serial are the same as for the microcontroller.

Matlab read the serial as file .txt and the data read in polling mode have been plotted in real-time on the GUI. Data have been sent through a Push Button specially designed to increase or decrease the position of 20 degrees, as shown in Figure 4.10 or to allow the DC Motor to rotate clockwise or counterclockwise, as shown in Figure 4.11. Two Push Button Start and Stop have been implemented to begin or, for safekeeping, to stop immediately the rotation of the motor even if it has not reached the target position (Figure 4.12).

Figure 4.10. Push Button used for increase or decrease the DC Motor to 45°

Figure 4.11. Push Button used for clockwise or counterclockwise rotation

Figure 4.12. Push Button used to Start and Stop the DC Motor

4.6.2 Code

Transmission

The code below shows the data sent from Matlab in a polling mode to Microcontroller, using two control characters "b" and "e", in order to guarantee a correct transmission.

```
while(1)
% True when Push Button of Decrease 45 is pressed
if(handles.tx_dec==true)
fopen(s);
fwrite(s,'d');
fclose(s);
handles.tx_dec=false;
guidata(hObject,handles)
end

% True when Push Button of Increase 45 is pressed
if(handles.tx_inc==true)
fopen(s)
fwrite(s,'i');
fclose(s)
handles.tx_inc=false
guidata(hObject,handles);
end

```

```
% True when Push Button of Counterclockwise is pressed
if(handles.counterclockwise==true)
fopen(s)
fwrite(s,'k');
fclose(s)
handles.counterclockwise =false
guidata(hObject,handles);
end

% True when Push Button of Clockwise is pressed
if(handles.clockwise==true)
fopen(s)
fwrite(s,'c');
fclose(s)
handles.clockwise=false
guidata(hObject,handles);
end

% True when Push Button of Stop is pressed
if(handles.stop==true)
fopen(s)
fwrite(s,'x');
fclose(s)
fopen(s)
fwrite(s,'x');
fclose(s)
fopen(s)
fwrite(s,'x');
fclose(s)
handles.tx_inc=false
guidata(hObject,handles);
break;
end
end
```

Listing 4.8. Tx data with GUI-Matlab

Reception

```
while(1)
% Read UART communication
fopen(s);
start = fread(s,1,'char');

% Char control
if(start == 'b')
% Read on UART Current, Velocity, Position Target and Position read
data(i,:) = fread(s,4,'int16');

% Trasmission finished whit char 'e'
ended = fread(s,1,'char');
if(ended == 'e')
% Print ok for debug
disp('ok')

% Plot Current data on GUI-Matlab
axes(handles.Current)
plot(data_position(:,1),'r');
grid on, hold on

% Plot Position data on GUI-Matlab
axes(handles.Position)
plot(data_position(:,2),'g');
grid on, hold on
axes(handles.Position)
plot(data_position(:,3)/24,'r');

% Plot Torque data on GUI-Matlab
axes(handles.Torque)
plot(data_position(:,4),'r');
grid on, hold on
i = i+1;
end
end

% Close UART communication
fclose(s);
```

Listing 4.9. Rx data with GUI-Matlab

Chapter 5

Tuning and Stability

5.1 Introduction

After the implementation part, the study of PI system was performed in terms of the overshoot, settling time and the error criterion.

Tuning a control loop is the adjustment of its control parameters to the optimum values for the desired control response. Stability (no unbounded oscillation) is a basic requirement, but beyond that, different systems have different behavior, different applications have different requirements, and requirements may conflict with one another.

5.2 Tuning

Designing and tuning a PI controller appears to be conceptually intuitive, but can be hard in practice, if multiple (and often conflicting) objectives such as short transient and high stability are to be achieved. PI controllers often provide acceptable control using default tuning, but performance can generally be improved by careful tuning, and performance may be unacceptable with poor tuning.

If the system must remain online, one tuning method is to first set K_i value to zero. Increase the K_p until the output of the loop oscillates, then the K_p should be set to approximately half of that value for a "quarter amplitude decay" type response. Then increase K_i until any offset is corrected in sufficient time for the process. However, too much K_i will cause instability.

In this phase, the correct parameters have been performed setting a different

gain for PI position controller, PI speed controller and PI current controller.

5.2.1 Test

First test

The fist value for gain variables were:

- $K_{p_p} = 1$
- $K_{i_p} = 0$
- $K_{p_s} = 1$
- $K_{i_s} = 0$
- $K_{p_c} = 1$
- $K_{i_c} = 0$

How shown in Figure 5.1 when all Integrative Gain was set to zero there was an overshot and is important for the position control, but the output oscillated without reach a target position. This system was unstable.

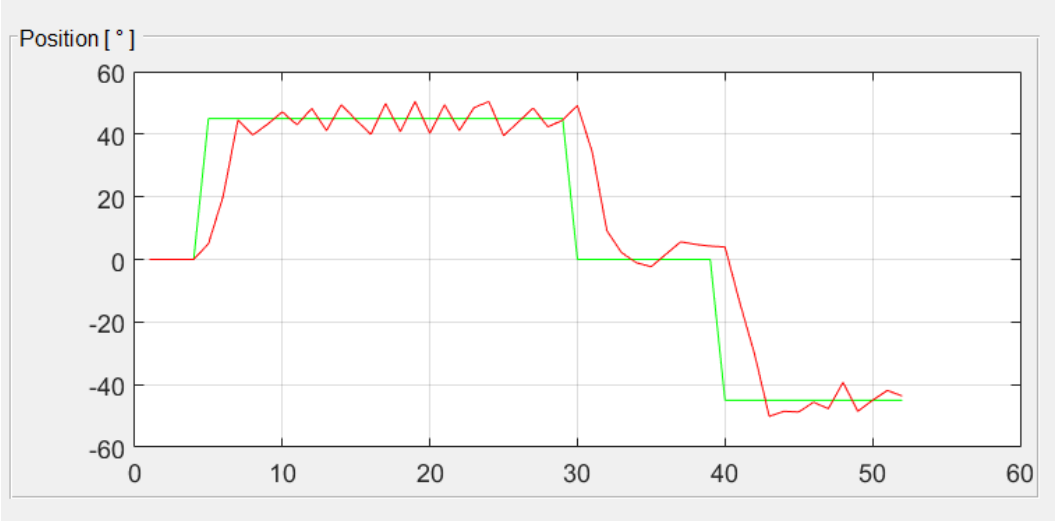

Figure 5.1. System with $K_{p_p} = 1$, $K_{i_p} = 1$,$K_{p_s} = 1$, $K_{i_s} = 1$, $K_{p_c} = 1$, $K_{i_c} = 1$

Second test

The second test was performed setting the integrative part equal to 0.2 and see how this parameter contributed to the system.

- $K_{p_p} = 1$
- $K_{i_p} = 0.2$
- $K_{p_s} = 1$
- $K_{i_s} = 0.2$
- $K_{p_c} = 1$
- $K_{i_c} = 0.2$

How shown in Figure 5.2 the system was unstable, the rise time was not conforme of the specific of the system.

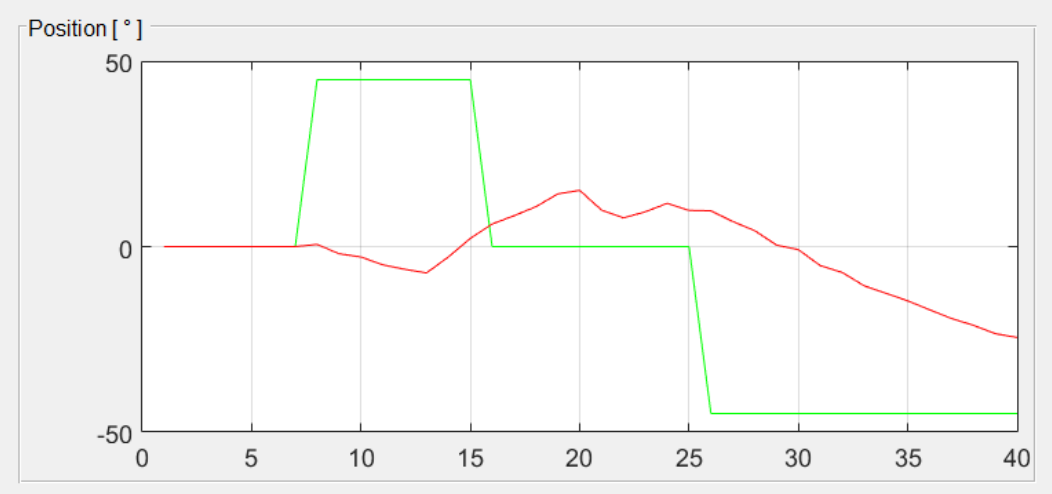

Figure 5.2. System with $K_{p_p} = 1$, $K_{i_p} = 0.2$, $K_{p_s} = 1$, $K_{i_s} = 0.2$, $K_{p_c} = 1$, $K_{i_c} = 0.2$

Third test

The third test was performed setting the proportional parameter equal to 2, in order to saw the output of the system.

- $K_{p_p} = 2$
- $K_{i_p} = 0.2$
- $K_{p_s} = 2$
- $K_{i_s} = 0.2$
- $K_{p_c} = 2$
- $K_{i_c} = 0.2$

How shown in Figure 5.3 the real position was 0, it did not follow the target position.

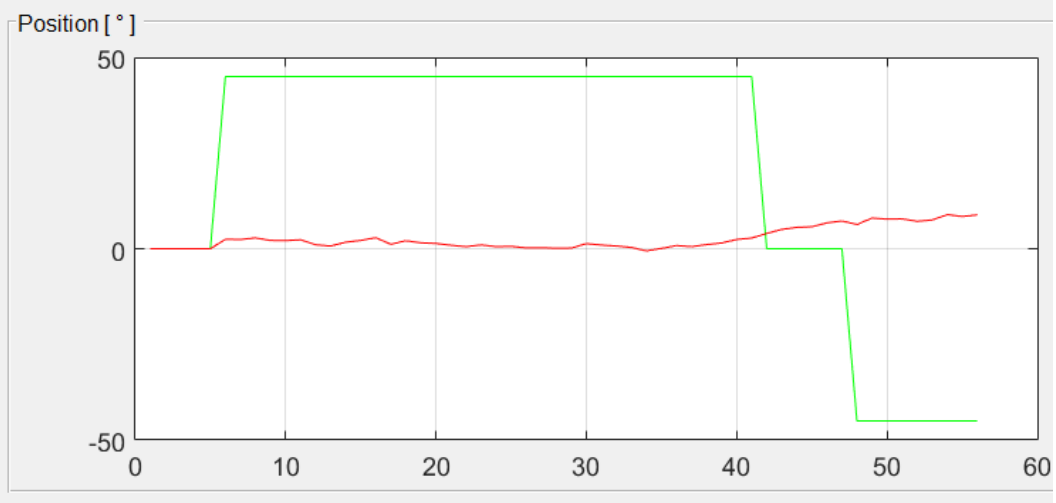

Figure 5.3. System with $K_{p_p} = 2$, $K_{i_p} = 0.2$, $K_{p_s} = 2$, $K_{i_s} = 0.2$, $K_{p_c} = 1$, $K_{i_c} = 0.2$

Fourth test

In this fourth test, the integrative parameters were changed divided by 10 the previous value.

- $K_{p_p} = 2$
- $K_{i_p} = 0.03$
- $K_{p_s} = 2$
- $K_{i_s} = 0.02$
- $K_{p_c} = 2$
- $K_{i_c} = 0.02$

How shown in Figure 5.4 the output followed the target position. The system was fast(rise time), but there was a high overshoot when the system changed the target position. The system was defined unstable because for the controller position the real position should not exceed the target position.

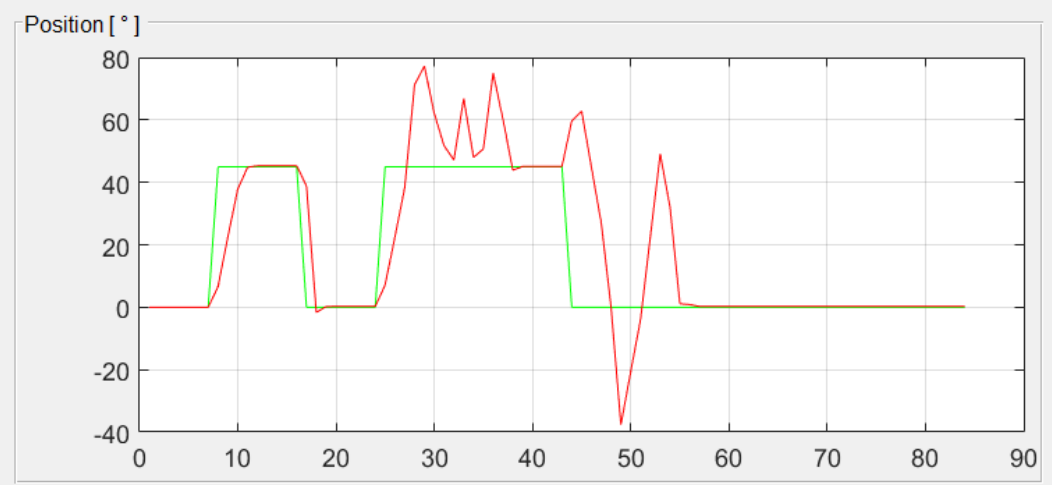

Figure 5.4. System with $K_{p_p} = 2$, $K_{i_p} = 0.03$, $K_{p_s} = 2$, $K_{i_s} = 0.02$, $K_{p_c} = 2$, $K_{i_c} = 0.02$

Fifth test

In the fifth test, the proportional gains were increased and the system was performed.

- $K_{p_p} = 3$
- $K_{i_p} = 0.03$
- $K_{p_s} = 2$
- $K_{i_s} = 0.02$
- $K_{p_c} = 4$
- $K_{i_c} = 0.02$

How shown in Figure 5.5 the system reached the desired position but the rise time was slow and there were too overshoot when the position was changed. The settling time was not reached. The system was defined as unstable.

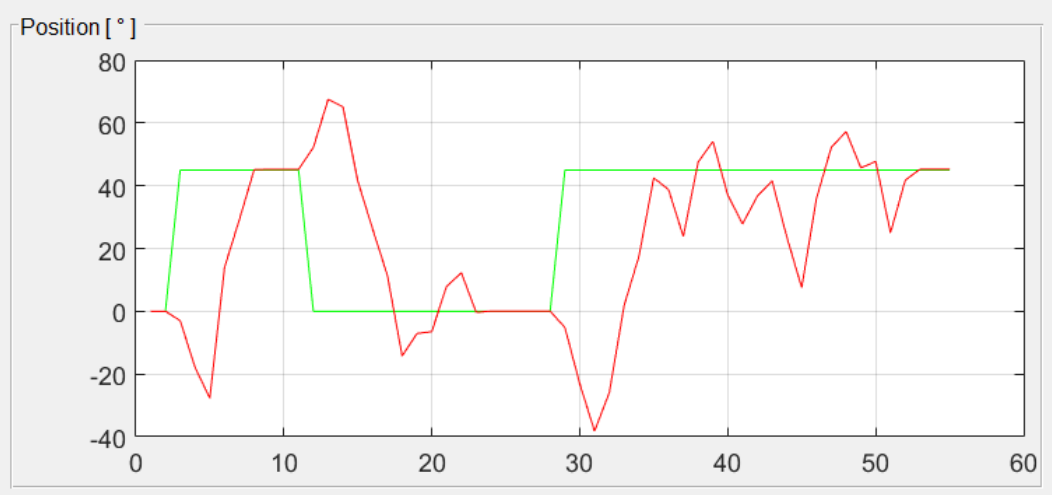

Figure 5.5. System with $K_{p_p} = 3$, $K_{i_p} = 0.03$, $K_{p_s} = 2$, $K_{i_s} = 0.02$, $K_{p_c} = 4$, $K_{i_c} = 0.02$

Sixth test

In this last test, the parameters of proportional gain were increased in order to reach the 90% of the target position with fast rising time and the integrative gain were decreased, one-tenth of the previous ones, so as not to have a very large settling.

- $K_{p_p} = 7$
- $K_{i_p} = 0.003$
- $K_{p_s} = 8$
- $K_{i_s} = 0.002$
- $K_{p_c} = 6$
- $K_{i_c} = 0.001$

How shown in Figure 5.8 the system is stable, there was no overshoot when the position was changed and the rising time was fast.

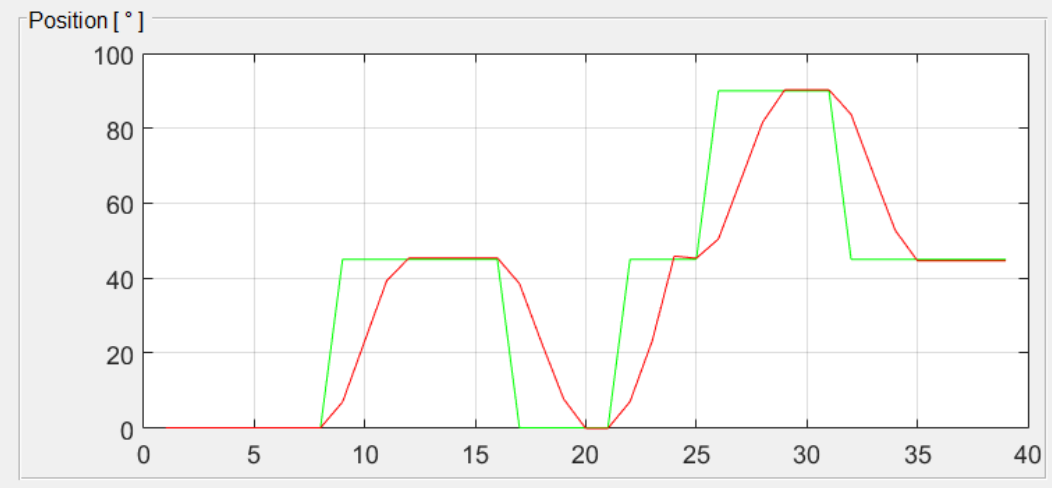

Figure 5.6. System with $K_{p_p} = 7$, $K_{i_p} = 0.003$, $K_{p_s} = 8$, $K_{i_s} = 0.002$, $K_{p_c} = 6$, $K_{i_c} = 0.001$

5.3 Stability

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e., its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Instability is caused by excess gain, particularly in the presence of significant lag.

Generally, stabilization of response is required and the process must not oscillate for any combination of process conditions and set-points, though sometimes marginal stability (bounded oscillation) is acceptable or desired.

5.3.1 Motion Profile

To improve the system a position profile has been developed, in order to reach the target position without or small peak overshoot, limiting the power of the DC Motor in case it has high inertia.

The position profile was done through the error between target position and real position, as shown in the Figure 5.7.

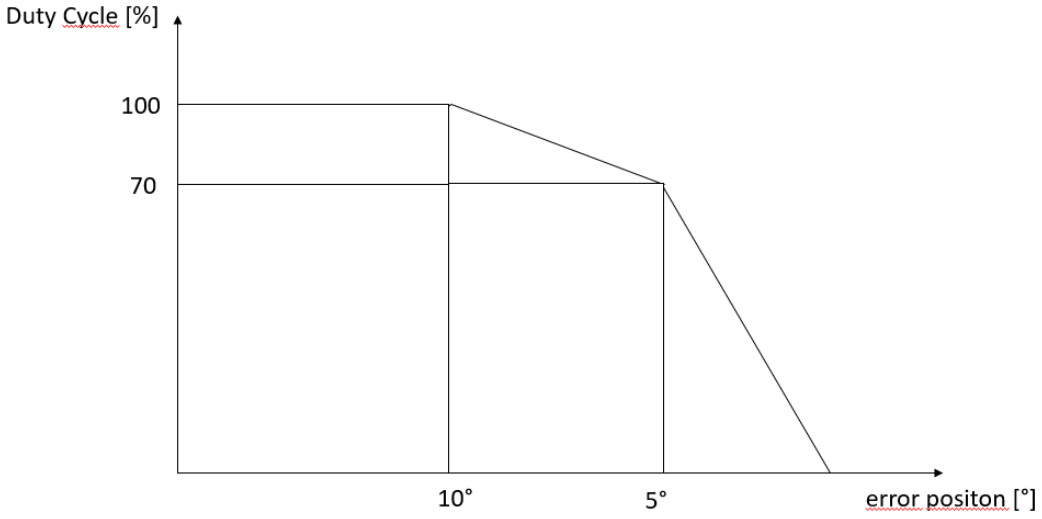

Figure 5.7. Motion Profile

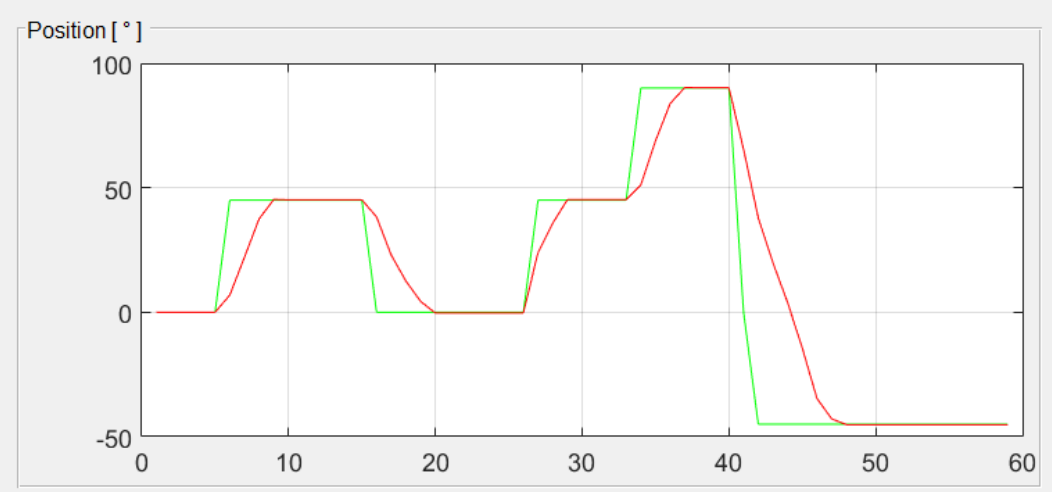

Figure 5.8. System with $K_{p_p} = 7$, $K_{i_p} = 0.003$,$K_{p_s} = 8$, $K_{i_s} = 0.002$, $K_{p_c} = 6$, $K_{i_c} = 0.001$

Chapter 6

Torque estimation

6.1 Introduction

The final step of the thesis was focused on torque estimation using a 3D-beam specially designed for motor shaft and a load cell to measure weight.

6.2 Differential Amplifier

Firstly, for a high accuracy a differential amplifier was designed for load cell(Figure 6.1), because the smallest detectable incremental change in voltage of the ADC in terms of LSBs is

$$1LSB = Vref/4096 = 805{,}6\mu V$$

and the sensibility of the load cell is the order of milliVolt.

This time, the operational amplifier has been used in mode STANDALONE, because the gain is equal to 1000 and the maximum amplification using an operational amplifier in PGA internally connected is equal to 32. The value of the differential amplifier are:

$$Vout = G(V1 - V2)$$
$$G = R1/R3$$
$$R3 = R2$$
$$R1 = R4$$

where $R1 = 100k\Omega$, $R2 = 100\Omega$ and the gain $G = 1000$. With this gain, the resolution of the ADC is guaranteed. For the correct torque estimation, because a variation voltage caused by noise is amplified by the gain of differential amplifier was make by software through an iteration of measurement, calculating the mean value force, as shown with the code below. The calibration of load cell generates a relationship between weight and ADC equal to 2.95 $gr.cm/LSB$.

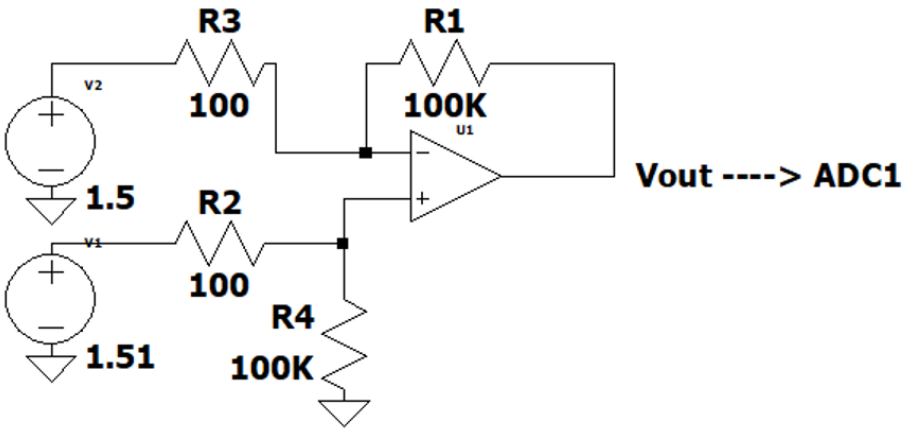

Figure 6.1. Differential Amplifier for torque estimation

Code

```
\* Iteretion for mean value of force *\
for(uint8_t i=0; i<20; i++)
{
HAL_Delay(1);
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
raw = HAL_ADC_GetValue(&hadc1);
Torque = voltage + raw;
}

\* Torque[gr.cm] = Voltage[LSB]*2.95[gr.cm/LSB] *\
Torque = (voltage/20)*2.95;
```

Listing 6.1. Mean value of load cell

The Figure 6.2 shown three functions estimation, which are Torque[$kg.cm$] - Current[mA], Torque[$kg.cm$] - Duty Cycle[%] and Current[mA] - Duty Cycle[%] in order to monitoring the data in real-time.

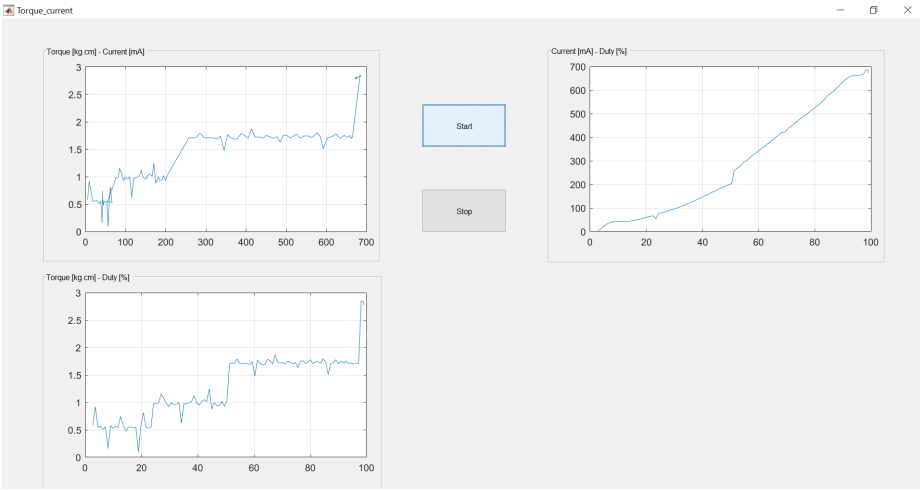

Figure 6.2. Real-Time data using GUI-Matlab for torque estimation

6.3 Estimation

The data of torque and current has been saved in Matlab workspace and the $T(I)$ function was calculated in order to monitor how much Torque the DC Motor is producing at any given moment. The function $T(I)$ has been linearized using the Ordinary Least Square calculating the torque constant (k_t) of DC Motor, as shown in the Figure 6.3. The coefficient has been calculated by the function polyval in Matlab and the torque constant ()k_t) was equal to 0.0035[$kg.cm/mA$].

The equation used for the torque function (T(I)) is:

$$T(I)[kg.cm] = k_t[\frac{kg.cm}{mA}]I[mA] = 0.0035[\frac{kg.cm}{mA}]I[mA]$$

To measure the torque, an ad hoc project had to be created, blocking the load cell on the wood on one side and the motor on the other side. The implementation took place with a calibration of the screws on the load cell, while for the motor a wooden thickness had to be added to ensure that the force exerted on the meter

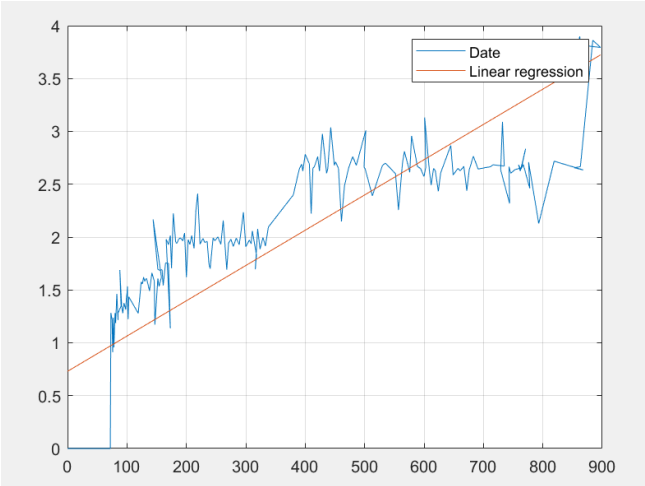

Figure 6.3. Workspace Matlab of torque(blue) and linear regression(red).

is perfectly perpendicular, so as to have an angle of 90 degrees and make null the value of sen in the vector product.

In Figure 6.4 is shown the measurement of the toque whit load cell, arranged at a distance of 10 cm and perfectly perpendicular to the force of the arm, using the following equations:

$$\vec{T} = \vec{F}\vec{r}$$
$$T = Frsin(\theta), \text{ with } \theta = 90°$$
$$T = Fr$$

where T is the torque, F is the force and r is the distance between load cell and motor shaft.

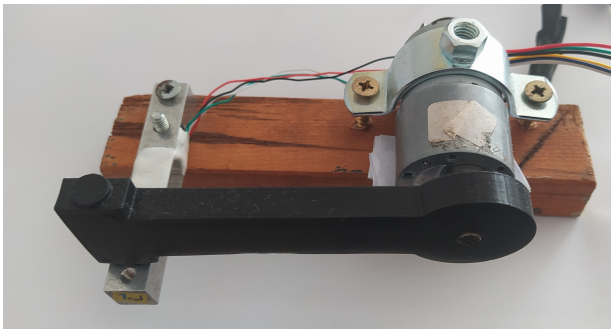

Figure 6.4. Torque estimation using a load cell and 3D-Beam.

6.4 Final Model

After the torque estimation and having reached the stability of the system by choosing the correct proportional(K_p) and integral(K_i) variables, the final project is shown in the Figure 6.5. The data of Position, Speed, Current and Torque were plotted in real-time.

In the Position graph, we can see in red the real position of DC Motor and in green the target position send by the push-button at the bottom-left. When the bar was blocked by an obstacle, the DC Motor tried to reach the target position and it increased the torque, as shown in the picture below.
At 60 data transmitted, the position was not reached and in torque plot, as well as in the current plot, there was a peak. Otherwise, in speed graph the velocity is zero.
When the target position was decreased, in order to rotate in the opposite direction to the obstacle, the revolution was in a nominal operating point.

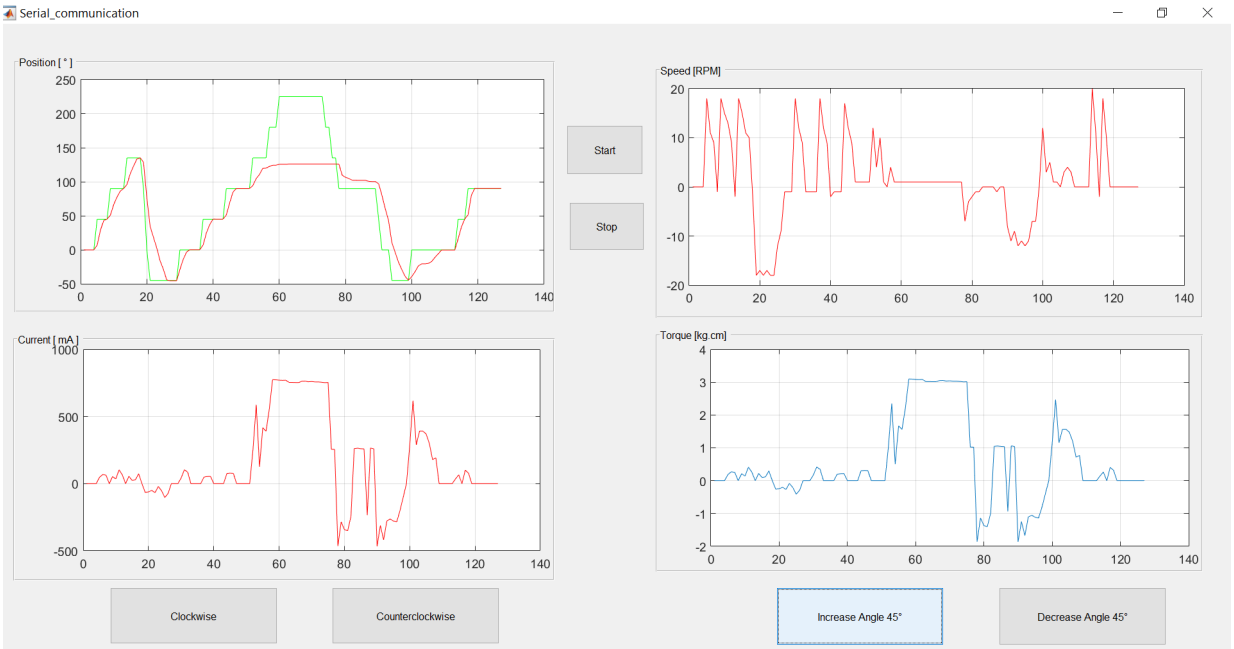

Figure 6.5. GUI-Matlab monitoring torque estimation in closed-loop position controller.

Chapter 7

Conclusion

7.1 Summary of the results and Future work

In this thesis was designed PI position controller for Brushed DC Motor and a torque estimation in closed loop.
A STM32-G431RB Nucleo Board and Shield X-Nucleo-IHM13A1 to control the motor and a graphical interface was used to monitoring the data.

The results of the experiment show that the system is efficient in terms of stability and robustness. Several tests were carried out by perturbing the system, in the first tests we wanted to analyze only each PI individually, in order to evaluate the correct functioning and implementation with external hardware, such as the welding of a jumper wires on the shield and an active low pass filter in order to filter the voltage ripples caused by slow decay. The velocity and position PIs were then analyzed respectively. The position control was the last feedback as it is the most important for the calibration of the position between that of target and that of position, in fact a motion profile had to be created, in order to never exceed the desired position if the DC motor had a larger self inertia. The estimate of the torque was carried out with a load cell, using a differential amplifier because the variation was of the order of milliVolts and the results obtained were satisfactory because there was no linearity for the calculation of the toque constant, as expected, but through the linearization of the data obtained it was possible to obtain a good estimate of the torque constant. Finally, torque control has been introduced throughout the system, in order to have an accurate estimate of how much torque is being produced at any given moment.

The project is versatile for other Brushed DC motor, which they use different voltage supplies, because the code has been written using define variables and the transmission in UART protocol and the GUI-Matlab could be used for other project without dependency of Brushed DC Motor, the transmission takes place following the protocol write in Matlab and the data will be plot directly on the graphical interface.

In the future we will try to improve the torque measurement, so that it is not in step and therefore we will not use the linear regression, but we will directly have a torque constant from the estimate. Finally, we will try to develop a code for a customized board in which a Brushless DC motor will be controlled without the use of a shield, directly driving the six MOSFETs of a three-phase inverter.While the graphics in which to monitor the data in real time will be developed in ROS interface.

```
while (1)
{
  if (uwStep == 0)
  {
    //Safety step. All OFF!
    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
    HAL_TIMEx_OCN_Stop(&htim1, TIM_CHANNEL_1);

    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
    HAL_TIMEx_OCN_Stop(&htim1, TIM_CHANNEL_2);

    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
    HAL_TIMEx_OCN_Stop(&htim1, TIM_CHANNEL_3);

    HAL_Delay(100);
    uwStep = 1;
  }
  if (uwStep == 1)
  {
/* Next step: Step 2 Configuration ---------------- */

    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, uPWM);
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
```

```
24      HAL_TIMEx_OCN_Stop(&htim1, TIM_CHANNEL_1);
25
26      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 4249);
27      HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
28      HAL_TIMEx_OCN_Start(&htim1, TIM_CHANNEL_2);
29
30      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, uPWM);
31      HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
32      HAL_TIMEx_OCN_Stop(&htim1, TIM_CHANNEL_3);
33
34      HAL_Delay(100);
35      uwStep++;
36    }
37
38    if (uwStep == 2)
39    {
40      /* Next step: Step 2 Configuration
       ------------------------------------ */
41
42      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, uPWM);
43      HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
44      HAL_TIMEx_OCN_Stop(&htim1, TIM_CHANNEL_1);
45
46      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 4249);
47      HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
48      HAL_TIMEx_OCN_Start(&htim1, TIM_CHANNEL_2);
49
50      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, uPWM);
51      HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
52      HAL_TIMEx_OCN_Stop(&htim1, TIM_CHANNEL_3);
53
54      HAL_Delay(100);
55
56      uwStep++;
57    }
58    else
59    {
60  /* Next step: Step 1 Configuration ---------------------- */
61  //Do nothing;
62  HAL_Delay(100);
63  uwStep = 0;
64  }
```

```
65 /* USER CODE END WHILE */
66
67 /* USER CODE BEGIN 3 */
68 }
69 /* USER CODE END 3 */
70 }
```

Listing 7.1. C code for Brushess DC

Chapter 8

References

- STMicroelectronics, "Integrated Development Environment for STM32" https://www.st.com/en/development-tools/stm32cubeide.html
- STMicroelectronics, "Low voltage brush DC motor driver expansion board for STM32 Nucleo based on the STSPIN250", https://www.st.com/en/ecosystems/x-nucleo-ihm13a1.html
- STMicroelectronics, "Mainstream Arm Cortex-M4 MCU 170 MHz with 128 Kbytes of Flash memory, Math Accelerator, Medium Analog level integration ", https://www.st.com/en/microcontrollers-microprocessors/stm32g431rb.html#documentation
- STMicroelectronics, "STM32G431x6 STM32G431x8 STM32G431xB", Reference manual.
- STMicroelectronics, "STSPIN250 Low voltage brush DC motor driver."
- STMicroelectronics, "X-NUCLEO-IHM13A1 Low voltage brush DC motor driver expansion board for STM32 Nucleo based on the STSPIN250", Reference manual.
- STMicroelectronics, "Description of STM32G4 HAL and low-layer drivers", User manual
- Microchip, "Brushed DC Motor Basics", https://www.microchip.com/stellent/groups/SiteComm_sg/documents/DeviceDoc/en543041.pdf

- Wikipedia, "Brushed DC electric motor", https://en.wikipedia.org/wiki/Brushed_DC_electric_motor
- MathWork, "Create apps with graphical user interfaces in MATLAB", https://www.mathworks.com/discovery/matlab-gui.html
- Ni, "PID Theory Explained", https://www.ni.com/it-it/innovations/white-papers/06/pid-theory-explained.html/#section-818077577
- Wikipedia, "PID controller", https://en.wikipedia.org/wiki/PID_controller#Loop_tuning
- Lancet, "D.C. Motor Torque/Speed Curve Tutorial" http://lancet.mit.edu/motors/motors3.html
- STMicroelectronics, "An introduction to electric motors.", User manual.
- STMicroelectronics, "Current Sensing in motion control applications". Application note.

Chapter 9

Appendix

STM32CubeIDE

TIM16 PWM

Timer 16 was used to create PWM for motor power control. Frequency has been set to $170MHz$, Prescaler has been set 0 and Counter has been set 2200.

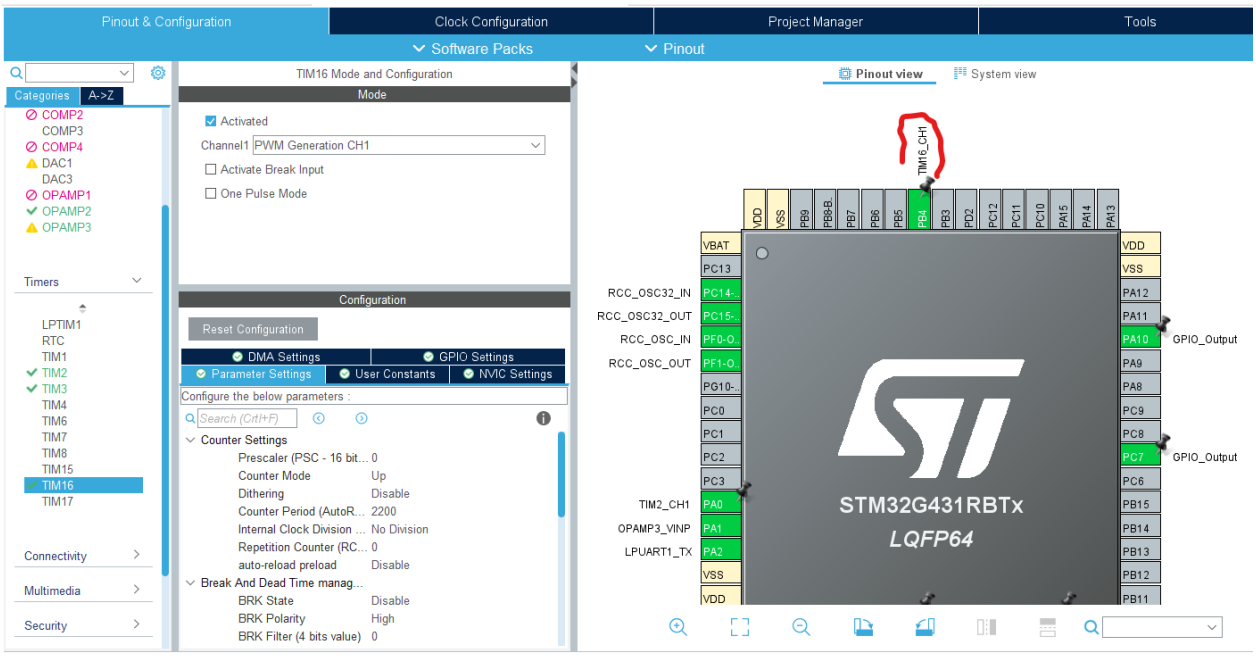

Figure 9.1. TIM16- PWM DC Motor

TIM3 PWM

Timer 3 has been set on channel 2 as input capture to calculate the encorder speed. Frequency has been set to $170MHz$, Prescaler has been set to 170 and the Counter has been set $0xffff$.

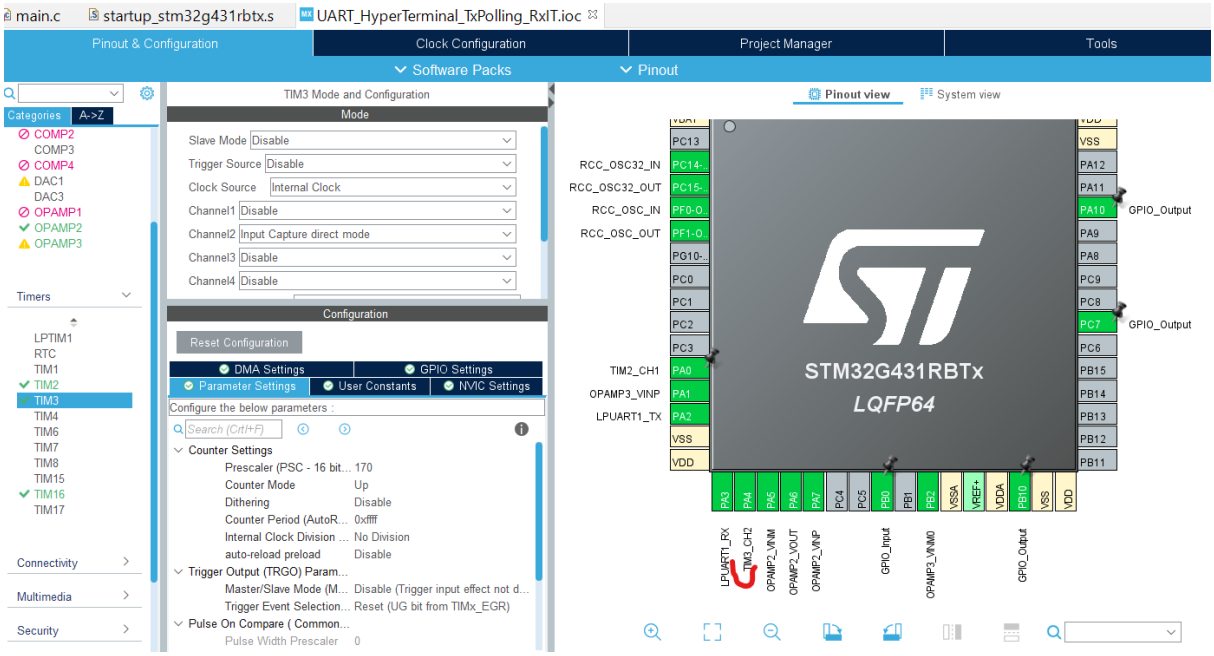

Figure 9.2. TIM3 - Input Capture Encoder

ADC2 and OPAMP3

ADC2 was internally connected with OPAMP3 for amplification and measurement of current. ADC2 has been set as indipindent mode and resolution equal to 4096, while the amplification of operation amplifier has been set to 8.

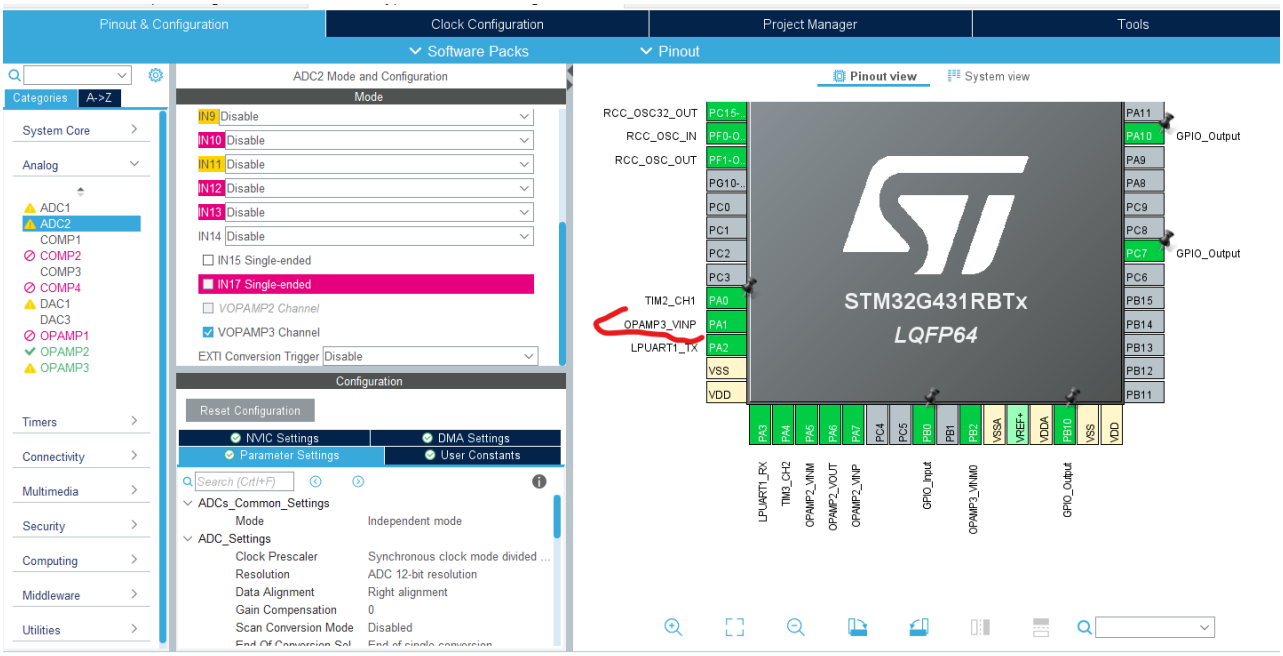

Figure 9.3. GUI-Matlab

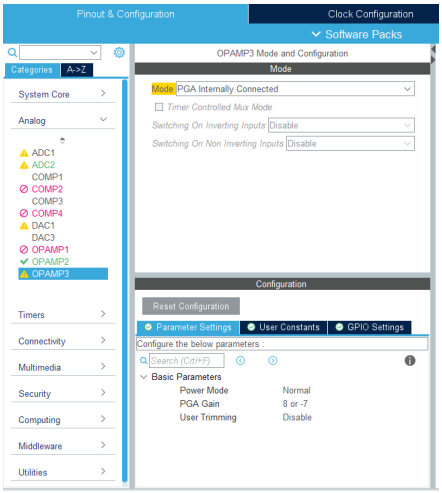

Figure 9.4. ADC2 and OPAMP3 for current measurement

GPIO

Four pins have been used in GPIO mode for the following functions:

- GPIO_PA10: Used in Output mode to ENABLE the shield;
- GPIO_PB0: Used in Input mode to calculate the encoder position;
- GPIO_PB10: Used in Output mode to reset the shield;
- GPIO_PC7: Used in Output mode to set the direction of the motor.

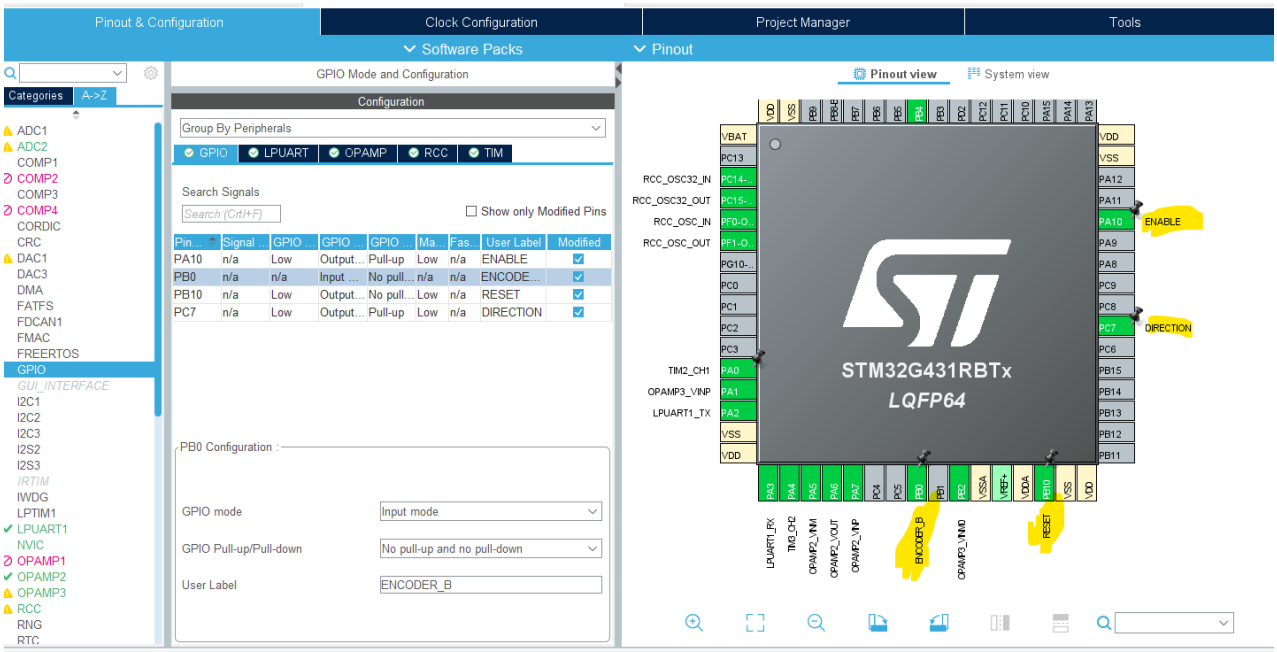

Figure 9.5. GPIO Pin