Control in effort of a robotic hand

2017 / 2018

Veronica Elena

07/03/2018

First Master Thesis Report

Useful links:
https://www.alpes-instruments.com
https://www.tekscan.com
https://www.maxonmotor.com/maxon/view/content/index

Chapter 1

Electronics of the robotic hand

This chapter has the aim of explaining and summarizing the main working principles of the Alpes Instruments robotic hand.
All information and pictures come from the documentation provided by the company.

1.1 General overview

The electronic scheme of the robotic hand is represented in (Figure 1.1). It is a simplified version which is useful to understand the basic functionality of the robotic device.

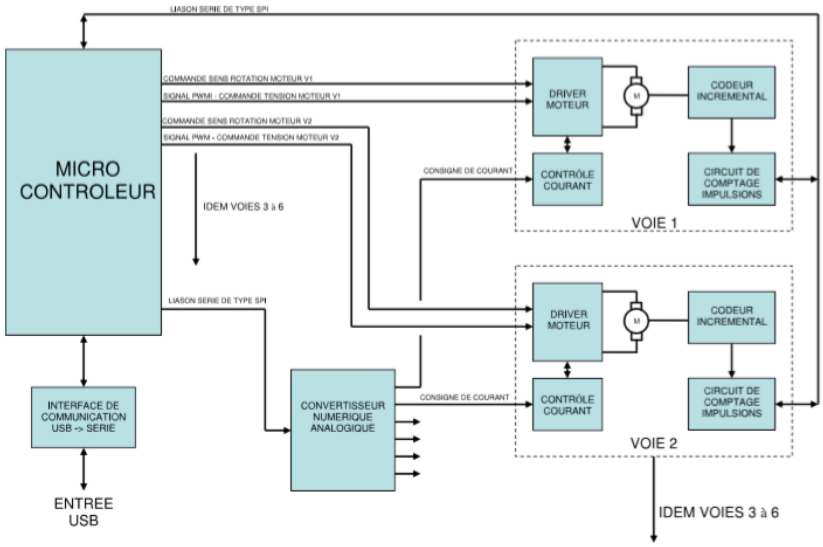

Figure 1.1: Electronic scheme of the robotic hand.

From the electronic point of view, the robotic hand is composed of a main ARM7 microcontroller which works to 72 Mhz. It represents the interface between the PC and the different components to send commands to the actuators. It can be programmed in C language and it works with two modes: the manual mode and the position instructions one. Using the first mode, the PC sends the tension value, the polarity or the current limit value to the microcontroller which applies the value to the motors directly. Using the second mode, the PC sends the desired position to the microcontroller which runs an algorithm with the aim of maintaining each motor to the given position. It is possible to interact with the microcontroller via a interface circuit USB-COM.
There exist six encoder pulse counting circuits, one for each incremental encoder associated to

each motor, and they are seen by the main microcontroller as peripheral components which communicate through a SPI connection. The readings of the circuits occur every 166 microseconds.
Since the considered robotic hand has six motors, at electronic level six circuits that control the actuators are present. They have two logic inputs: "ENABLE" and "SENS". The former input is set to 0 when a null tension is applied to the motor, while it is 1 to apply the tension on the motor. This entrance is connected to the microcontroller. The latter input defines the direction of motor rotation.
In order to convert the digital information coming from the microcontroller to the analogical tension to be applied to the actuator circuits, a DAC with 8 outputs is used. Each output tension can be independently modified from the PC. The converter component and the microcontroller communicate via a serial SPI connection.
Finally, a current control system allows to establish the current on the actuator progressively as soon as the tension is applied on it. Given a threshold current, once the current on the motor reaches the fixed one the corresponding tension is switched off. In this way an average current is obtained. In order to do that a comparator is used.

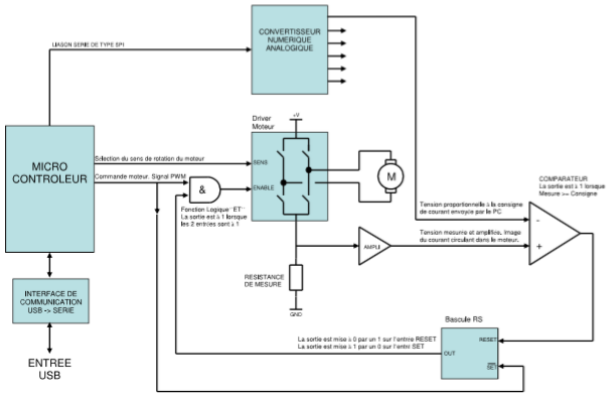

Figure 1.2: Current control scheme of the robotic hand.

1.2 Communication protocol - COM

Each motor is mechanically coupled to an incremental encoder. Their union is called channel. The channels are numbered from 0 to 5 corresponding to the fingers from the little one to the thumb respectively. In the same way, the registers are numbered from 0 to 41[1]. At each channel is assigned a set of registers defining its operations. The access to a particular register of a given channel is via its memory location which is calculated as follows:

$$((Number of channel + 1) \cdot 1000) + Number of Register. \tag{1.1}$$

The robotic hand incorporates a component that interfaces between a USB port of the PC and a UART port of the microcontroller so that the link is seen from both sides as a serial link of COM port type. The communication speed is 460,800 bauds. The communication between the PC and the hand is based on readings and writings of registers. It is always initiated by the PC sending a request frame to which the hand responds with a response frame as the one presents in (Figure 1.3). The communication string always starts with a code identifying the request type. In the following a list of the commands is reported:

[1]see *Alpes Instruments*' documentation

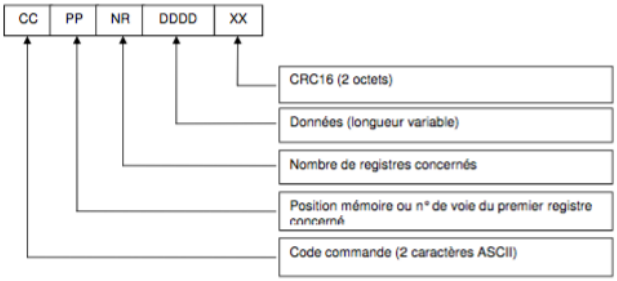

Figure 1.3: Communication string template.

- RD: read n registers starting from the certain memory position.
- WR: write on n registers starting from the certain memory position.
- W1, W2, W3: write respectively on "MODE_CMD_MOTEUR", "CONSIGNE_TENSION_POSITION" and "LIMITE_COURANT" registers of the n consecutive channels and reading of the "POSITION_CODEUR" register for the same n channels.
- W4, W5, W6: write respectively on "MODE_CMD_MOTEUR", "CONSIGNE_TENSION_POSITION" and "LIMITE_COURANT" registers of the n consecutive channels and reading of the "VITESSE_MOTEUR" register for the same n channels.

The commands RD and WR make possible to read or write a series of registers by indicating the memory position of the first register to be read or written. The commands W1 to W6 perform reading and writing tasks in a single exchange. In general, they allow to write one of the registers of n consecutive channels and to obtain in response one of the registers of these channels. All digital values are transmitted directly using hexadecimal coding starting from the least significant byte and going up to the most significant one. Transmissions are secured by adding a CRC16 at the end of the string. Upon receipt of a string, the microcontroller verifies that:

- The received CRC16 is identical to the one it recalculated from the bytes received;
- The order code is known;
- The start and end positions are contained in its memory area;
- The requested command is allowed in the specified memory area.

If all these conditions are satisfied, the string is taken into account and a response is returned. The microcontroller detects the end of a frame when it no longer receives data during 100 microseconds. When no string is received from the PC for 500 ms, the channels configured to operate in voltage setpoint mode are automatically disabled. The registers "MODE_CMD_MOTEUR" and "CONSIGNE_TENSION_POSITION" reset to 0.

1.3 Internal registers

The robotic hand is controlled from the PC. It can operate following 2 distinct modes. The choice of the operating mode is made by writing specific values in a register named "MODE_CMD_MOTEUR". The operating mode can be configured independently from one channel to another.

Position setpoint mode In this mode, the PC writes the position setpoints to the register named "SET_TENSION_POSITION" of each channel. The microcontroller will then place and maintain each motor at the indicated position using the following algorithm:

- Reading of the current position "POSITION_CODER";
- Calculation of the current deviation:

  $$ECART\_POSITION = CONSIGNE\_TENSION\_POSITION – POSITION\_CODEUR;$$

- Calculation of the cumulation of the differences:

  $$SOMME\_ECARTS = SOMME\_ECARTS + ECART\_POSITION;$$

- Calculation of the variation of the difference:

  $$DELTA\_ECARTS = ECART\_POSITION - MEMO\_ECARTS;$$

- Calculation of the voltage to be applied to the motor

  $$SORTIE\_PWM = CALCUL\_P + CALCUL\_I + CALCUL\_D$$

  $$CALCUL\_P = ECART\_POSITION \cdot (\frac{COEF\_P}{1000})$$

  $$CALCUL\_I = POS\_ERROR\_SUM \cdot (\frac{COEF\_I}{10000})$$

  $$CALCUL\_D = DELTA\_ECARTS \cdot (\frac{COEF\_D}{100})$$

- Apply the new voltage to the motor.

The above operations are repeated once per ms for each channel and at 166.6 microseconds interval between two consecutive channels. It is important to notice that the home position is set to 0 when the finger is fully open and increases when the finger closes. Moreover, the voltage actually applied to the motor is limited in negative to the value of the parameter "MIN_SORTIE_PWM" and in positive to that of "MAX_SORTIE_PWM". The value of the "OUTPUT_PWM" register is not modified when these limits are applied. The value of "SUM_ECARTS" is limited in negative with the value of parameter "MIN_SOMME_ECARTS" and in positive with that of "MAX_SOMME_ECARTS".

Tension mode Using this modality, the PC writes the desired tension to be applied on the motor on the "CONSIGNE_TENSION_POSITION" register of each channel. The microcontroller applies the tension on the motor directly. Applying a negative tension the hand opens, while with a positive tension it closes.

Chapter 2

Functions

This section presents and organised the functions which are used in order to access to the information and to the memory position useful to interact with the robotic hand functionalities. The functions are organised in three main categories: the *Basic functions* to perform the basic operations to interact with the hand, the *Reading functions* which allow to read some information on the hand's registers and the *Writing functions* used to write on registers.
The Matlab library has been created by Benjamin Gautier, a master student during 2014/2015.

2.1 Basic functions

initialization.m This function initialize the connection with the robotic hand writing 1 in the register at address 100. Considering the resting position as the configuration in which the hand is completely open, when the function is called the hand performs the initialization procedure. It consists in completely rotating the thumb and putting it back to the initial position, completely flexing the thumb and moving it back to the rest position and doing the same with the other four fingers.

initializationtest.m It checks if the initialization procedure has been corrected reading the register which is at address 100. If the value contained in this position is 1, the initialization is occurred correctly.

arret_initialisation.m It allows to stop the initialization procedure writing the 0 value on the initialization register (memory position 100).

```
function arret_initialisation (serial_port)
```

CRC16.m This function checks if the communication between the PC and the microcontroller is correct. It takes as input the command string (buf) containing two bytes identifying the operation, two bytes for the memory position of the register, 2 bytes for the register number and 4 bytes containing the data to be transmitted. The function returns the ..
The CRC (Cyclic redundancy check) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption.

```
function [crc16hi, crc16lo] = CRC16 (buf)
```

2.2 Reading functions

lecture_position.m It reads the encoder position for the selected finger. It takes as inputs an integer number corresponding to the selected finger and the serial port associated to the hand object. The output is the value of the position in decimal scale. The function accesses to the register 26 (POSITION_CODEUR) which contains the position information sending the reading command to the robotic hand. The position information is contained between the 7th and 10th hex words. Beside the position information in hand unit, the function also returns the encoder position in radiant. Geometrically the following equation is valid:

$$position\_radiant = \frac{R}{R_m}((\frac{\pi}{2} - q_1) + (2\pi - q_2) + (2\pi - q_3)), \tag{2.1}$$

where **q** is the vector of the position of the joint coordinates, R and R_m are the pulley radii and the radius of the motor shaft respectively.

```
function [position, position_radiant] = lecture_position(serial_port, finger)
```

lecture_vitesse.m This function gets the velocity information from the register 27 (VITESSE_MOTEUR). As the previous one, it takes as input an integer number corresponding to the selected finger and the serial port associated to the hand object. The output is the value of the velocity in decimal scale.

```
function vitesse = lecture_vitesse(finger, serial_port)
```

lecture_couple.m It gets the motor torque information using the knowledge of the current given by the corresponding register 2 (LIMITE_COURANT). As the previous two cases, it takes as input an integer number corresponding to the selected finger and the serial port associated to the hand object.
From the documentation related to the Maxon motor specifications, the nominal value of the motor torque (Γ_n) is 2.03 mNm, while the current one (C_n) is 0.211 A. The stall torque (Γ_i) has a value of 4.38 mNm and the starting current (C_i) is 0.438 A. Thus the motor's torque constant K_t can be defined:

$$K_t = \frac{(\Gamma_i - Gamma_n)}{(C_i - C_n)} = 10.35.$$

The output of the function is the motor torque value (Γ):

$$\Gamma = real\_current \cdot K_t.$$

Moreover, from the documentation it comes out that the current values read from the register is expressed for points. In order to obtain the current value at the comparator entrance expressed in Ampere, the following equation is valid:

$$real\_current = \frac{(limit\_current \cdot 3.3)}{65535}.$$

```
function couple_doigt = lecture_couple (finger, serial_port)
```

lecture_puissance.m This function compute the power value of the considered finger. It takes as input an integer number corresponding to the selected finger and the serial port associated to the hand object.
Using the previous defined function, it gets the velocity (v) and torque (Γ) values and it return the finger power (W) as:

$$W = \Gamma v.$$

```
function Puissance = lecture_puissance (finger, serial_port)
```

read_all_positions.m This function is equal to lecture_position.m one, but it gets the position values for all fingers.

read_all_speeds.m This function is equal to lecture_vitesse.m one, but it gets the velocity values for all fingers.

Lecture_coeffs.m This function gets the PID controller parameters from the COEF_P, COEF_I and COEF_D registers respectively. Moreover, it also reads the value of the limit current from the namesake register.

```
function [P, I, D, limit_current] = lecture_coeffs (serial_port)
```

lecture_coef_p.m This function accesses to the register number 8 in order to read the value of the PID controller proportional coefficient (P). The register's name is COEF_P.

```
function P = lecture_coef_p (finger, serial_port)
```

lecture_coef_i.m It reads the Integral coefficient (I) of the PID controller contained in the 9th register. The name of the register is COEF_I.

```
function I = lecture_coef_i (finger, serial_port)
```

lecture_coef_d.m This function reads the derivative coefficient (D) of the PID controller. The value is contained in the register 10 which name is COEF_D.

```
function D = lecture_coef_d (finger, serial_port)
```

lecture_calcul_p.m This function reads from the register 35 (CALCUL_P) the proportional term of the position control (Equation 1.3). The function takes as input the serial port associated to the hand and the integer number associated to the considered finger.

```
function calcul_p = lecture_calcul_p (finger, serial_port)
```

lecture_calcul_i.m This function reads from the register 36 (CALCUL_I) the proportional term of the position control (Equation 1.3). The function takes as input the serial port associated to the hand and the integer number associated to the considered finger.

```
function calcul_i = lecture_calcul_i (finger, serial_port)
```

lecture_PWM.m This function reads the value contained in the register 33 (SORTIE_PWM). The Pulse-width modulation (PWM) is a modulation technique mainly used to control of the power supplied to electrical devices. The PWM output represents the voltage applied to the motor before a stop occurs (Equation 1.3). Its value is expressed in PWM points and it is bounded between MIN_SORTIE_PWM and MAX_SORTIE_PWM values.
The tension in Volt can be computed as follow:

$$Tension = \frac{12}{4095} \cdot SORTIE\_PWM$$

```
function [PWM, tension] = lecture_PWM (finger, serial_port)
```

lecture_tempo_mode_pi.m This function accesses to the 34th register (TEMPO_MODE_PI). It contains the remaining time before ignoring CALCUL_D for calculating SORTIE_PWM.

```
function tempo_mode_pi = lecture_tempo_mode_pi (finger, serial_port)
```

main_droite_gauche.m It determine which is the used hand reading the register 24 (ID_DROITE_GAUCHE). If the read value is 1 the right hand is used, otherwise if the value is equal to 2 the left hand is used. The function takes as input the serial port associated to the hand and the *name* variable which is the handle of the graphics object where the information will be shown. Indeed the function is a void one.

```
function main_droite_gauche (name, serial_port)
```

2.3 Writing functions

mouv_doigts.m This function takes as input an integer number corresponding to the selected finger, the serial port associated to the hand object and the position value to move the considered finger. Its aim is to move a specific finger to the absolute given position. In order to do that, the Position mode is chosen writing 1 on the register 0 (MODE_CMD_MOTEUR) and the position value for one finger is sent to the register 1 (CONSIGNE_TENSION_POSITION).

```
function mouv_doigts (position, finger, serial_port)
```

close_motion_prop.m This function is equal to mouv_doigts.m one, but it applies a given position to each fingers.

mouv_main.m This function is equal to close_motion_prop.m one, but it moves all fingers such that the hand is completely closed.

back_main This function is equal to close_motion_prop.m one, but it moves all fingers such that the hand is completely opened. This configuration of the hand represents the reference one in which all the fingers positions are zero.

ecriture_limite_courant.m It sets a limit value to the current writing on the LIMITE_COURANT register which is the second one. The function takes as input an integer number corresponding to the selected finger, the serial port associated to the hand object and the value of the limit current applied to the considered finger. The latter parameter has to be within the interval from 0 to 0.075 A.

```
function ecriture_limite_courant (finger, current_value, serial_port)
```

ecriture_limite_courant_tous_doigts.m This function is equal to ecriture_limite_courant.m one, but it sets the limit current value to all fingers.

ecriture_coef_p.m This function accesses to the register number 8 in order to write the value of the PID controller proportional coefficient (P). The register's name is COEF_P. The function takes as input the integer number corresponding to the considered finger, the serial port associated to the hand object and the value of the P coefficient.

```
function ecriture_coef_p (finger, P, serial_port)
```

ecriture_coef_i.m This function accesses to the register number 9 in order to write the value of the PID controller integral coefficient (I). The register's name is COEF_I. The function takes as input the integer number corresponding to the considered finger, the serial port associated to the hand object and the value of the I coefficient.

```
function ecriture_coef_i (finger, I, serial_port)
```

ecriture_coef_d.m This function accesses to the register number 10 in order to write the value of the PID controller derivative coefficient (D). The register's name is COEF_D. The function takes as input the integer number corresponding to the considered finger, the serial port associated to the hand object and the value of the D coefficient.

```
function ecriture_coef_d (finger, D, serial_port)
```

mode_tensionDoigt.m It writes on the register 0, 1 and 2 in order to set the Tension Mode, the tension value and the current value respectively. These last two values are passed to the function with the integer number corresponding to the considered finger, the serial port associated to the hand object.
The tension has a value between [-1150, +1150], which is equivalent to $\pm 11,50$ Volt. The bound is given by the motor specification, indeed the motor has a maximum applicable tension of 12 Volt. The current value for each finger belongs to the interval [0, 750]. The upper bound represents also the default value of the current contained within the 3rd register (LIMITE_COURANT_DEFAUT).
From the practical point of view, when this function is called, the tension is applied on the motor for 500 ms unless another instruction pre-empt it. After this time interval the 0 register is set to *Stop Mode*.

```
function mode_tensionDoigt (tension, courant, finger, serial_port)
```

Chapter 3

DC motor

3.1 General principle of a DC motor

A DC motor is a device that converts electrical energy (direct current system) into mechanical energy. It is made up of 4 basic components: the stator, the rotor (or armature), brushes and the commutator (Figure 3.1). The stator generates a stationary magnetic field that surrounds the rotor and it is generated by either permanent magnets. The rotor is made up of one or more windings.

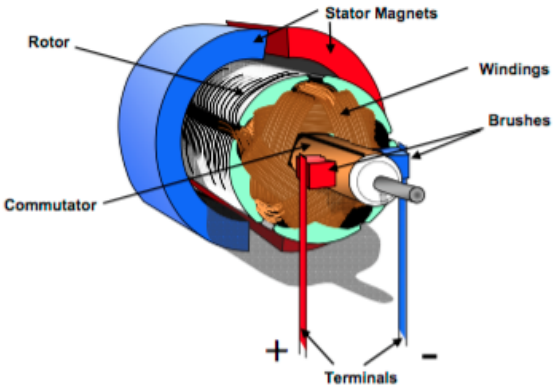

Figure 3.1: Brushed DC motor with all components labelled.

As soon as we supply direct current in the armature, a mechanical force acts on it due to electromagnetic effect of the magnet. If the direction of current in the wire is reversed, the direction of rotation also reverses accordingly to the Fleming's left hand rule.
This behaviour is based on the Faraday's law: if a conductor of length L carrying a current I is placed in a magnetic field **B**, a magnetic force **F** is created such that:

$$F = BLIsin\alpha,$$

where α is the included angle between **B** and I. The (Figure 3.2) explains this principle.

Since the two wire conductors are separated by some distance (w), the two opposite forces produces a rotational force or a torque that results in the rotation of the armature conductor. The expression of the torque is given by the two following equations:

$$\tau = Fcos\theta \wedge w$$

$$\tau = BILwcos\theta$$

Where θ is the angle between the plane of the turning armature and the reference plane which is the one along the direction of permanent magnetic field. Therefore, when θ is equal to 90^o

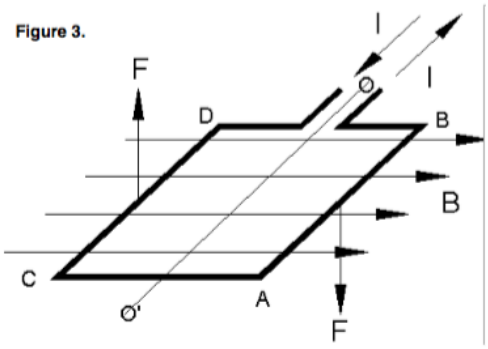

Figure 3.2: Faraday's law application.

the torque is zero. Thus the continuous rotation is achieved by reversing the direction of the current flow in the wire. This operation is performed by a switching device called "Commutator".

A DC motor's output torque τ is directly proportional to the current through the windings:

$$\tau = K_t I,$$

where K_t is the torque motor constant $[\frac{Nm}{A}]$ and I is the current.

3.2 Maxon DC motor

Alpes Instruments robotic hand is equipped with six Maxon DC motors (DCX10L EB KL 12V), one motor for each finger and two motors in the thumb. The motor specifications are available in the documentation provided by the *Alpes Instruments company*.
From the documentation related to the Maxon motor specifications, the nominal value of the motor torque (Γ_n) is 2.03 mNm, while the current one (C_n) is 0.211 A. The stall torque (Γ_i) has a value of 4.38 mNm and the starting current (C_i) is 0.438 A. Thus the motor's torque constant K_t can be defined:

$$K_t = \frac{(\Gamma_i - \Gamma_n)}{(C_i - C_n)} = 10.35.$$

Chapter 4

Force sensor

4.1 The FlexiForce A201 sensor

FlexiForce force sensors are ultra-thin (about 0.203 mm) and flexible printed circuits, which can be easily integrated into force measurement applications. The sensor acts as a force sensing resistor in an electrical circuit. When the force sensor is unloaded, its resistance is very high. When a force is applied to the sensor, this resistance decreases.

Figure 4.1: The FlexiForce sensor.

4.2 The FlexiForce board

The FlexiForce Quickstart Board is an analog circuit intended to act as an interface between the FlexiForce sensors and a circuit or data acquisition system. Basically, it is a Signal Conditioning Circuit which manipulate an analog signal in such a way that it meets the requirements of the next stage for further processing. Most common use is in analog-to-digital converters (ADC).
Signal conditioning can include amplification, filtering, converting, range matching, isolation and any other processes required to make sensor output suitable for processing after conditioning. The (Figure 4.2) shows how the sensor should be connected to the board. In this configuration the sensor output is processed by the board. The sensor has three pins: the first one carries the data, the second one provides the alimentation, while the third one is empty. The order in which the sensor pins are attached to the board does not modify the result. (Figure 4.3) explains the meaning of the board connections. It is important to notice that V_{in} corresponds to the reference input (GND) and the third connection brings the final measured force information. In both figures it is possible to see a little blue box which is a potentiometer. Changing the value of the feedback resistor adjusts the full scale of the force range. As the value of the feedback resistor increases, the maximum measurable force before

Figure 4.2: Sensor configuration.

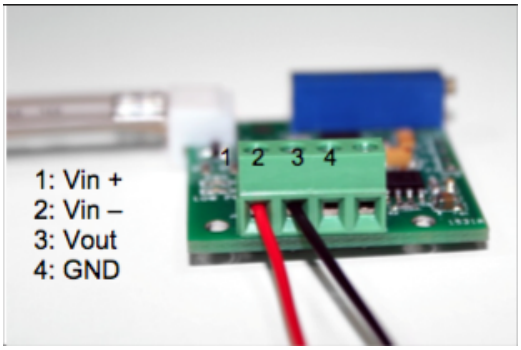

Figure 4.3: Board connection.

the output saturates decreases. The opposite is also true, as the feedback resistor decreases the maximum measurable force that increases.

4.3 Minimization of the computational time

In order to obtain good performances performing a grasping procedure, it is necessary to minimize the computational time. The main Matlab code, which allows to control the robotic hand, is composed of:

- reading from the sensor;
- sending commands to the hand;
- making graphics;
- control part.

At the beginning the attention is focused on the first two parts. Both are composted of sending a request and receiving an answer. In order to make this parts as fast as possible, the Arduino baud rate is set to the faster one and the Matlab library which manages the hand communication will be improved in fastness.
Concerning the *Arduino UNO* model, it is possible to set the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, the available rate values are: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. Thus, the last value is the chosen one. The syntax used is:

```
Serial.begin(speed),
```

where *speed* is the chosen baud rate.

When the PC interacts with the robotic hand, a command is sent on a certain register using the fwrite() Matlab function and the consecutive fread() operation is performed to acquire data or, in case of void function, to leave the buffer empty. This latter operation is fundamental to read the correct data from the buffer at the next operation, but at the same time it represents a waste of time. Indeed, most of the time, it does not provide any information. In order to make the code faster it is possible to change the functions such that the fread() operation is avoided. The buffer will be made empty only if it is necessary to get an information from it. The Matlab function used to make the buffer empty is:

```
flushinput(serial_port).
```

4.4 Calibration procedure

The calibration procedure is a method which allows to relate the sensor's output to an actual unit. In the case of an Flexiforce sensor, the electrical output will be transformed in Newtons. In order to calibrate the sensor, a set of five known weights are applied to the sensor and the corresponding output is acquired. The weights are scaled from 0 to 5.3 Newtons. Indeed, since the maximum force the sensor can measure is 5 N (1lb), this value is chosen as upper bound for the calibration. Then, the data are used to fit a linear regression model in order to build the sensor characteristic (Figure 4.4). Once the parameters of the characteristic are known, the

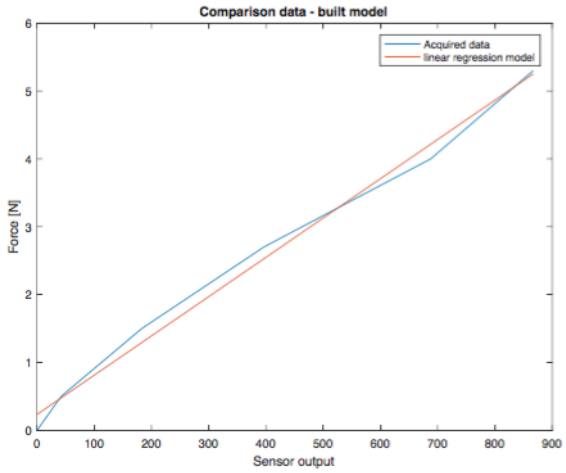

Figure 4.4: Comparison between the acquired data and the built model.

sensor's output can be transformed in a meaningful value expressed in Newtons.
The following Matlab function performs this transformation:

```
newtons = adc_to_newtons (adc_value, adc_offset ),
```

where *adc_value* is the sensor's measurement (between 0 and 1023) coming from the analog to digital converter (adc) and *adc_offset* is the offset measured with no load on the sensor. The default offset value is 10, but it may vary depending on sensor orientation with respect to gravity vector. The function returns the force in Newtons.

Unfortunately, after the calibration the the measurements remain quite noisy (Figure 4.5). For this reason a filter is implemented.

4.5 Filtering

The sensor measurements are filtered in order to reduce the noise. For this purpose, an "Exponential smoothing filter" is chosen. It is one of the most used filter for the analysis of time-series data which acts as low-pass filters to remove high frequency noise.
Given a windowed function composed of N discrete observation, the filter is based on a weighted sum of the current observation and the preceding observation. The raw data sequence is often represented by x_t beginning at time t=0, and the output of the exponential smoothing algorithm is commonly written as s_t, which may be regarded as a best estimate of what the next value of x will be. When the sequence of observations begins at time t=0, the simplest form of

exponential smoothing is given by the formulas:

$$s_0 = x_0$$

$$s_t = \alpha x_t + (1 - \alpha) s_{t-1}, \quad t > 0$$

where α is the smoothing factor, and $0 < \alpha < 1$.

After the filtering procedure, the acquired data from the sensor appears less noisy as can be

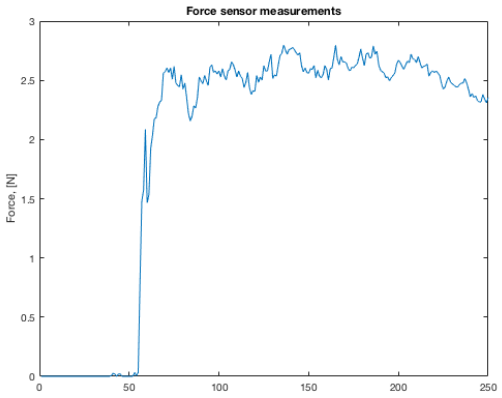

Figure 4.5: Force measurements after calibration.

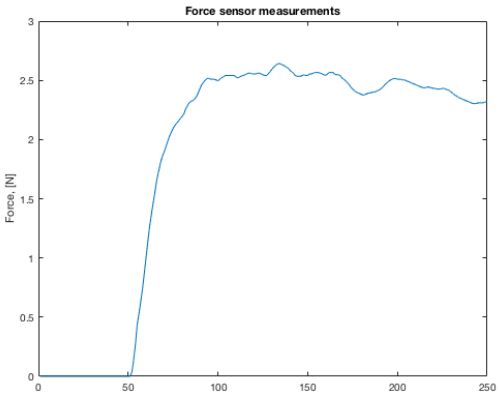

Figure 4.6: Force measurements after filtering procedure.

seen from the (Figure 4.6). The experimental set up used to analyse the force measurements is the same and it is shown in (Figure 4.7).

Figure 4.7: Experimental set-up during the force acquisition.