EDUCATIONAL SOCIETY OF SANTA CATARINA – SOCIESC
INSTITUTO SUPERIOR TUPY – IST

MATHEUS NORBERTO HAGEMANN
RAFAEL SOARES DE SOUZA

PROGRAMMABLE STEPPER MOTOR WITH INTEGRATED DRIVER AND ENCOLDER

Joinville-SC
2010

MATHEUS NORBERTO HAGEMANN

RAFAEL SOARES DE SOUZA

PROGRAMMABLE STEPPER MOTOR WITH INTEGRATED DRIVER AND ENCOLDER

Final Paper submitted to Instituto Superior Tupy as a partial requirement to obtain the title of Control and Automation Engineer, under the guidance of the

PROF. ROBERTO DO AMARAL SALES

Joinville-SC

2010

MATHEUS NORBERTO HAGEMANN
RAFAEL SOARES DE SOUZA

PROGRAMMABLE STEPPER MOTOR WITH INTEGRATED DRIVER AND ENCOLDER

This work was judged and approved in its final form, being signed by the professors of the examining board.

Prof. Roberto do Amaral Sales

Prof. Albarello

Prof. Alexandre Werner Arins

Joinville, December 14, 2010.

We dedicate this work to God who
sustained to achieve this
goal, and to our
family, and friends.

THANKS

We thank God for having given us the necessary strength to achieve this goal. To our families, for the affection and understanding for the moments when we were not present and to our friends who helped and encouraged us on this journey

SUMMARY

The purpose of this project is to develop a programmable stepper motor via serial port (RS232 "DB9") of the computer, which will include a driver along with an encoder that will interact in order to command the displacement of the motor ensuring its positioning. It is intended to elaborate it in such a way that its dimensional has little volume, since its components will be coupled to the same set. The possibility of cost reduction in the implementation of this component in industrial automation in general is studied, and it is expected that it will be easier to install and implement. The electronic circuit will be built focusing on using a microcontroller that will have the logical function of the implement.

Keywords: Stepper Motor. Programmable. Driver. Encoder. Microcontroller

ABSTRACT

The purpose of this project is to develop a programmable stepper motor via serial port (RS232 "DB9") computer, which include a driver with an encoder that will interact to control the displacement of the engine ensuring your position. The aim is to elaborate it so that its volume has little dimensional components since MT will be engaging them in the same set. It studies the possibility of reducing cost in implementing this component in industrial automation in general and expected a greater ease in installation and implementation. The electronic circuit will be built focusing using a microcontroller that has the logical function of the implement.

Keywords: Stepper motor. Driver. Encoder. Microcontroller.

LIST OF ILLUSTRATIONS

SUMMARY

1 INTRODUCTION

The stepper motor is found in equipment that needs precise movements. Its use in industrial automation is quite wide, and it is possible to control rotation angle, speed, position and synchronism.

The speed control will vary with frequency as the pulses are applied at the entrance. The angle of rotation is related to the number of pulses that is applied to its coils, and as for its direction, it is directly linked to the sequence of application of the pulses. It is observed that the stepper motor for industry covers a large area, especially in simpler applications, however for these applications it is known that there is a need for an electrical panel to incorporate the process control.

Thus, a product is suggested that can be either programmed or controlled by a computer, and that in a simple way allows the elaboration of a precise solution in control of rotation angle, speed, position and synchronism. Focusing on low cost, encolder, easy implementation and programming, the goal is to develop a driver that interacting with an encolder will be coupled to a stepper motor.

1.1 THEME AND THEME DELIMITATION

As the theme of this work suggests: **"Programmable stepper motor with integrated driver and enceder",** this work will be focused on the study for the development of a solution for the industry, in stepper motors.

1.2 PROBLEMATIZATION

The stepper motor currently used in the industry needs a driver that occupies a good physical space in an electrical control panel, it does not get a quick installation, and often also requires an external "PLC" controller for its control, which in turn ends up requiring specialized labor, so a simple application ends up adding up to an unnecessary cost.

A scenario was created to obtain a better comparison between what is currently offered by industrial automation" VS., and the product suggested in this project as described below: An equipment to verify the length "X" of iron bars through transverse displacement, it must show "Good Parts" and "Tailings Parts". The mechanical part consists of a stop on a spindle that will be rotated through a stepper

motor, which in turn through rotation control logic will monitor the displacement in the spindle axis). Quantified the automation for this equipment, the result will be a complete electrical panel, not counting labor, which would add up: PLC + DRIVER + STEPPER MOTOR + SOURCE + UTILITIES: 3,900.00 R$. In this application described, the implementation of an encolder that would be necessary to ensure positioning for the equipment is not foreseen.

In the proposed project, there will be an encolder for monitoring the positioning and this can be programmed or controlled by a computer with the simple requirement of having a serial port. Quantifying its cost for the same application: STEPPER MOTOR + DRIVER + ENCOLDER = 400 R$, and the driver will be built accommodating a PIC 16F887A microcontroller, which will have the logical function of the implement.

1.3 GENERAL OBJECTIVE

The main idea for this study is to offer a low-cost product that is a solution for industrial automation and that obtains the same or higher quality than the products of the same line found in the current market.

A control with programmable driver and encolder will be developed in a stepper motor in order to couple them, seeking to obtain the smallest possible volume for this set, in order to facilitate the process of installing this product in industrial machines. A programming menu will be developed, in the simplest and most objective way possible to facilitate its implementation.

There will be a mechanical device to simulate the control operation of the stepper motor, monitored through the programming software. The description of how to obtain this device follows the scenario already described above: (An equipment to check the length "X" of iron bars through transverse displacement, it must show "Good Parts" and "Reject Parts", the mechanical part consists of a stop on a spindle that will be rotated through a stepper motor, which in turn through logic will monitor the

displacement in the spindle axis).

1.4 SPECIFIC OBJECTIVE

A) Develop a board with a microcontroller "PIC16F887", with the function of controlling the stepper motor, which is the driver that will cover the pre-programmed routine for the stepper motor.

B) Develop together with the driver board, an RS-232 communication port to program and execute the program in the PIC.

C) Attach an encoder to the stepper motor shaft to ensure its rotation. Thus, a closed loop circuit will be formed, which when the driver logic is implemented will inform if there is slippage in the stepper motor.

D) Develop and build a nylon coupling that will integrate all the components of the assembly, ie; stepper motor + driver board + encolder.

E) Study the possibility of building an encolder with the objective of maximizing the volume of the set, and the smaller the volume obtained, the better its applicability.

F) Build a mechanical device as described in the general objective, it will be built based on 1020 carbon steel, spindle, and linar guide with ball skates.

1.5 HYPOTHESES

At first, the biggest difficulty is in the implementation of the encoder, because the stepper motor vibrates in a way that can interfere with the reading of the encoder, so if the stepper motor interferes with the reading of the encoder, the use of a closed loop circuit will be inappropriate, that is, the use of the encolder will be discarded. It is also not guaranteed that the plates to be built will be in the desired size for integration into the stepper motor, so the assembly may obtain a disproportionate size (expected size: width, base and height with at most the same dimensions as the stepper motor to be used). However, because this project is a prototype, its dimensional is not considered primordial.

1.6 JUSTIFICATION

The initiative for the study and implementation of this project is designed to innovate in stepper motors, applying low-cost solutions with ease of adaptation in machines for the industry, eliminating the control panel of the same.

2 THEORETICAL FOUNDATION

From this chapter the main theories of foundation to develop the work will be explained. They focus on the concepts and applicability of the components used in order to facilitate the understanding of the project.

2.1 STEPPER MOTOR

According to Messias (2007), stepper motors are electro-magnetic mechanical devices that can be controlled digitally through specific hardware and/or through software. It is a transducer that converts electrical energy into controlled movement through pulses, which enables displacement by step, where the step is the smallest angular displacement.

2.1.1 Types of Stepper Motors

According to Souza (2007) regarding the internal structure, there are basically 3 types of stepper motors:

- Variable reluctance;
- Permanent magnet;
- Hybrid.

Variable reluctance – consists of a multipolar mild steel rotor with windings on the stator. When the stator coils are energized, the rotor teeth attempt to align with the stator poles. By switching the coils that are fed into the stator, the stator field changes and thus the rotor moves to a new position. Figure 1 shows the internal structure of a variable reluctance stepper motor. (SOUZA, 2007)

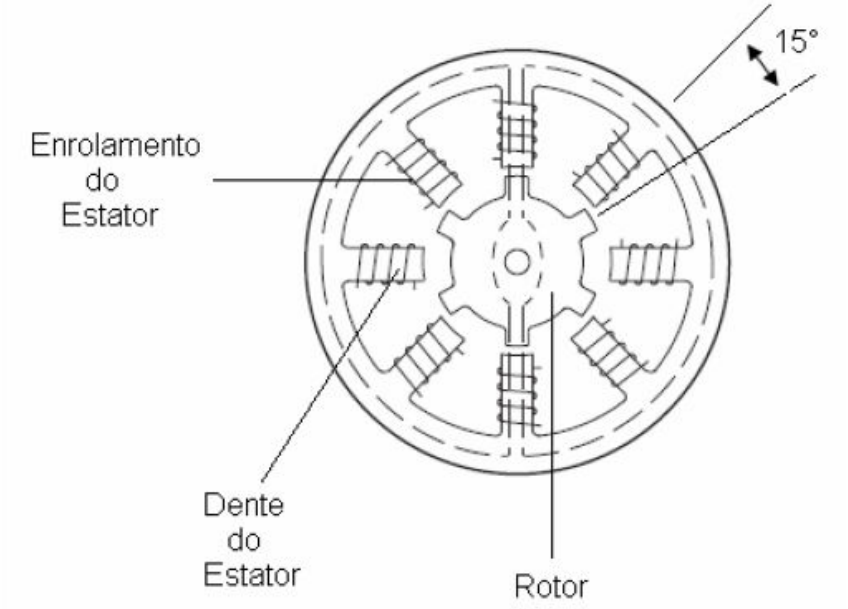

Figure 1 – Internal Structure of a Variable Reluctance Stepper Motor

Source: Technical English – Spanish Vocabulary, (2010)

Permanent magnet – It has a rotor made up of a permanently magnetized material Its operation is based on the reaction between the fixed magnetic field of the rotor and the field generated in the stator winding. (SOUZA, 2007)

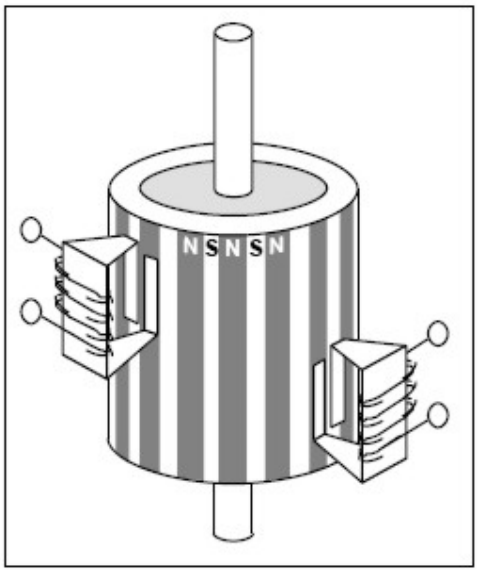

Figure 2 - Permanent Magnet Stepper Motor Rotor

Source: Technical English – Spanish Vocabulary, (2010)

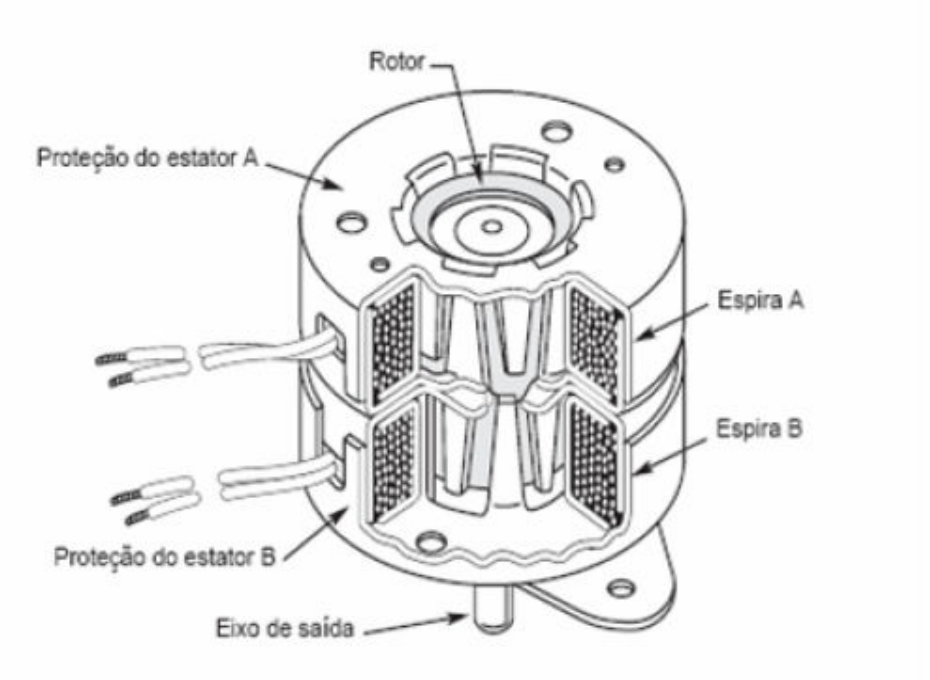

Figure 3 - Cutting view of a permanent magnet stepper motor

Source: LYRA (2010, p.3)

Hybrids - It is the most commonly used type in industrial applications. It combines the operating principles of the other two types of engines presented earlier. Being a small motor with high torque and small steps. (SOUZA, 2007)

> The core structure of the stator of a hybrid motor is essentially the same as that of a variable reluctance motor, the difference being that in the variable reluctance motor only one of the coils of a phase is wound on one pole, whereas a typical hybrid motor has the coils of two different phases on the same pole. These two coils on the same pole are wound in a configuration known as a bifilar connection. (KENJO, 1986)

The motor frame consists of two pieces of multi-toothed pole. Between these pole pieces is a permanent magnet magnetized in parallel with the rotor axis, making one end a north pole and the other a south pole. Figure 4 shows the cross-section of a hybrid engine. (SOUZA, 2007)

> The hybrid motor combines the best features of permanent magnet motors and variable reluctance motor. The rotor is multi-toothed as in the variable reluctance motor and contains a permanent magnet around its shaft. The rotor teeth provide a better path that helps guide the magnetic flux to preferred locations in the air gap. (SOMESSARI, 2010, p.20).

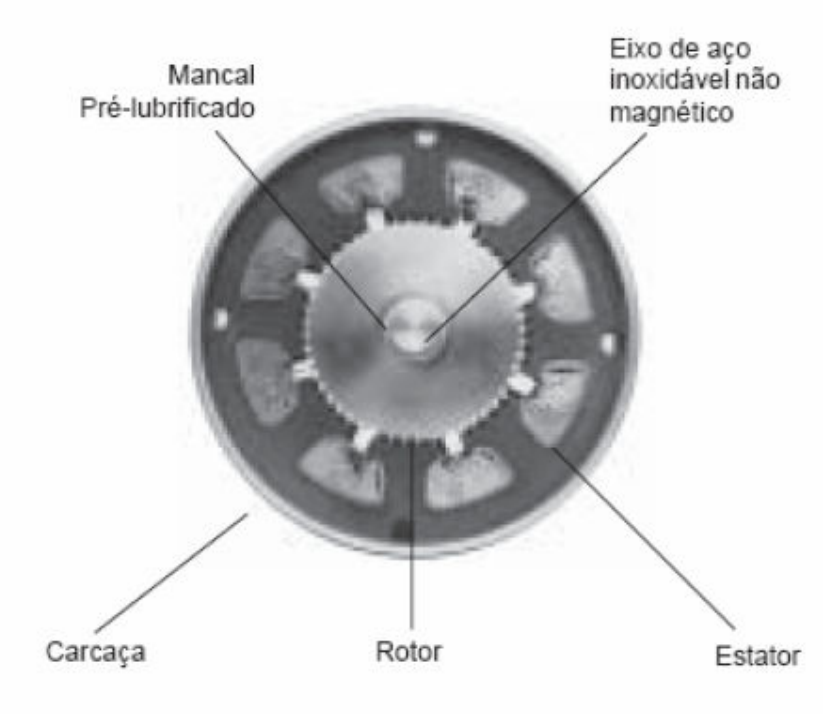

Figure 4 - Cross-section of a hybrid engine
Source: (SOUZA, 2007)

2.1.2 Types of power supply of windings of a stepper motor

Unipolar power supply – According to Marco Antonio de Souza (2007) It has a central derivation in each of the windings, where the number of phases is twice the number of coils. Figure 5 below represents a 4-phase unipolar stepper motor. (SOUZA, 2007)

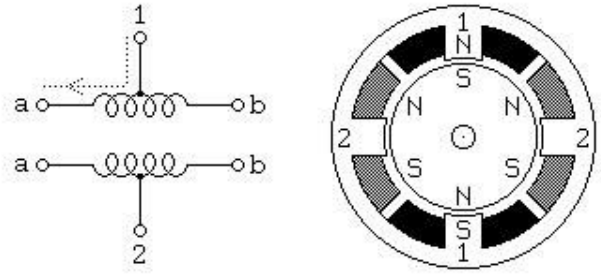

Figure 5 - Unipolar stepper motor

Source: SOUZA (2007)

Bipolar power – It has a single winding per phase, which must be fed in both directions to allow the advancement of one step. To do this, the polarity of the voltage in the coil must be reversed and sequentially. Figure 6 below represents a bipolar stepper motor.

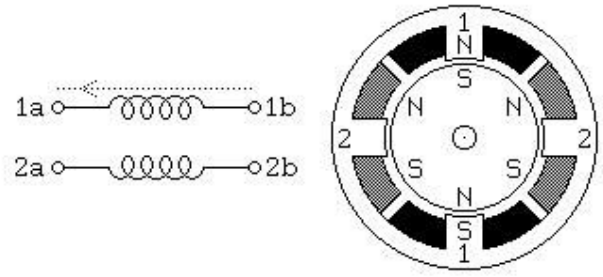

Figure 6 - Bipolar stepper motor

Source: SOUZA (2007)

2.1.3 Stepper Motor Operating Modes

> Energizing one and only one coil at a time produces a small displacement in the rotor. This displacement occurs simply because the rotor is magnetically active and the energization of the coils creates an intense magnetic field that acts to align with the rotor teeth. Thus, by properly polarizing the coils, we can move the rotor between the coils (half-step) or aligned with them (full-step). (BRITES & SANTOS, 2008, p.5)

There are three modes of operation for a stepper motor:

Full-step 1:

According to Brites & Santos (2008), only one coil is energized at each step, where energy consumption and torque are low with a higher speed. Figures 7 and 8

show, respectively, the energization of a coil producing a small displacement in the rotor of single-pole and bipolar full-step 1 motors .

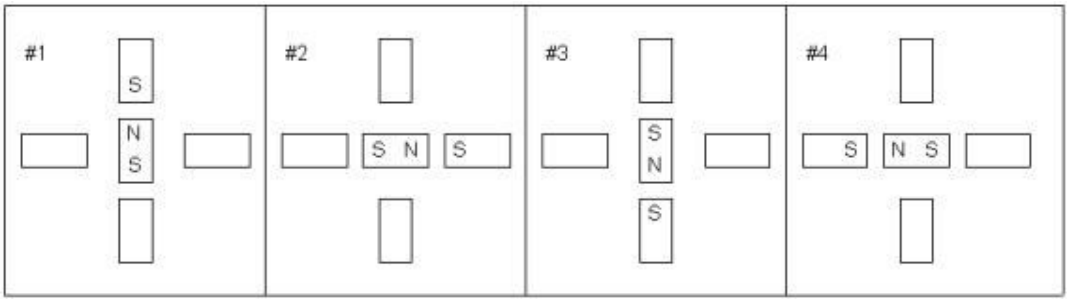

Figure 7 - Integer Stepper Unipole Motor

Source: Brites & Santos (2008)

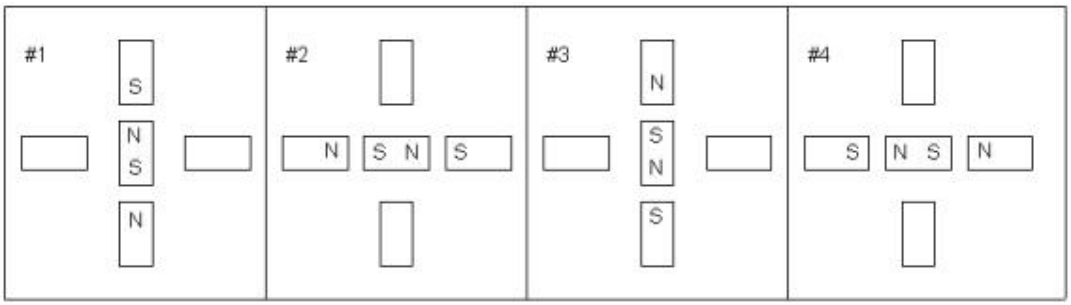

Figure 8 - Bipolar integer motor

Source: Brites & Santos (2008)

Full-step 2

Two coils are energized at each step where the torque and power consumption are higher than the full step 1 but speed is the same. Figures 9 and 10 show respectively the energization of two coils producing a small displacement in the rotor of single-pole and bipolar full-step 2 motors . (MESSIAS, 2006)

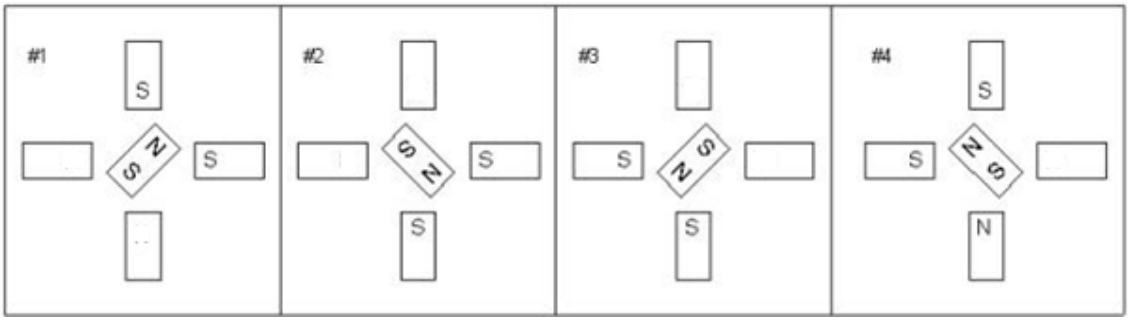

Figure 9 - Integer unipolar motor

Source: www2.eletronica.org/artigos/outros/estudo-do-motor-de-passo-e-seu-controle-digital

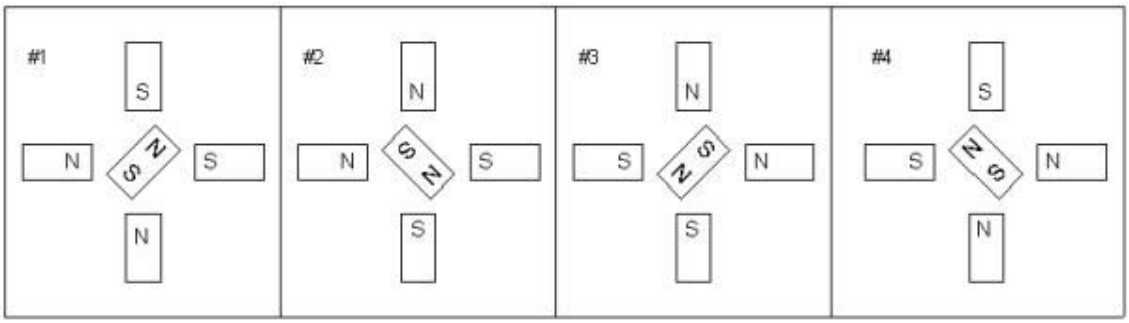

Figure 10 - Bipolar integer motor

Source: www2.eletronica.org/artigos/outros/estudo-do-motor-de-passo-e-seu-controle-digital

Half-step

The combination of full step 1 and full step 2 that generates a half-step effect of revolution. This mode consumes more energy and is more accurate, but its speed is lower than that of Full Step 1 and 2, and the torque is close to that of Full Step 2.

Figures 11 and 12 show, respectively, the energization sequence of the coils producing a small displacement in the rotor of unipolar and half-step motors. (MESSIAS, 2006)

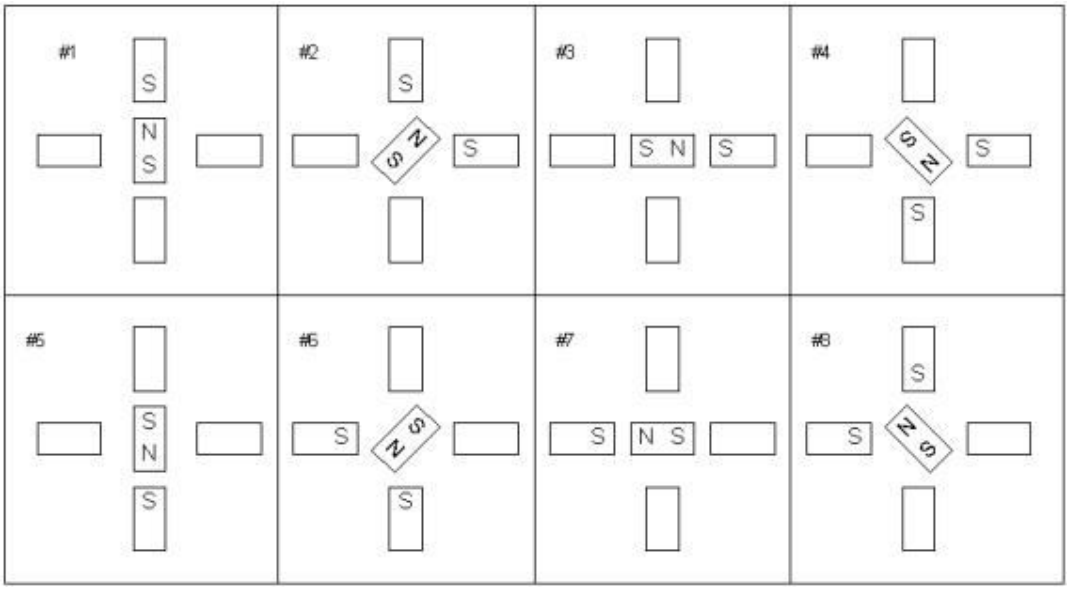

Figure 11 - Half-step unipolar motor

Source: www2.eletronica.org/artigos/outros/estudo-do-motor-de-passo-e-seu-controle-digital

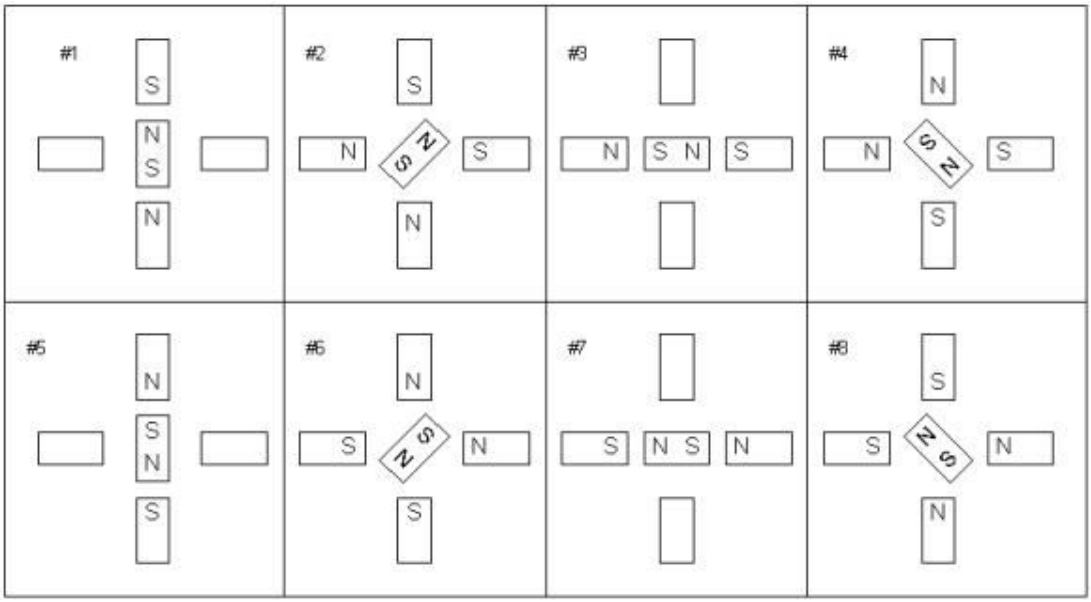

Figure 12 - Half-step Bipolar Motor

Source: www2.eletronica.org/artigos/outros/estudo-do-motor-de-passo-e-seu-controle-digital

2.1.4 Stepper Motor Control Modes

For the control of these stepper motors, the following figures with the drive sequences are as follows:

Número do Passo	Bobina 3	Bobina 2	Bobina 1	Bobina 0	Decimal
1	1	0	0	0	8
2	0	1	0	0	4
3	0	0	1	0	2
4	0	0	0	1	1

Figure 13 - Full-step 1 (Full-step) operating mode

Source: (MESSIAS, 2008)

Número do Passo	Bobina 3	Bobina 2	Bobina 1	Bobina 0	Decimal
1	1	1	0	0	12
2	0	1	1	0	6
3	0	0	1	1	3
4	1	0	0	1	9

Figure 14 - Full-step 2 (Full-step) operating mode

Source: (MESSIAS, 2008)

Número do Passo	Bobina 3	Bobina 2	Bobina 1	Bobina 0	Decimal
1	1	0	0	0	8
2	1	1	0	0	12
3	0	1	0	0	4
4	0	1	1	0	6
5	0	0	1	0	2
6	0	0	1	1	3
7	0	0	0	1	1
8	1	0	0	1	9

Figure 15 - Half-step mode of operation

Source: Source: (MESSIAS, 2008)

2.1.5 Speed, Positioning, and Direction

The speed of the stepper motor, according to Messias (2008), is controlled when a sequence of digital pulses is sent at a certain interval. This interval should not be less than 10ms between each step, as the motor will lose torque and will not rotate. The size of the rotational angle is directly related to the number of steps applied.

The direction of the stepper motor is controlled by the sequence of the feed of its coils, which in turn give movement to the motor. Figures 16 and 17 show the sequences that must be used to move the stepper motor clockwise and counterclockwise.

Número do Passo	Bobina 3	Bobina 2	Bobina 1	Bobina 0	Decimal
1	1	0	0	0	8
2	0	1	0	0	4
3	0	0	1	0	2
4	0	0	0	1	1

Figure 16 - Complete Step 1 (right)

Source: Source: (MESSIAS, 2008)

Número do Passo	Bobina 3	Bobina 2	Bobina 1	Bobina 0	Decimal
1	0	0	0	1	1
2	0	0	1	0	2
3	0	1	0	0	4
4	1	0	0	0	8

Figure 17 - Full Step 2 (left)

Source: Source: (MESSIAS, 2008)

2.2 MICROCONTROLLER (PIC 16F628A)

Pereira (2007) states that a microcontroller is an integrated circuit that makes use of all the components necessary for the control of a process. Microcontrollers have an Arithmetic Logic Unit (ULA) that processes all parts of logical and mathematical operations, all compressed into a single IC (Integrated Circuit).

Figure 18 exemplifies the internal block diagram of a microcontroller.

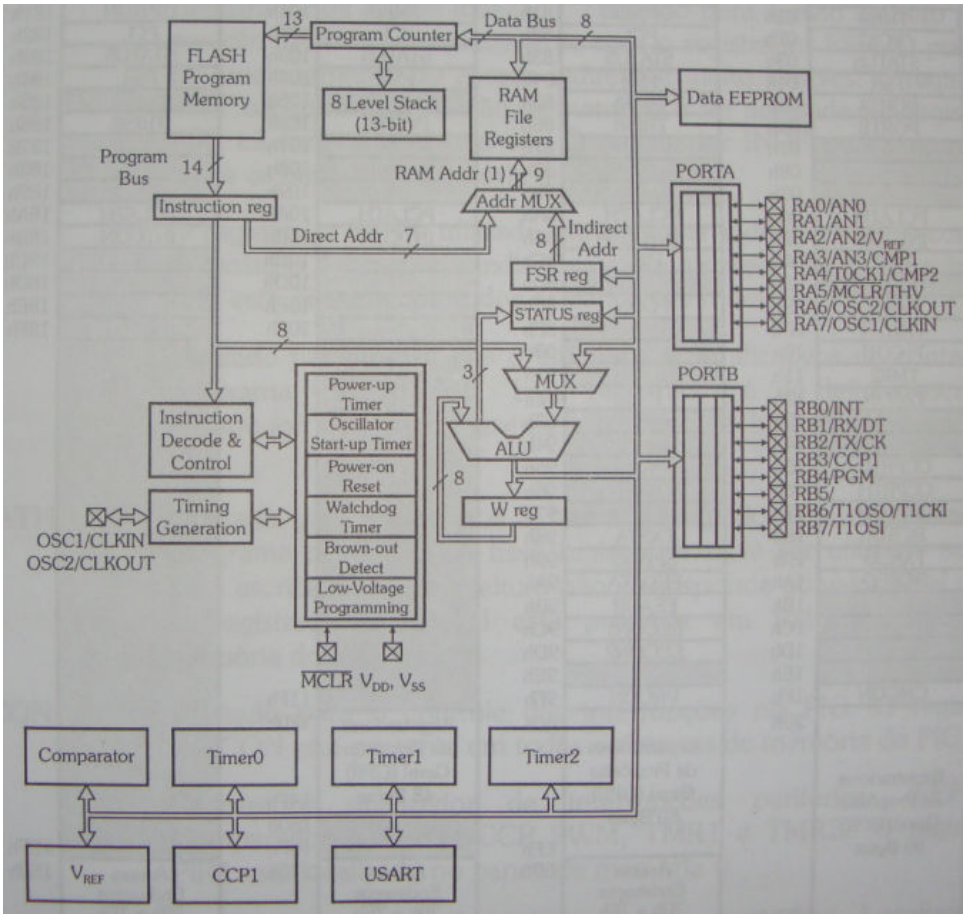

Figure 18 - Internal structure of PIC 16F628

Source: Pereira (2007, p.101)

The microcontroller chosen was the PIC 16F627 from the manufacturer Microchip, as it is an intermediate "Mid-rande" controller that technically meets the needs of this project.

According to Pereira (2007, p.35):

> PIC microcontrollers are a family of devices manufactured by MIcrochop. Using a RISC architecture, with clock frequencies of up to 40Mhz, up to 2048K Word of program memory and up to 3968 bytes of RAM. In addition, they can be found with several internal peripherals, such as: up to four timers/counters, internal EEPROM memory, PWM generator/comparator/sampler, A/D converters up to 12 bits, CAN bus interface, 12C, SPI, among others.

Its main characteristics are as follows, according to Pereira (2007, p.99):

- 1024x14bits FLASH memory (2048x14bits for the 16F628)
- 224x8bits of SRAM memory available to the user

- 128x8bits internal EEPROM memory
- 8-level stack
- 15 I/O pins (input or output)
- (1x) Input Pin
- (1x) 8-bit timer/counter
- (1x) 16-bit timer/counter
- (1x) 8-bit timer
- (1x) PWM Channel with Capture and Sampling (CCP)
- (1x) Serial USART communication channel
- (2x) Analog comparators with internal programmable voltage reference
- (1x) Timer watchdog
- (10x) Independent interrupt sources
- 25mA current capacitors per I/O pin
- 35 Instructions
- Operating frequency from DC (0 Hz) to 20 Mhz
- Internal 4Mhz/37Khz oscillator
- Operating voltage between 3.0 to 5.5V
- Pin-to-pin compatible with 16F84 and other 18-pin PICs

According to Pereira (2007), there are fundamentally three families of PICs differential by the size of the word of the program memory: 12, 14 and 16 Bits. All of these devices have an internal eight-bit data bus.

2.2.1 Physical Characteristics and Pinout of PIC 16F628A

The pinout structure of the PIC 16F628A microcontroller is shown in figure 19, extracted from the book by David J. de Souza (2007).

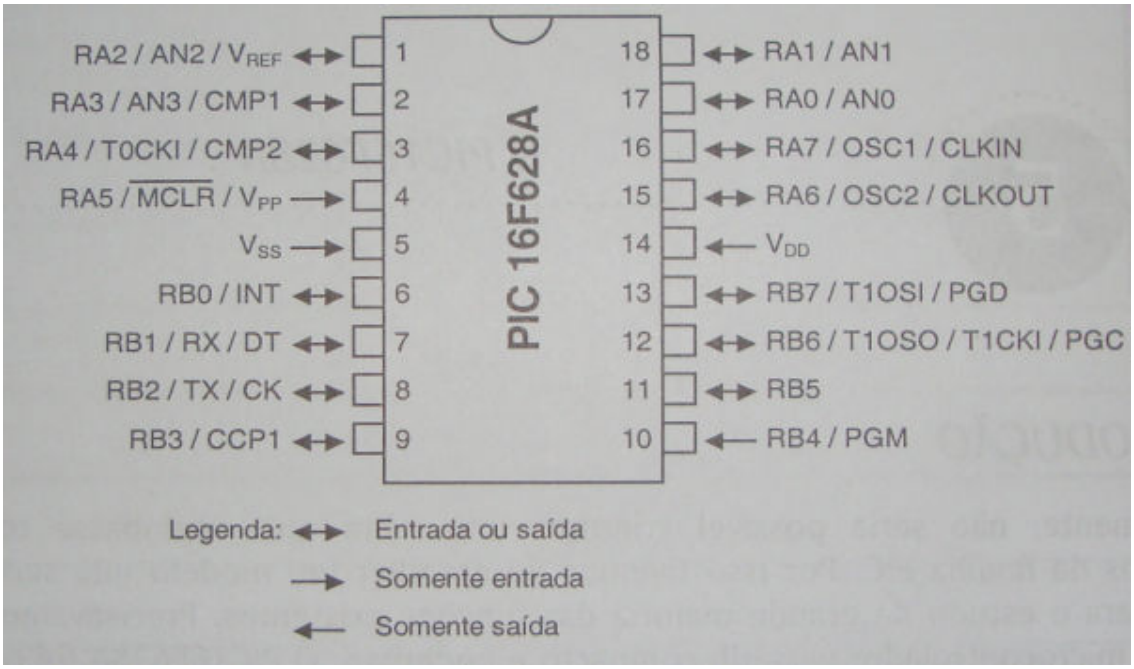

Figure 19 - PIC 16F628A Pinout

Source: Souza (2007, p.36)

The following chart (figure 20) describes the details of the naming of each pin:

Pin Number	Function	Input Type	Output Type	Description
17	RA0	ST	CMOS	Biodirectional digital I/O
	AN0	ANP	-	Analog input for comparators
18	RA1	ST	CMOS	Biodirectional digital I/O
	AN1	ANP	-	Analog input for comparators
1	RA2	ST	CMOS	Biodirectional digital I/O
	AN2	ANP	-	Analog input for comparators
	Vref	-	ANP	Programmable reference voltage output
2	RA3	ST	CMOS	Biodirectional digital I/O
	AN3	ANP	-	Analog input for comparators
	CMP1	-	CMOS	Output of comparator 1
3	RA4	ST	OD	Biodirectional digital I/O
	T0CKI	ST	-	TMR0 Counter External Input
	CMP2	-	OD	Output of comparator 2
4	RA5	ST	-	Digital input
	MCLR	ST	-	External Master Cler (reset). The PIC only works when this pin is at a high level
	Vpp		-	Input for programming voltage (13V)
15	RA6	ST	CMOS	Biodirectional digital I/O

	OSC2	-	XTAL	External crystal output
	CLKOUT	-	CMOS	Square wave output at 1/4 of the frequency imposed on OSC1 when in RC mode. This frequency is equivalent to the internal machine cycles
16	RA7	ST	CMOS	Bidirectional digital I/O
	OSC1	XTAL	-	External crystal input
	CLKIN	ST	-	Input for external oscillators (hybrid or RC)
6	RB0	TTL	CMOS	Bidirectional digital I/O with internal pull-up
	INT	ST	-	Input for external interruption
7	RB1	TTL	CMOS	Bidirectional digital I/O with internal pull-up
	RX	ST	-	Reception for Asynchronous USART Communication
	DT	ST	CMOS	Data Path for Asynchronous USART Communication
8	RB2	TTL	CMOS	Bidirectional digital I/O with internal pull-up
	TX	-	CMOS	Transmission for asynchronous USART communication
	CK	ST	CMOS	Clock path for synchronous USART communication
9	RB3	TTL	CMOS	Bidirectional digital I/O with internal pull-up
	CCP1	ST	CMOS	I/O for Capture, Compare, and PWM
10	RB4	TTL	CMOS	Bidirectional directional I/O with internal pull-up. Interruption due to change of state.
	PGM	ST	-	Input for low voltage programming (5V)
11	RB5	TTL	CMOS	Bidirectional digital I/O with internal pull-up. Interruption due to change of state.
12	RB6	TTL	CMOS	Bidirectional digital I/O with internal pull-up. Interruption due to change of state.
	T1 OSO	-	XTAL	External crystal output for TMR1
	T1 CKI	ST	-	External counter input TMR1
	PGC	ST	-	Serial Programming Clock (ICSP)
13	RB7	TTL	CMOS	Bidirectional digital I/O with internal pull-up. Interruption due to change of state.
	T1 OSI	XTAL	-	External crystal input for TMR1
	PGD	ST	CMOS	Serial Programming Date (ICSP)
5	Vss	P	-	GND
14	True	P	-	Positive eating

Figure 20 - PIC 16F628A Pinout Nomenclature

Source: Souza (2007 p.37-38)

Caption:

P = *Power* (food)
- = Unused
TTL = TTL type input
ST = Input type *Schmitt Trigger*
CMOS = CMOS type output
OD = Open Drain
IN = Analog input / output (SOUZA, 2007, p.37-38)

2.3 MICROCONTROLLER PROGRAMMING

The microcontrolled PIC 16F628A will be programmed by the MPLAB software of the company MICROCHIP "figure 21", in the C++ programming language.

Figure 21 - MICROCHIP Company Logo
Source: Microchip Technology (2010, p.1)

2.3.1 MPLAB

MPLAB is a software that establishes the programming logic of the microcontroller.

2.4 C++ LANGUAGE

According to Wikipedia (2010), C++ is a multiparadigm and general-purpose programming language. The language can be considered medium-level, since it combines the characteristics of high- and low-level languages .

> In C++, the standard library is a collection of classes, functions, and variables written in the language itself to facilitate application development. It also incorporates the C standard library, and all of its functionality is declared in the namespace" "The C++ standard library provides several generic containers , functions that use and manipulate such containers, object functions, strings, and *streams* support for some language facilities and general-purpose functions, such as mathematical functions. (WIKIPEDIA, 2010, p.1).

C++ language offers greater ease of programming by having more specific and defined instructions. It has a wide library for a better application in the most particular routines. (WIKIPEDIA, 2010).

The following is the standard C++ library, according to Wikipedia (2010, p.2):

List of Headers

Containers
<bitset> - bit array manipulation, something like vector<bool> (which is a non-recommended construction[1])
<deck> - double-linked list handling
<list> - simply linked list manipulation
<map> - Ordered Associative Set Manipulation (Join: Key → Value)
<queue> - FIFO list manipulation
<set> - set handling
<stack> - LIFO list manipulation
<vector> - arrangement manipulation
General Purpose
<algorithm> - generic algorithms
<functional> - object functions
<iterator> - iterator declaration
<locale> - manipulation of various cultural conventions of the user, such as the representation of numbers, currency and dates, for the purpose of internationalization
<memory> - functions for memory management
<stdexcept> - <exception> specialization, provides reporting on exceptions
<utility> - operations with pairs of elements (comparison and construction)
Strings
<string> - string manipulation
Streams *and input/output*
<fstream> - Manipulating file-based data stream
<iOS> - More general dataflow statement
<iostream> - standard system data stream handling (standard input, standard output, and standard error output)
<IOSFWD> - Declaration of data flows present in the language
<Yomanip> - handling the presentation and processing of data streams
<istream> - Data Input Manipulation
<ostream> - handling data output
<sstream> - handling data stream on strings
<streambuf> - handling data stream buffers
[edit] Numerical functionalities
<complex> - complex number manipulation
<numeric> - operations with numerical sets
<valarray> - arrangement of mutable values
C++ Language Support
<exception> - exception handling
<limits> - manipulation of numerical limits of the types embedded in the language

<new> - handling memory allocation and deallocation
<typeinfo> - C ++ RTTI Aid
C Standard Library
<cassert> - Suitability of <assert.h>
<cctype> - Fit < ctype.h>
<cerrno> - Adequacy of <errno.h>
<cfloat> - suitability of <float.h>
<climits> - < limits.h suitability>
<cmath> - <math.h fitting>
<csetjmp> - Fit <setjmp.h>
<cSignal> - Suitability of <signal.h>
<cstdlib> - Adequacy of <stdlib.h>
<cstddef> - Adequacy of <stddef.h>
<cstdarg> - Appropriateness of <stdarg.h>
<ctime> - < time.h fit>
<cstdio> - suitability of <stdio.h>
<cstring> - < string.h fit>
<cwchar> - Suitability of <wchar.h>
<cwctype> - < wctype.h fit>

Detailed Descriptions

This header, according to Wikipedia (2010), provides several useful generic algorithms for searching, sorting, and transforming containers (data structures), among others. They can be invoked for different containers through the common interface of the iterators, and the specific operators that each algorithm requires from the data structure used. Algorithms are commonly specified through the start and end position of the data structure, and the iterator at the end of the structure must be accessible from the iterator at the beginning of the same structure through continuous increments in the iterator. Some algorithms promote a special entry condition; for example, binary search algorithms, which require a previously ordered data structure. Note, however, that this requirement is implicit and not detectable at compile time, and the developer is responsible for meeting the requirements. (WIKIPEDIA, 2010).

The algorithms can be classified from this library into two large groups, those that mutate or mutate in containers and those that do not. Element search, comparison, and counting algorithms do not mutate, they only read the container and rewind references to elements in the structure. On the other hand, algorithms for sorting, copying, transforming, and adding or removing elements mutate. (WIKIPEDIA, 2010).

For algorithms that perform mutation, they have the special suffix copy, which recommends that the algorithm keeps intact the container passed as a parameter, returning in the output a new container that corresponds to the original container plus the processing performed. For all algorithms, modifiable or not, there is the special suffix _if, used in algorithms that involve comparing elements. The indication is that a comparison function is being passed per parameter to the algorithm, favoring the use of the standard comparison operators that are provided by the container's data type. (WIKIPEDIA, 2010).

With the information resources available on Wikipedia (2010), they were used for the following description:

1) <fstream>

std::fstream is a transmitter of computer file data streams designated for the native CHAR data type. It allows reading and writing in text mode (bit, << and >> offset operators must be used) or binary (read and write methods are used for data buffers). The standard library also provides classes for read-only (std::ifstream) or write-only (std::ofstream) use cases. (WIKIPEDIA, 2010).

The implementation of std::fstream follows the RAII pattern. Management of the open file (a system resource) is the responsibility of the class. This implies that by initializing std::fstream (through information such as the file name and opening mode) the resource is acquired, and on the destruction of std::fstream the resource is released to the system automatically. Although the standard library provides the close method for manually flushing the file to the system, RAII allows this to be done automatically when the std::fstream instance goes out of scope in the program.(WIKIPEDIA, 2010).

2) <functional>

This header provides support for object functions, classes that encapsulate functions so that the class instance can be invoked just like any other function. For example, object functions are used in the STL for passing predicates to generic <algorithm> algorithms. Unary (which require one argument) and binary (which require two arguments) object functions, adapters that allow you to convert function pointers into object functions, and adapters that allow you to convert binary object functions into unary ones by associating a value to one of the arguments. (WIKIPEDIA, 2010).

The header still defines some general-purpose object functions such as arithmetic and logical operations. An example is equal_to, a binary object-function that tests whether two values are equal. It is nothing more than a generic comparison function encapsulated in a class. (WIKIPEDIA, 2010).

3) <iostream>

This header is responsible for manipulating the system's standard data flow (standard input, standard output and standard error output) and is representative of the evolution simulation of the <stdio.h> header of the C language. output, *unbuffered* error and buffered error at the same time; for this we use the bit-shift operators (<< and >>).(WIKIPEDIA, 2010).

Methods for formatting the data stream are also provided, such as width, which defines a width for the output, fill, which defines a specific character to be printed if the stream is less than the expected minimum, and precision, which defines the number of significant digits of floating-point numbers. (WIKIPEDIA, 2010).

Some compilers are unable to remove unnecessary code when producing executables that include this library via static binding. For example, a Hello World program using the GNU implementation of the standard library produces an executable larger than the equivalent using <cstdio> due in part to linker deficiencies. (WIKIPEDIA, 2010).

4) <locale>

This header manipulates various user cultural conventions, such as the representation of numbers, currency, and dates, for the purpose of internationalization. The library makes use of the facet, an interface to a service in a specific location. Each locale has a set of facets. The default constructor of the std::locale class defines a copy of the locale of the machine running the program, with the user's current conventions. (WIKIPEDIA, 2010).

5) <map>

The std::map<Key, Data, Compare, Alloc> container is an ordered associative set that maps an object of type Key (the key) to objects of type Data (the value). Keys are unique: if an object is inserted with an already existing key, the present value is replaced by the value entered. (WIKIPEDIA, 2010).

The time required for random access to each element is $O(\log(n))$, and the assigned iterators are not invalidated after insertion and removal operations. Therefore, the most commonly used implementation for the container is the self-

balancing binary search tree (although any other data structure that respects the constraints of computational complexity can be used, such as a *skiplist*).

Internally, map elements are sorted using curly brackets.

A variation of the container is std::multimap, which allows for repeated keys. (WIKIPEDIA, 2010).

6) <Sep>

The std::set<Key, Compare, Alloc> container is an associative set that allows fast random access to data. It differs from the std::map container in that the values of the elements are also their keys. For this reason, each value (and therefore its key) is unique, it cannot repeat. The container can be accessed bidirectionally, from the beginning or the end. (WIKIPEDIA, 2010).

The internal implementation of the container is usually a binary search tree. A variation of the container is std::multiset, an associative multiset, which allows for repeating values. (WIKIPEDIA, 2010).

7) <sstream>

std::stringstream is a specialized string data stream handler for the native char data type. It allows reading and writing in text mode (using bit, <<, and >> offset operators) or binary (using the read and write methods for data buffers).

The standard library also provides classes for read-only (std::istringstream) or write-only (std::ostringstream) use cases. (WIKIPEDIA, 2010).

8) <string>

The std::string container is a specialized string for the native char data type. It removes many of the problems introduced by the C language by relying on the programmer to manage strings, internally encapsulating routines and considerations that the programmer does not need to be aware of. It also allows conversion to and from C text strings (const char*). (WIKIPEDIA, 2010).

Unlike a C string, which comprises a pointer to a memory region containing the string, the contents of std::string are stored by value. For this reason, the copy operation has O(*n*) time. To avoid unnecessary copies, passing text strings as a function parameter is usually done by constant reference. This has a second advantage because it allows the same function to implicitly receive a string from C, without requiring overhead for the const char* data type.

Specialization for char chains is not always desired. For example, an application that implements UTF-32 (the mapping of the Unicode standard to 32 bits,

sufficient to represent the entire standard) should reserve four bytes for each character, while a char stores only one byte. Therefore, the generic text string concept is implemented in the std::basic_string container, which can be specialized for any type of data. In the case of UTF-32, you can use a generic container specialization for a data type that stores at least four bytes. Since most of the time a char is sufficient, specialization is also set in the standard for convenience. (WIKIPEDIA, 2010).

9) <vector>

The std::vector container is an arrangement and generalizes the concept of a vector in C. Access can be through indexes for the data as well as in C (through an overload of the appropriate operator) and its memory is placed contiguously. However, unlike a vector in C, the size of the container is dynamic that manages itself automatically and there is greater flexibility to add elements. In addition to knowing its current size, an instance of std::vector can also know how many elements it can still allocate before it needs resizing: the allocation is done in blocks and not for each element, and it is possible to manually define the size of the block. Unlike a vector in C, std::vector provides the at method to access elements from the index, but with exception handling for invalid indexes (values negative or above the current container size).

For insertions in the middle of the container std::list is more efficient. While std::vector provides efficient insertion at the end of the container, std::d e provides efficient insertion at both the beginning and end of the container. In addition to efficient insertion at the end, strengths of std::vector include constant-time access to any element through the index and linear-time iteration.[8]

Being a generic container, it can be specialized for different types of data. However, it is not advisable to specialize it for the native Boolean type , bool, whose specialization is already in the standard library. (WIKIPEDIA, 2010).

2.5 SERIAL COMMUNICATION

As Canzian (undated) expresses, the distance that a given signal travels in a computer varies from a few millimeters, as in the case of connections to a simple IC, to several centimeters when the signal connection includes, for example, a motherboard with connectors intended for several circuits. Digital data for these

distances can be transmitted directly. Except in very fast computers, designers do not care about the shape and thickness of the conductors, or the analog characteristics of the transmission signals.

2.5.1 RS 232 Communication Port

According to Braga (2009, p.1), the RS-232 protocol is expressed as follows:

> The RS-232 protocol was established in 1962 and was called EIA232. It was created to be used with the old teletype machines, but due to its simplicity and versatility it became a standard that is still adopted today in short-distance applications, especially those involving communication between the machines (sensors and effectors) and the computers that control them.

RS232 serial communication is a standardized protocol of easy access and low cost, it works with standardized transmission rates lower than 20kbps (4.8, 9.6 and 19.2 kbps) and with voltage levels referenced to the GROUND pin (Ground - pin 9 in DB-9), and when the voltage level is greater than +3V the logic level signal is zero (0), and when the voltage level is less than -3V then the logic level will be one (1). As much as it is a widely used communication, the standard (RS232) has serious limitations in relation to electromagnetic interference. (PEREIRA & LAGES, UNDATED)

2.5.2 Connections

In figure 22 according to the IDEV website, (undated) we have the pinout of a DB-9 connector, defined by the EIA-574 standard. This is found in the communication ports of the PC and is used in asynchronous communication of data (RS-232/V.24).

1 - Carrier Detects (CD)

2 - Receives data (RD)

3 - Transmits data (TD)

4 - Ready-to-ship (DTR) terminal

5 - Earth (SG)

6 - Ready to communicate (DSR)

7 - Ready to Ship (RTS)

8 - Ready to Stream (CTS)

9 - Tone Indicator (RI)

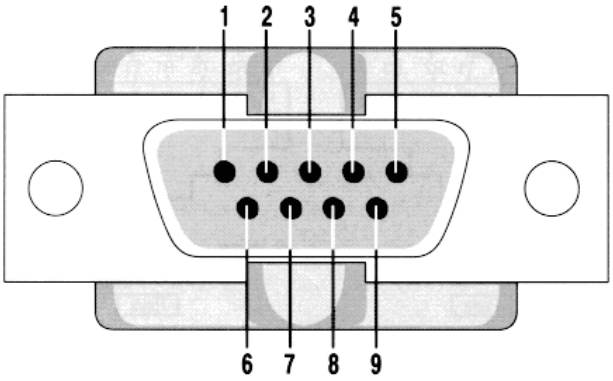

Figure 22 - DB-9 Connector Pinout

Source: (Website IDEV (UNDATED, P.1)

2.5.3 Serial transmission

The most common mode of signal transmission, as expressed (Souza (undated), is asynchronous. In this mode there is no need for the transmitter to be synchronized with the receiver, as it is informed when each "data packet" begins and ends, through bits called *start bit* and *stop bit.* Thus, the signal is formed by individual bits that are sent (one by one) in a "packet" of a defined size in ASCII format.

Each packet is made up of 10 to 11 bits, of which 8 bits make up the portion of the packet that refers to the portion of the message sent. These are sent after a signal from *start bit* is recognized. To do this, the line (Pin), which is normally at 'logical level 1', moves to 'logical level 0'. At the end of the data bit packet, a parity bit can be sent to verify that the information has been successfully received. Then a *stop bit,* indicating that the shipment of this package has been completed. If this was the last packet, the line level remains high (1), but if a new packet is to be sent, there is a new transition from level 1 to 0, which is recognized as *start bit*, and the process repeats itself. (SOUZA, S/D)

2.5.4 Driver (CI) for Transformation of TTL Levels to RS232

To communicate between a digital device and an RS232 interface, it is necessary to transform TTL levels (0 to 5 Volts) into RS232.

The MAX232 IC is equipped with a charge pump circuit, capable of generating voltages of +10 and -10 Volts from a +5 Volt source, with just a few external capacitors. This CI also has 2 receivers and 2 drivers in the same package. Figure 23 shows the pinout and configuration of this IC. (SOUZA, S/D)

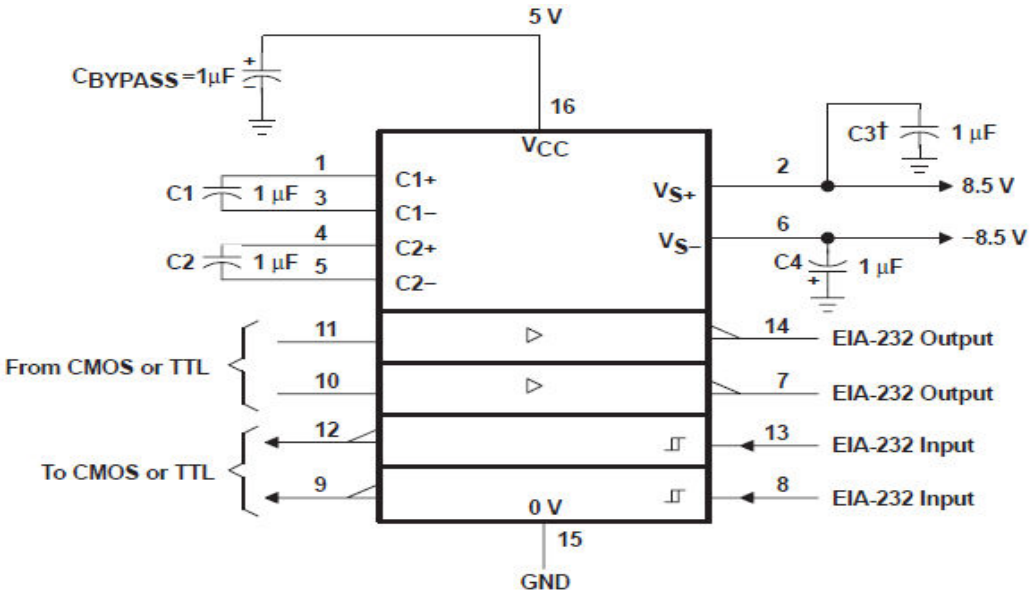

Figure 23 - Ci - Max 232 Pinout and Configuration

Source: datasheet Max 232 texas instruments

2.6 FEEDBACK DEVICES – ENCODERS

With the contribution of Matias (2008, p.1) it is clear that encolder's are transducers that convert an angular or linear movement into a series of electrical pulses. When coupled to the motors, they allow you to know their positions and speeds, enabling their control. In the case of using the encoder in stepper motors, it provides the verification of executed steps, thus ensuring the precision of their movements. The encoder reading system is done through an optical sensor "emitter" and a receiver where between them there is a rotating disk formed by small radial windows. When this disk moves, the radial windows in the rotating disk allow the passage of the infrared light emitted by the emitting sensor to the receiver, converting it into electrical pulses through an electronic board. Figure 24 shows the operation of a rotary encoder. (MATIAS, 2008, p.1)

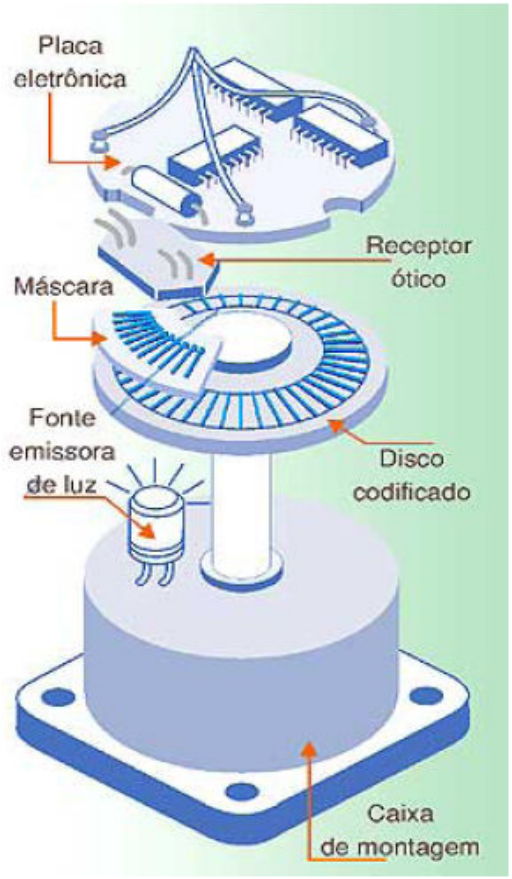

Figure 24 - Operation of the rotary encoder

Source: Matias (2008, p.1)

3 PROJECT DEVELOPMENT

The project was developed with the purpose of compacting and facilitating the control of a stepper motor, through peripherals coupled to it, where its programming is done through serial communication and its configuration is chosen by a menu developed in the "C++" programming of the microcontroller.

The peripherals coupled to the stepper motor are: an encoder, a power driver and the driver controller, allowing the connection to a computer, enabling the necessary configurations for each type of application.

Figure 25 shows the structure where the peripherals will be coupled to the stepper motor.

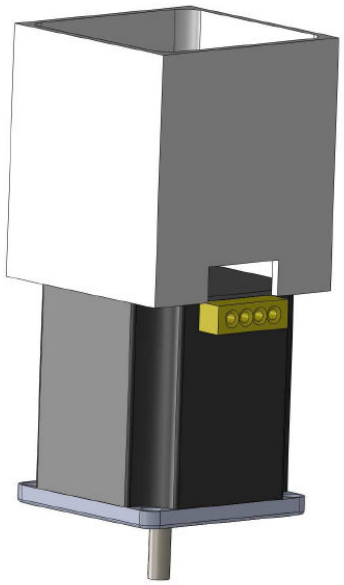

Figure 25 - Stepper Motor and Cabin Assembly for Driver's

Source: The authors

3.1 SETUP MENU

When you connect the serial communication cable to the computer, a C++ stepper motor configuration menu will appear. After opening the menu, choose the parameter you want to configure as shown in figure 26.

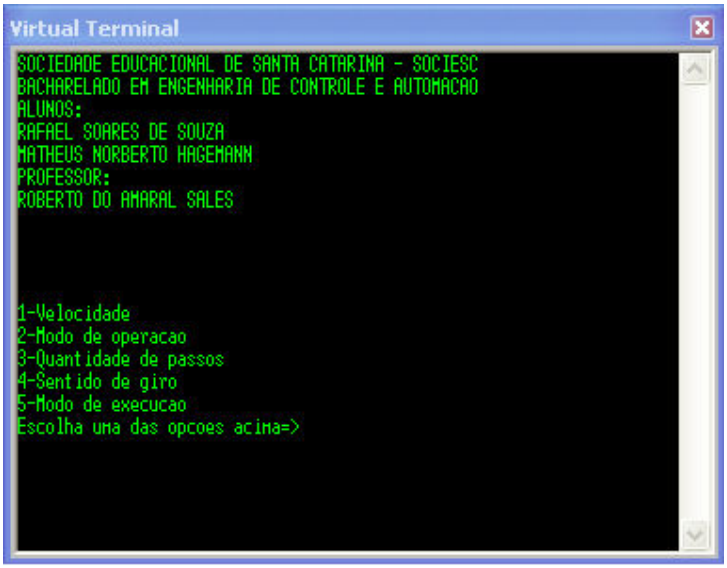

Figure 26 - Main Screen

Source: The authors

Speed:

If you choose to proceed with the speed setting, press number one (1) on the Keypad, as shown in figure 27.

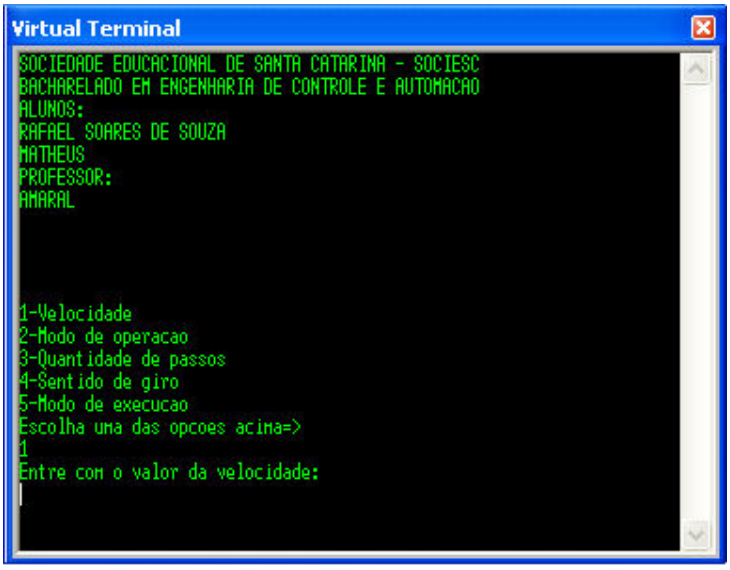

Figure 27 - Speed Parameter Screen

Source: The authors

Determine the desired value as shown in figure 28. In this case, the value of one thousand milliseconds "1000ms" of speed was set.

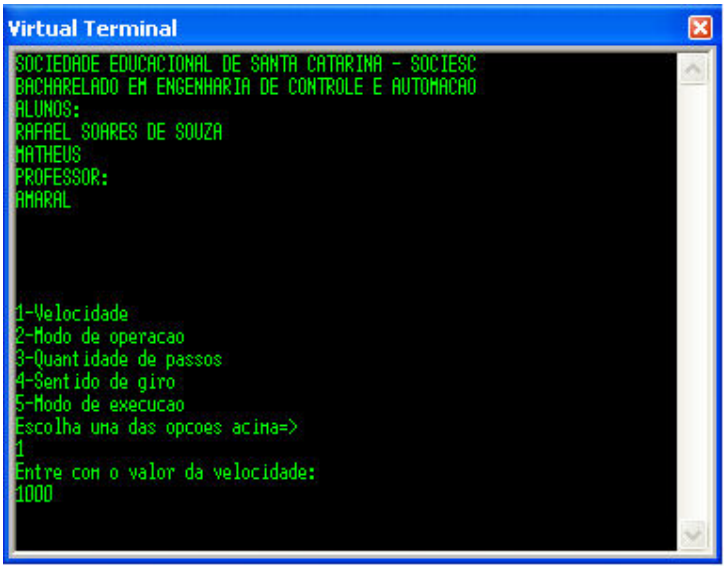

Figure 28 - Example Screen of Speed Parameter

Source: The authors

After doing this, press enter as shown in figure 29 to load the configured parameter, and return to the main screen.

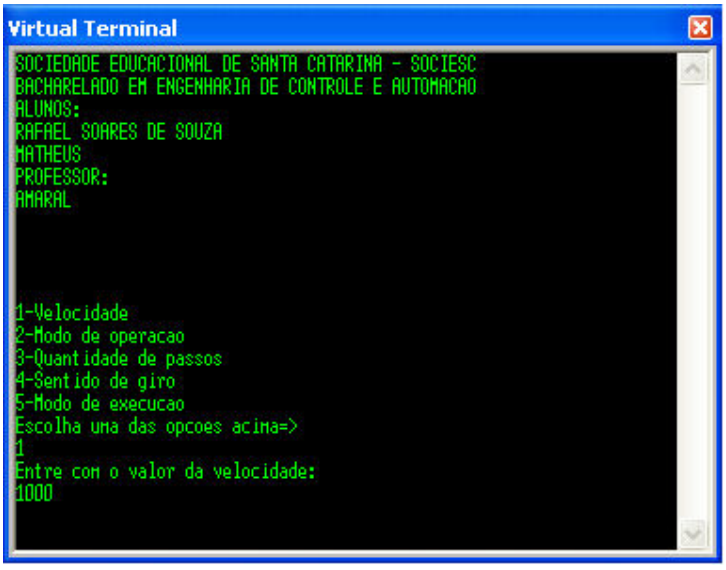

Figure 29 - Load Speed Parameter Screen

Source: The authors

Mode of operation:

In the operating mode, the stepper motor can be parameterized to work in the following modes:

•Half-step: Energizing one coil per step

•Full-step I – Energizes two coils per step

•Full-step II – It is the combination of Half-step and Full-step, energizes both one and two coils.

If you choose to proceed with the configuration of the operating mode, enter the number two (2) of the Keyboard, as shown in figure 30.

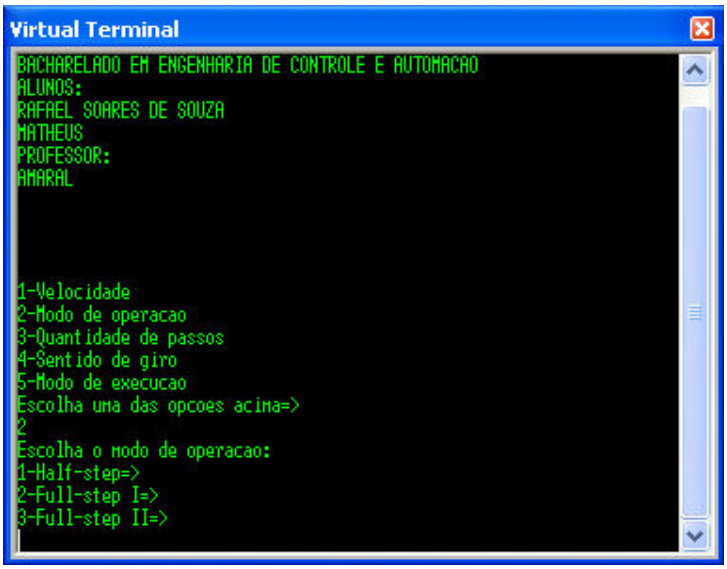

Figure 30 - Operation Mode Screen

Source: The authors

To choose one of the three operating modes, enter one (1) for Half-Step, two (2) for Full-Step or three (3) for Full-Step II. In the example below, we chose to work with the Half-Step operating mode, as shown in figure 31.

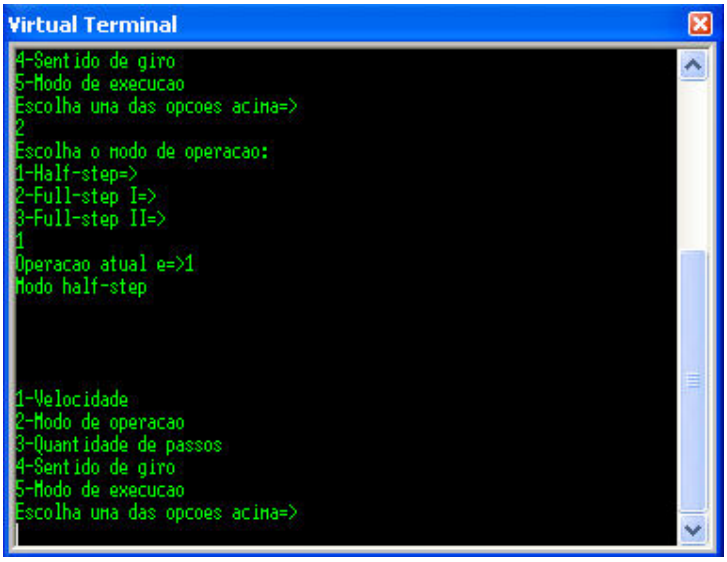

Figure 31 - Half-Step Operation Mode Screen

Source: The authors

Number of Steps:

Determines the number of steps you want to take in the operation.

The number of pulses of the stepper motor depends on the stepper angle, so the smaller this angle is, the greater the number of steps that the stepper motor can take in a 360º turn

This parameter makes it possible to choose the number of steps you want to operate.

Example:

To find out the number of steps required for the motor to make a complete 360º turn, perform the following calculations:

- Steps per turn = 360º / 1.8º
- Steps per turn = 200.

The number of steps for the engine to complete one revolution, knowing that its precision is 1.8th, it will be 200 steps.

When choosing to proceed with the step number setting, enter the number three (3) to perform the setup as per figure 32.

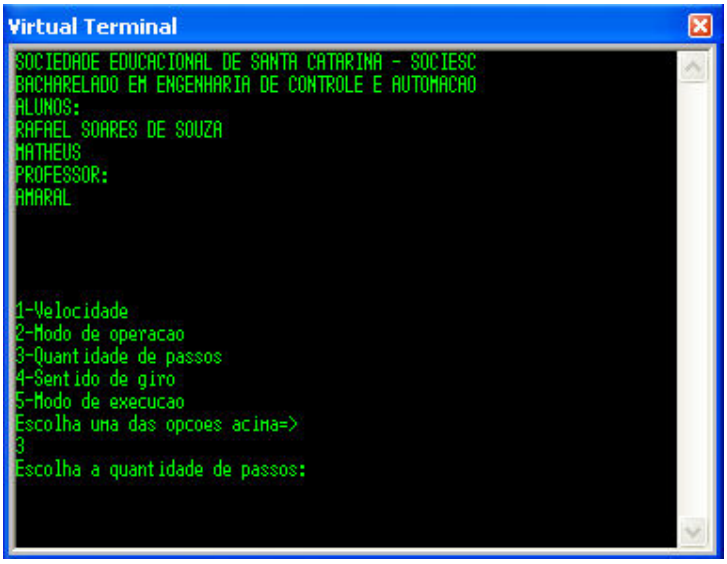

Figure 32 - Number of Steps Screen

Source: The authors

After entering the parameterization, determine the value of the step number that you want to obtain in the operation. In the example as shown in figure 33, the value 200 was put for the engine to complete one turn, that is, 360 degrees.

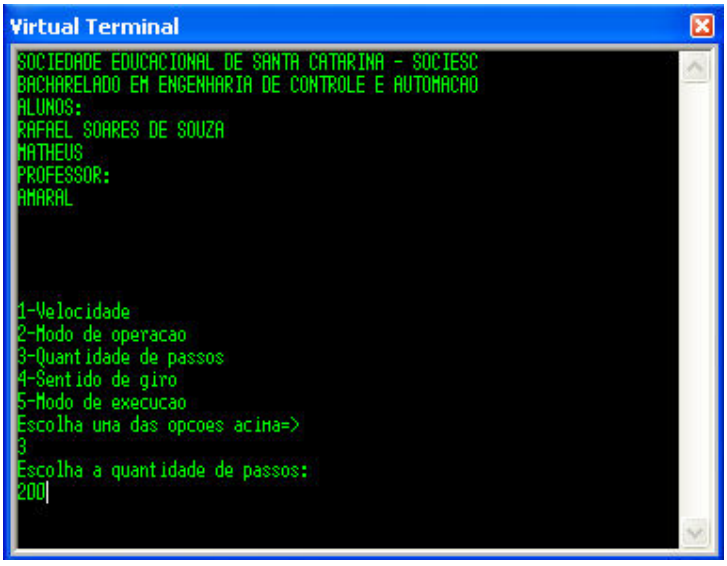

Figure 33 - Example Screen of Number of Steps

Source: The authors

After doing this, press enter as shown in figure 34 to load the configured parameter, and return to the main screen.

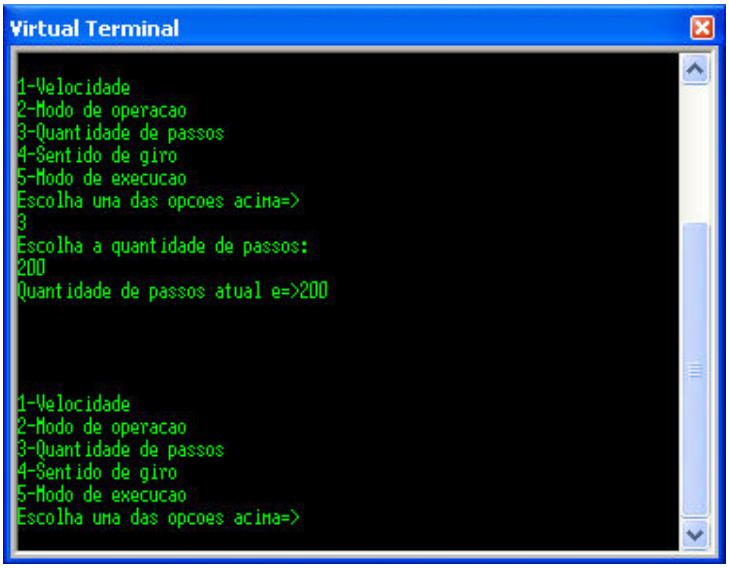

Figure 34 - Load Step Parameters Screen

Source: The authors

Direction of Turning:

Determines the direction in which you want to rotate the stepper motor.

To enter the turning direction configuration screen, enter the number four (4) as shown in figure 35.

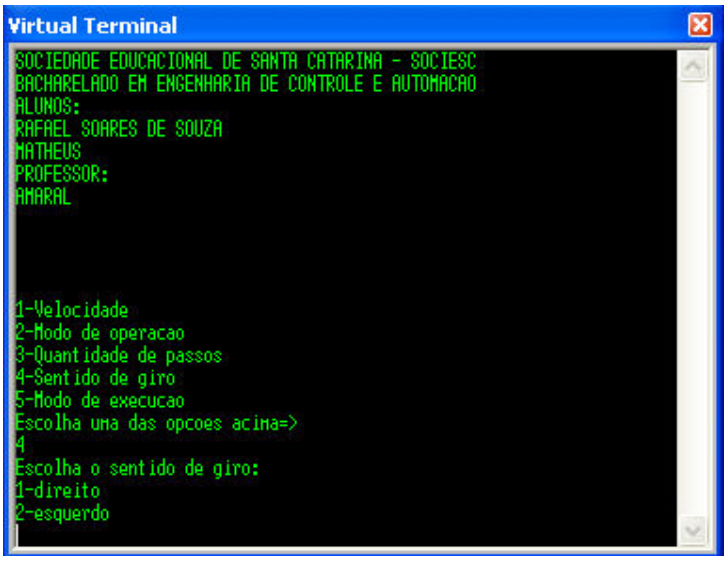

Figure 35 - Direction of Rotation Screen

Source: The authors

After doing this, enter one (1) for right direction (clockwise) or two (2) for left direction (counterclockwise), in this example option one (1) was chosen for the stepper motor to operate rotating in the right direction (clockwise), as shown in figure 36

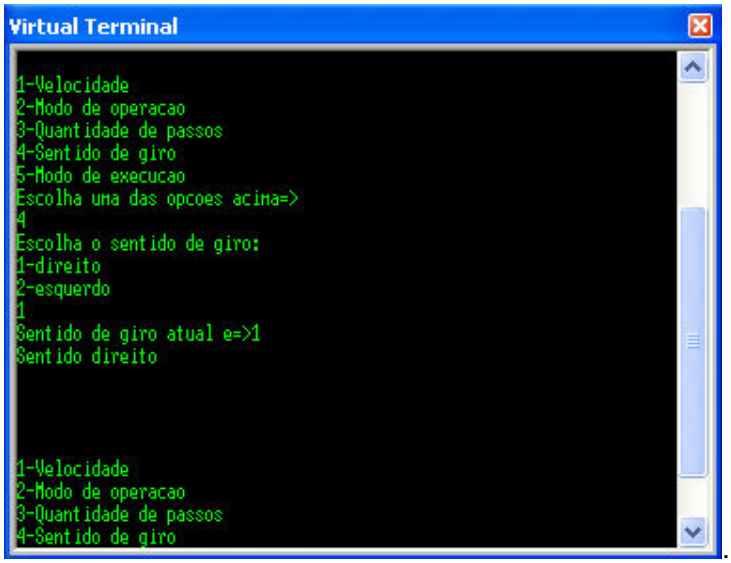

Figure 36 - Example Screen Direction of Turning

Source: The authors

Execution Mode:

It is the remote start of the configured operation. This starts the program execution cycle. After choosing all the desired parameters, enter five (5) to run the program and show the chosen settings.

Figure 37 shows the screen of the configurations that will be executed according to the parameterization seen above.

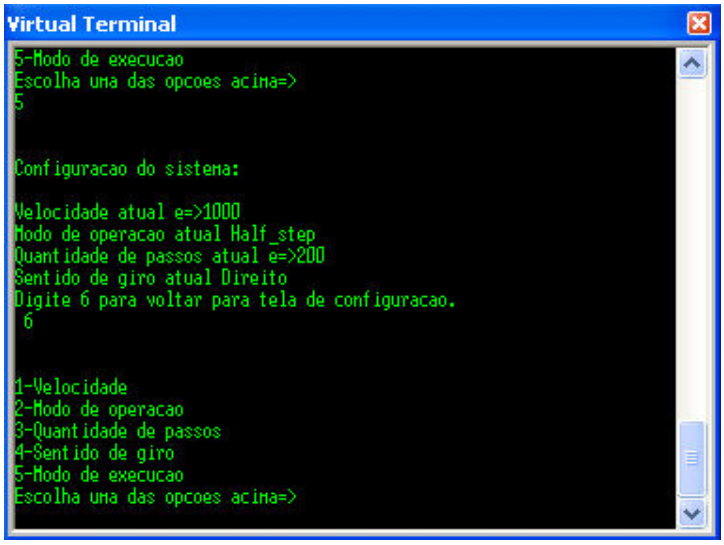

Figure 37 - Execution Mode

Source: The authors

3.2 PERIPHERALS COUPLED TO THE STEPPER MOTOR

3.2.1 Encoder

The encoder will be responsible for monitoring each step, ensuring accurate positioning. If the stepper motor happens to slip, a message will appear indicating an error.

Its assembly will be developed through an acrylic disc and two photodiodes, where the binary code generated will be printed on this disc, in such a way that it will control the position and direction of rotation of the stepper motor.

The encoder will be 400 pulses per rotation (incremental), 2 bits of precision, internal radius of 200 mm, bandwidth 100mm, with photodiode mask, its constructive form was dimensioned for this project, through the Optical Encoder wheel generator (undated) website, which generated binary code as shown in figure 38. Available at: *<http://www.bushytails.net/~randyg/encoder/encoderwheel.html>*

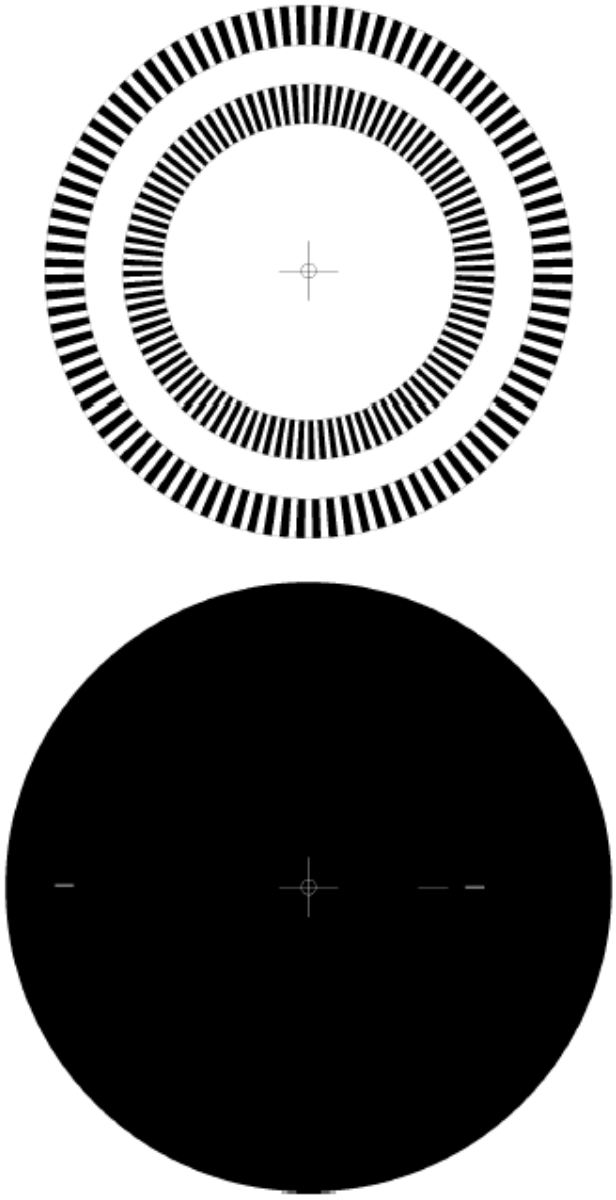

Figure 38 - Encoder Mask

Source: http://www.bushytails.net/~randyg/encoder/encoderwheel.html

3.2.2 Driver

The power driver will be responsible for driving the stepper motor. This board aims to switch the "NPN" power transistors according to the pulses received from the controller board to be able to drive the stepper motor in such a way that it supports the rated current of the stepper motor, which is three (3) amps.

The following components will be used:

- 4 NPN photodiodes

-8 x 330ohm resistors

-4 freewheel diodes

- 4 "TIP 122" type transistors supporting a current of up to 8 amperes.

•Printed circuit board with dimensions 50xmm.

Figure 39 shows the assembly of the power driver for the stepper motor.

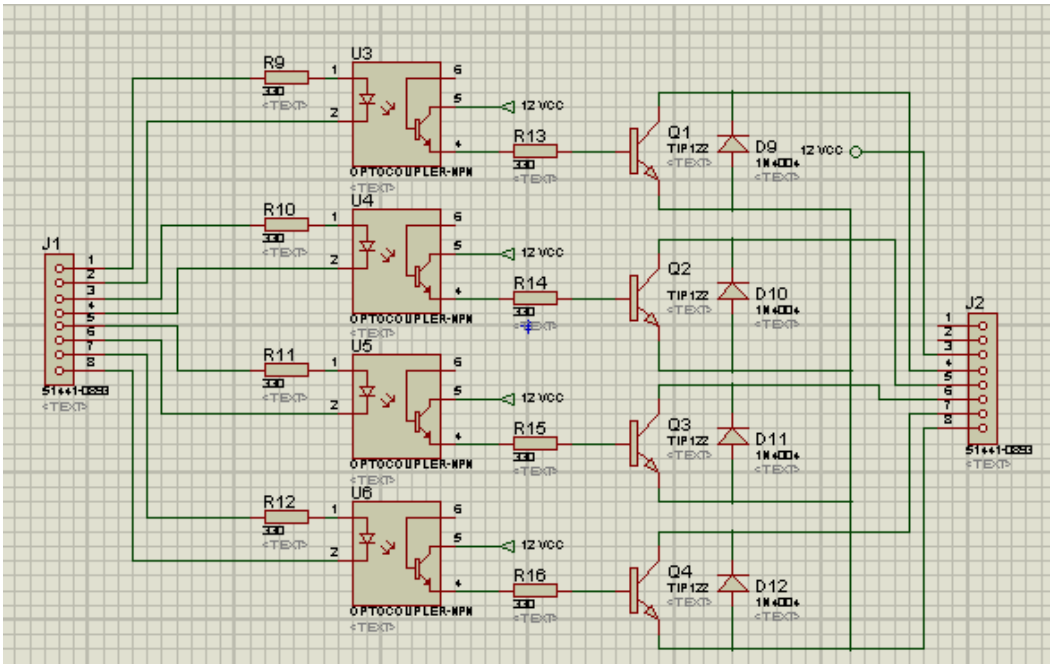

Figure 39 – Driver Board Design

Source: The authors

Figure 40 shows the rail schematic of the power driver board for the stepper motor.

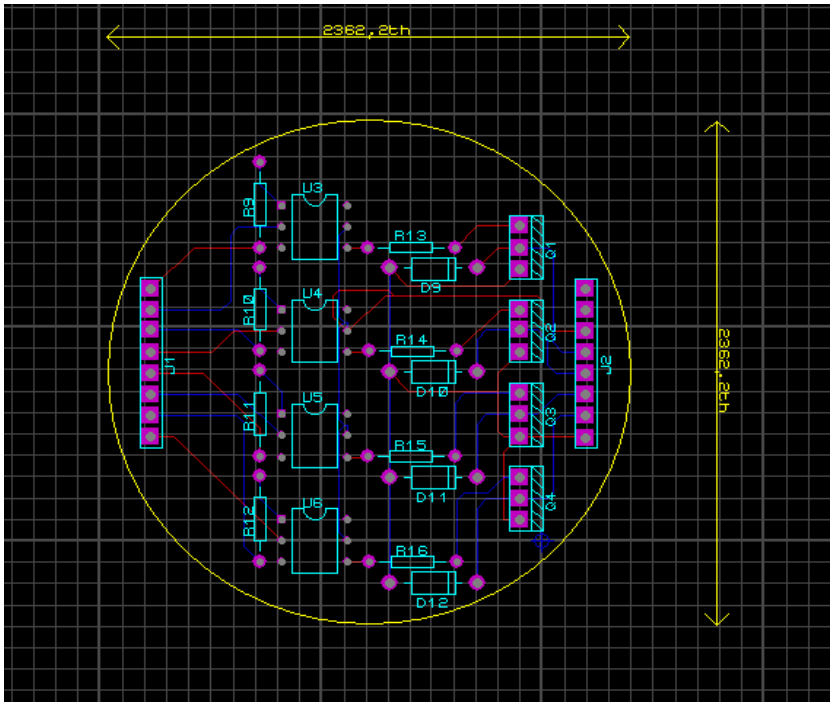

Figure 40 - Driver Card Schematic

Source: The authors

Figure 41 demonstrates the layout of the power driver for the stepper motor.

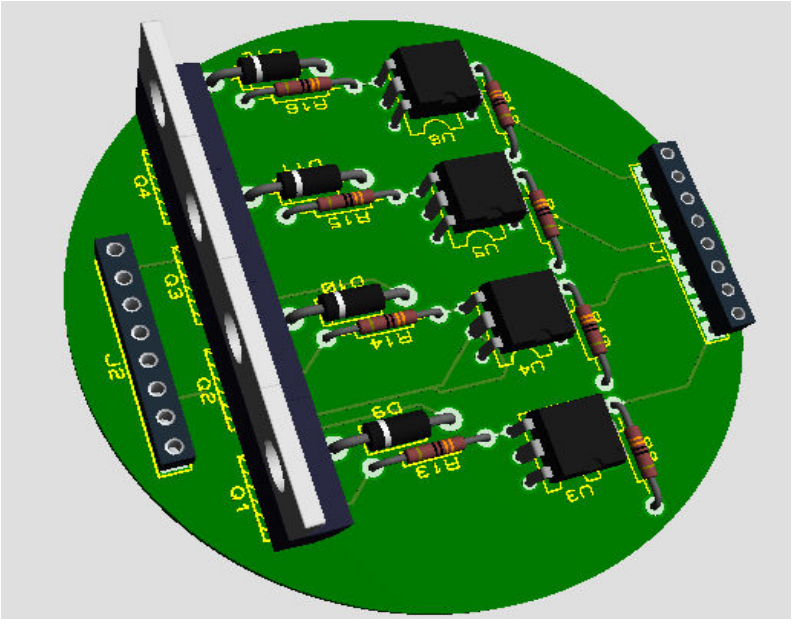

Figure 41 – Driver Board Layout

Source: The authors

3.2.3 Controller

This is what covers all the logic of the movement of the stepper motor, it will be responsible for executing the parameters on the Driver board: direction of rotation: working way, motor speed, amount of step. As parameterized in the programming menu.

The controller used will be pic 16F6248A for the purpose of recording and executing the desired programming. Figure 42 shows the design of the controller board.

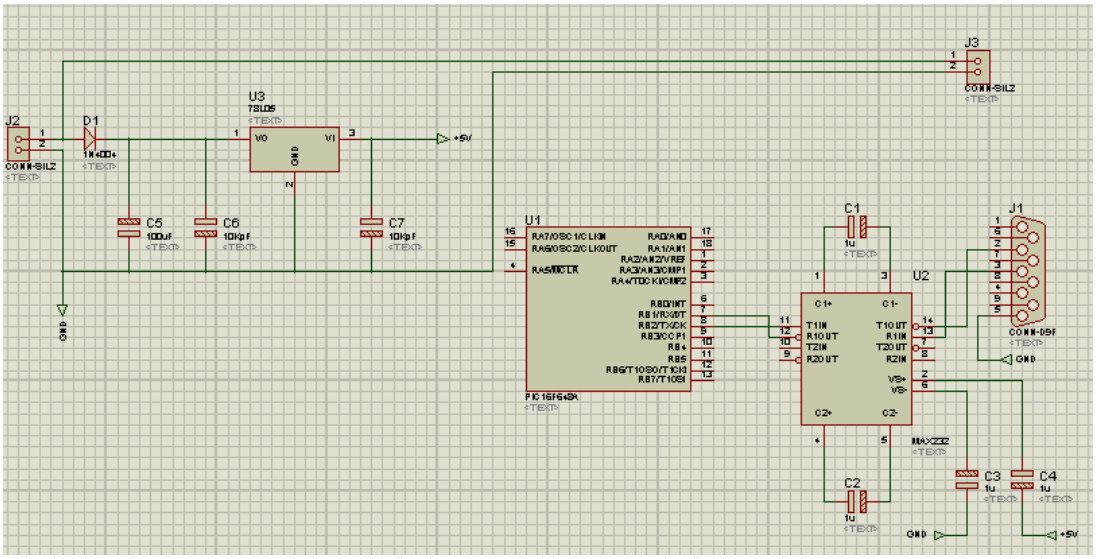

Figure 42 - Controller Board Design

Source: The authors

Figure 43 shows the rail schematic of the stepper motor controller board.

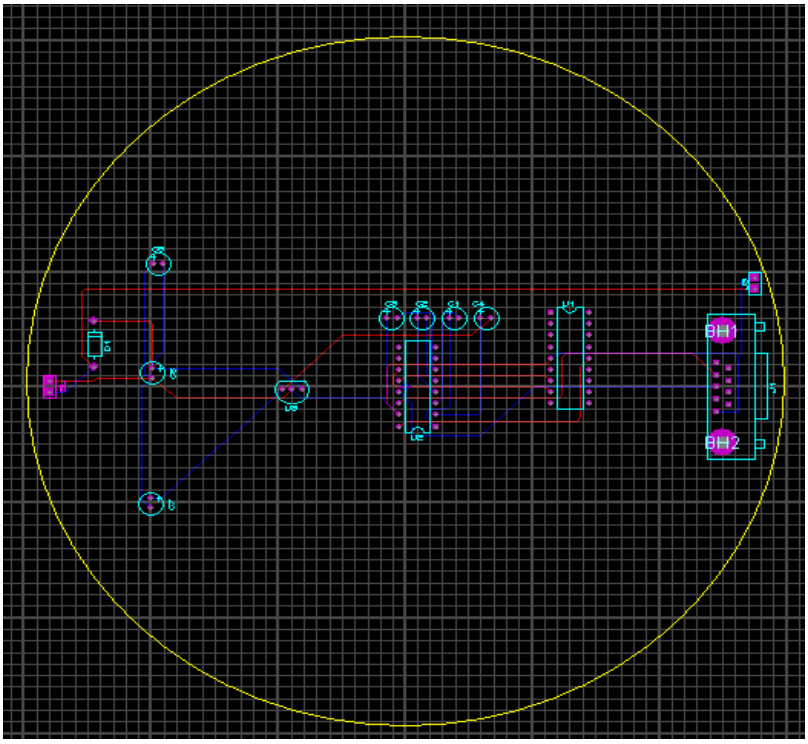

Figure 43 - Controller Board Schematic

Source: The authors

Figure 44 demonstrates the layout of the stepper motor controller.

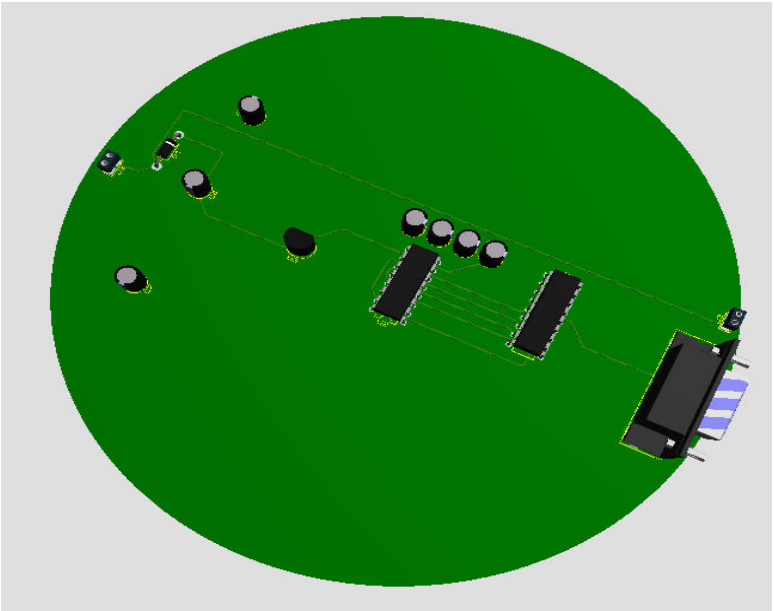

Figure 44 - Controller Board Layout

Source: The authors

3.3 BLOCK DIAGRAM

In this project, the block diagrams that demonstrate the logical sequences obtained in the project "PROGRAMMABLE STEPPER MOTOR WITH INTEGRATED DRIVER AND ENCODER" will be presented.

Figure 45 shows the parameterization sequence of Main Menu 1:

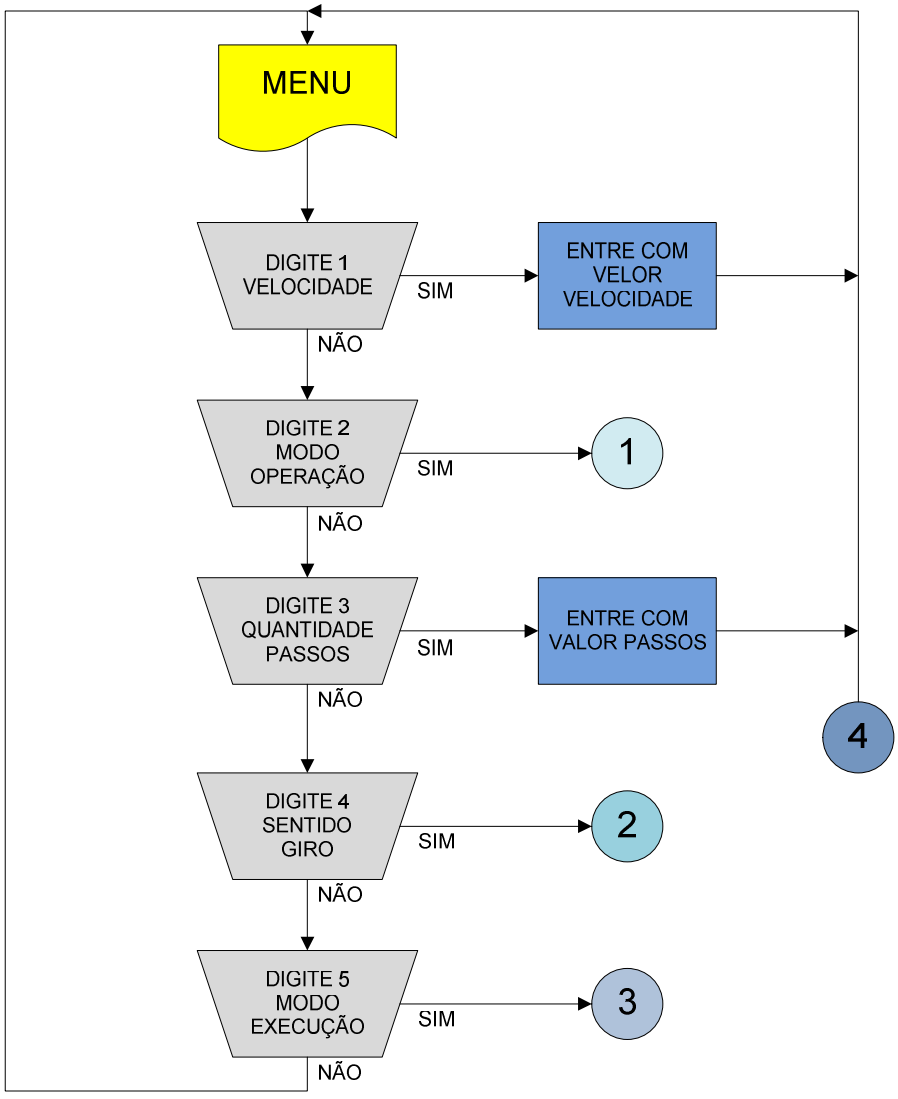

Figure 45 - Main Menu Block Diagram 1

Source: The authors

The Main Menu 2, as shown in figure 46, shows the logic obtained from the programming for the execution of the stepper motor movements:

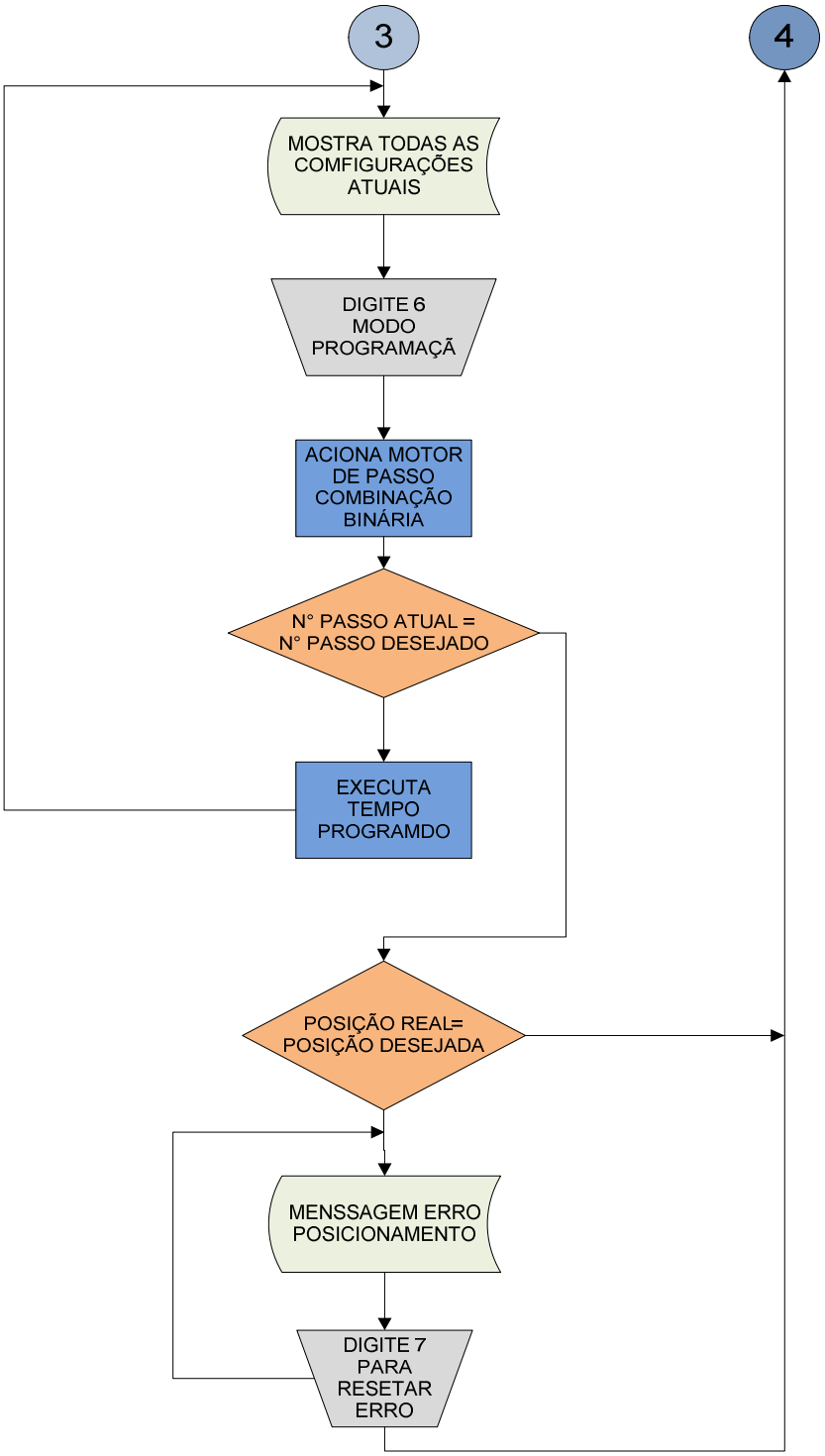

Figure 46 - Main Menu 2 Block Diagram

Source: The authors

Figure 47 shows the parameterization sequence of Main Menu 3, which defines the mode of operation that is desired to work on the stepper motor, HALL STEP, FULL STEP 1 or FULL STEP 2:

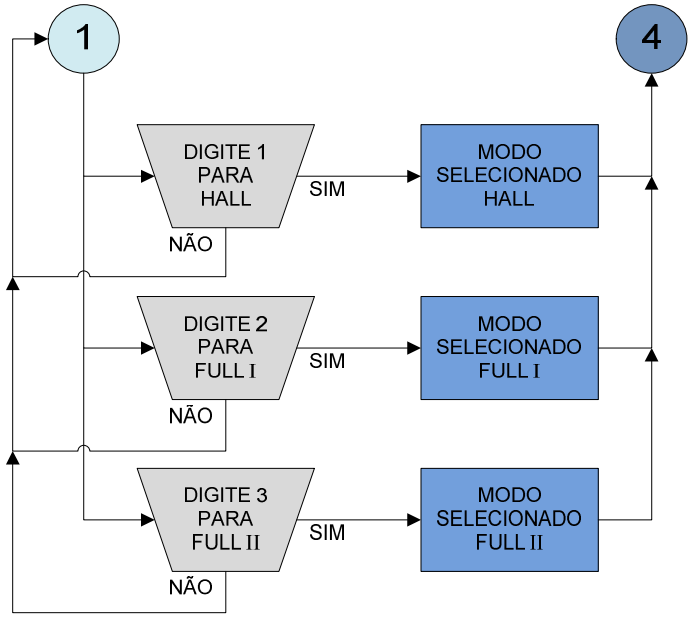

Figure 47 - Main Menu 3 Block Diagram

Source: The authors

Figure 48 shows the parameterization sequence of Main Menu 4, which defines the direction of rotation of the stepper motor, right or left:

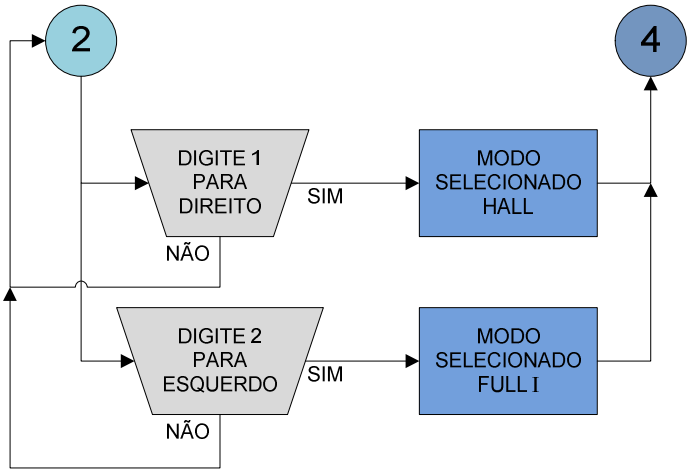

Figure 48 - Main Menu 4 Block Diagram

Source: The authors

Legend of the block diagram as shown in figure 49:

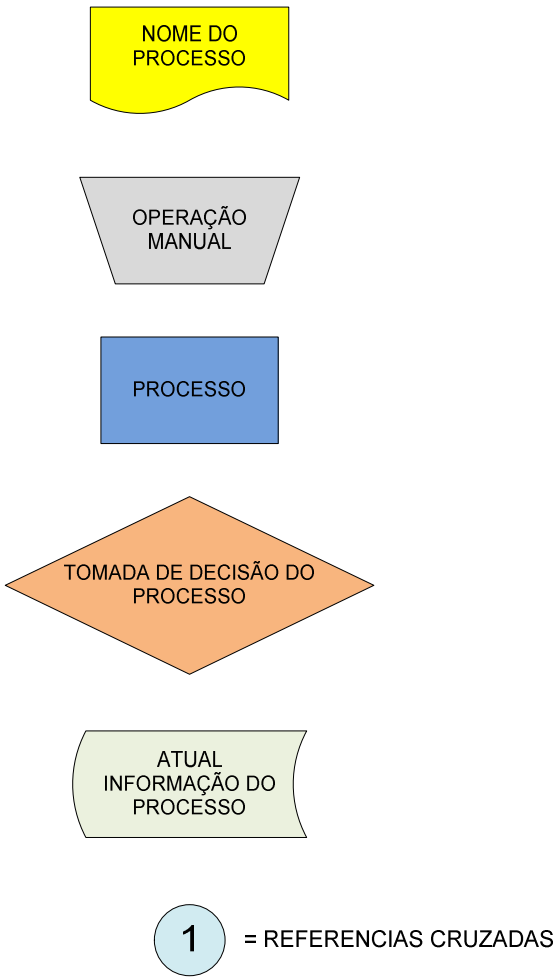

Figure 49 - Block Diagram Legend

Source: The authors

3.4 PROGRAMMING

Here is the code of the program developed in C++:

```
#include "Motor_Passo_Serial.h"
#include <stdlib.h>
#FUSES NOWDT //No Watch Dog Timer
#FUSES XT //Crystal OSC <= 4MHz
#FUSES NOPUT //No Power Up Timer
#FUSES NOPROTECT //Code not protected from reading
#FUSES NOBROWNOUT //No brownout reset
#FUSES NOLVP //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD //No EE protection
#use delay(clock=4000000)
#use rs232(baud=9600,parity=N,xmit=PIN_B2,rcv=PIN_B1,bits=8)
int step[8]={0b00000001,
             0b00000011,
             0b00000010,
             0b00000110,
             0b00000100,
             0b00001100,
             0b00001000,
             0b00001001};

int step1[4]={0b00000001,
              0b00000010,
              0b00000100,
              0b00001000};

int passo2[4]={0b00000011,
              0b00000110,
              0b00001100,
              0b00001001};
signed int pos;
```

```
unsigned int16 w_num,c_num,cont;
int y,z,d,v,h,x;
char c[5],w[13],modo_1[13],modo_2[13],modo_3[13],sentido_1[13],sentido_2[13],
sense[13],mode[13];
int letter=0;
void presentation(void);
void menu(void);
main void(void);
void main
{
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_ccp1(CCP_OFF);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  pos=0;
  output_a(step[pos]);
x=0;
presentation();
while (true)
{
if (x==0)
{
cont=c_num;
menu();
}
else
{
main();
}
if (kbhit())
{
letter=getc();
```

```
if (letter==0x36)
{
x=0;
}
}
}
}
////////////////////////////////////////////////////////////////////////////
Void Presentation (Void)
{
printf("SOCIEDADE EDUCACIONAL DE SANTA CATARINA - SOCIESC\n\r");
printf("BACHELOR'S DEGREE IN CONTROL AND AUTOMATION ENGINEERING\n\r");
printf("STUDENTS:\n\r");
printf("RAFAEL SOARES DE SOUZA\n\r");
printf("MATHEUS NORBERTO HAGEMANN\n\r");
printf("TEACHER:\n\r");
printf("ROBERTO DO AMARAL SALES \n\r");
printf("\n\r");
delay_ms(5000);
}
////////////////////////////////////////////////////////////////////////////
void menu(void)
{
of the
{
printf("\n\r");
printf("\n\r");
printf("\n\r");
printf("1-Velocity\n\r");
printf("2-Mode of operation\n\r");

printf("3-Number of steps\n\r");
printf("4-Direction of turn\n\r");
```

```
printf("5-Execution mode\n\r");
printf("Choose one of the options above=>\n\r");
y=getchar();
}
while ((y<'1')||(y>'5'));
   switch(y)
   {
 Case '1':
 printf("\n\r");
 printf("Enter the velocity value:\n\r");
 gets(w);
 w_num=atoll(w);
 printf("Current speed e=>");
 puts(w);
 printf("\n\r");
 Break;
  Case '2':
 printf("\n\r");
 printf("Choose the mode of operation:\n\r");
 printf("1-Half-step=>\n\r");
 printf("2-Full-step I=>\n\r");
 printf("3-Full-step II=>\n\r");
 z=getchar();
 printf("\n\r");
 printf("Current operation e=>");
 putchar(z);
  switch(z)
  {
  Case '1':
  printf("\n\r");
  printf("Half-step mode\n\r");
  printf("\n\r");
  h=1;
  modo_1="Half_step";
```

```
strcpy (mode,modo_1);
Break;
Case '2':
printf("\n\r");
printf("Full-step mode I\n\r");
printf("\n\r");
h=2;
modo_2="Full_step_I";
strcpy (mode,modo_2);
Break;
Case '3':
printf("\n\r");
printf("Full-step mode II\n\r");
printf("\n\r");
h=3;
modo_3="Full_step_II";
strcpy (mode,modo_3);
Break;
}
Break;
  Case '3':
printf("\n\r");
printf("Choose the number of steps:\n\r");
gets(c);
c_num=atoll(c);
printf("Current number of steps and=>");
puts(c);
printf("\n\r");
Break;
 Case '4':
printf("\n\r");
printf("Choose the direction of turn:\n\r");
printf("1-right\n\r");
printf("2-left\n\r");
```

```
d=getchar();
printf("\n\r");
printf("Current turning direction e=>");
putchar(d);
printf("\n\r");
  Switch(d)
 {
 Case '1':
 printf("Right sense\n\r");
 printf("\n\r");
 v=1;
 sentido_1="Right";
 strcpy (sense,sentido_1);
 Break;
 Case '2':
 printf("Left sense\n\r");
 printf("\n\r");
 sentido_2="Left";
 strcpy (sense,sentido_2);
 v=2;
 Break;
 }
Break;
 Case '5':
printf("\n\r");
printf("\n\r");
printf("\n\r");
printf("System Configuration:\n\r");
printf("\n\r");
  printf("Current speed e=>");
puts(w);

printf("Current operating mode %s",mode);
printf("\n\r");
```

```
 printf("Current number of steps and=>");
 puts(c);
 printf("Current turning direction %s",direction);
 printf("\n\r");
 printf("Enter 6 to go back to the setup screen.\n\r ");
 x=1;
 Break;
 }
}

////////////////////////////////////////////////////////////////////////////
Main Void
{
while (h==1)
    {
        if (v==1)pos++,cont--;
    if (v==2)pos--,cont--;
    if (post > 7) post = 0;
    if (pos < 0) pos = 7;
    output_a(step[pos]);
    if (cont==0)
    {
    x=0;
    }
    delay_ms(w_num);
         Break;
}
while (h==2)
{
    if (v==1)pos++,cont--;
    if (v==2)pos--,cont--;
    if (pos > 3) pos = 0;
    if (pos < 0) pos = 3;
    output_a(step1[pos]);
```

```
    if (cont==0)
    {
    x=0;
    }
    delay_ms(w_num);
    Break;
}
    while (h==3)
{
    if (v==1)pos++,cont--;
    if (v==2)pos--,cont--;
    if (pos > 3) pos = 0;
    if (pos < 0) pos = 3;
    output_a(step2[pos]);
    if (cont==0)
    {
    x=0;
    }
    delay_ms(w_num);
    Break;
}

}
```

4 SCHEDULE

The timeline below (figure 50) demonstrates the sequence for the development of the project

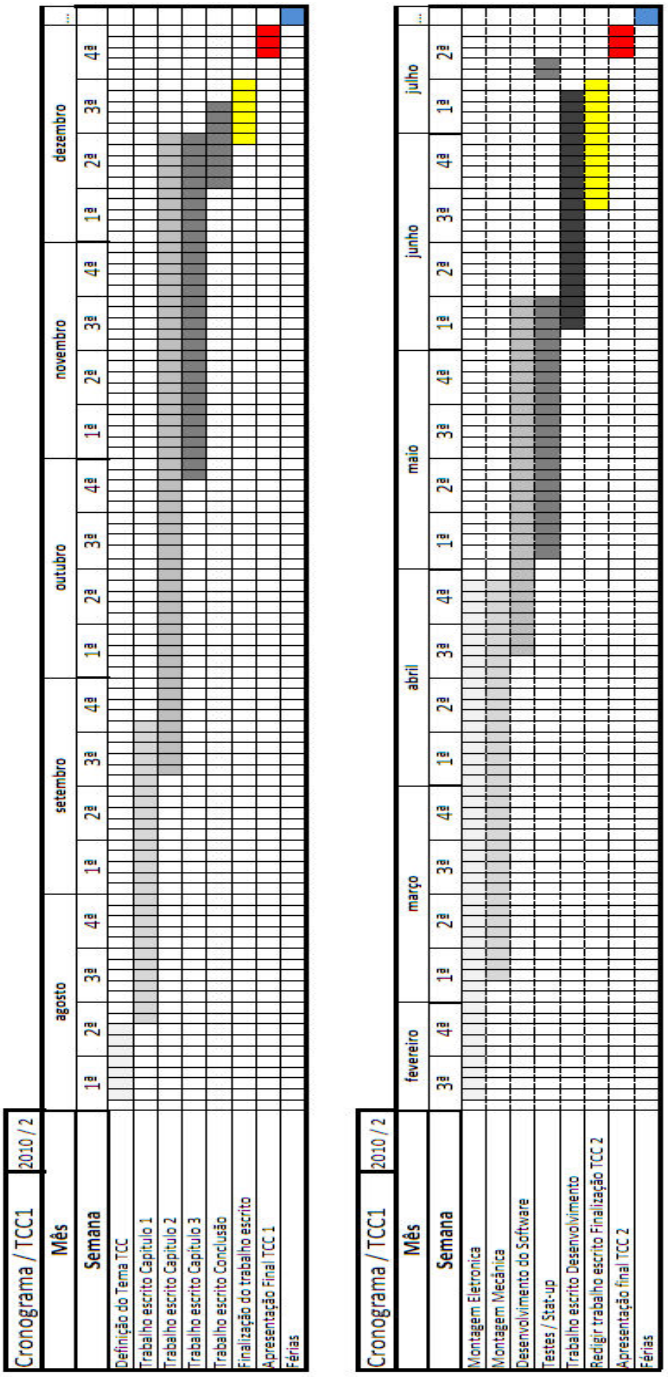

Figure 50 - Schedule

Source: The authors

5 CONCLUSION

The study for the elaboration of this project allowed me to acquire a broad knowledge about the operation of stepper motors. And the increase in the understanding and applicability of microprocessors using C++ programming.

Stepper motors, although a cheaper solution for industry, are still little used in precision applications compared to servo motors. However, it is observed that its closed-loop applicability would make it very reliable and that it is still possible to greatly reduce costs in a product that offers precision in displacement in stepper motors.

The many products offered in the industrial market in general can and should be explored, maximized to cover the highest level of applications possible, in this way, their implementation is facilitated and the cost is lowered.

REFERENCES

BRAGA, Newton C. **Serial communication in the industry using the RS-232 protocol.**

Available at: <http://www.mecatronicaatual.com.br/secoes/leitura/333> - Accessed on: 18 set. 2010.

BRITES, Felipe G. and SANTOS, Vinicius P. A. **Stepper Motor** - Fluminense Federal University. Available at: <http://www.telecom.uff.br/pet/petws/downloads/tutoriais/stepmotor/stepmotor2k81119.pdf > - Accessed on: 20 set. 2010.

CANZIAN, Edmur. **Serial communication - RS232.** Available at: <http://www.professores.aedb.br/arlei/AEDB/Arquivos/rs232.pdf> - Accessed on: 20 Sep. 2010.

CATÁLAGO. Datasheet Max 232 texas instruments. n.d.

IDEV. **RS 232**. Available at: http://ldev.wordpress.com/2009/04/18/rs-232/ - accessed on: 20 Sep. 2010.

KENJO, T. **Stepping motors and their microprocessor controls**. 1. ed. Oxford University Press, 1986.

LYRA, Pablo Vinicius Apolinário. **Development of a didactic CNC milling machine**. (2010). Available at: <http://alvarestech.com/temp/cnc/Fresadora%20CNC%20Did%E1tica.pdf> – Accessed on: 20 Oct 2010.

MATIAS, Juliano. **Encoders**. Available at: <http://www.mecatronicaatual.com.br/secoes/leitura/281> - Accessed on: 06 Oct 2010.

MESSIAS, Antonio Rogério. **Stepper motor control through parallel port**. Available at: <http://www.rogercom.com/pparalela/IntroMotorPasso.htm> - Accessed on: 02 Oct 2010.

MICROCHIP **Technology. Microchip.** Available at: http://www.microchip.com/ - Accessed on 20 Sep 2010.

NEWTON C. Braga. **Network automation**. Available at: <http://www.mecatronicaatual.com.br/secoes/leitura/333 / Automation - Networks> - Accessed on 20 Oct 2010.

OPTICAL Encoder Wheel Generator. Available at: <http://www.bushytails.net/~randyg/encoder/encoderwheel.html> - Accessed on: 06 out. 2010.

PEREIRA, Fábio. **Pic microcontrollers:** advanced techniques. 5. ed. São Paulo: Érica Ltda, 2007.

PEREIRA, Eduardo Pereira and LAGES, Walter Fetter. **Integration of signals and data**. Available at: <http://alvarestech.com/temp/simprebal/Relatorios_Tecnicos-

Publications-Dissertations/docs/courses/Aula2-integracaosinais-apostila-cap3.pdf> - Accessed on: 20 Sep. 2010.

SITE. Available at: <www2. eletronica.org/artigos/outros/estudo-do-motor-de-passo-e-seu-controle-digital> - Accessed on: 06 out. 2010.

SOMESSARI, Samir Luiz. **Automation of the laser welding process (Nd:YAG) for the manufacture of iodine-125 seeds used in brachytherapy**. Available at: <http://pelicano.ipen.br/PosG30/TextoCompleto/Samir%20Luiz%20Somessari_M.pdf> Accessed on: 20 out. 2010.

SOUZA, David, J. DE. **Breaking the PIC** – Amplified and updated to pic 16f628a. São Paulo: Editora Érica Ltda., 2007.

SOUZA, Marco Antonio. **Implementation of closed-loop stepper motor controller systems using technology based on digital signal controller.** Final paper. São Paulo: São Carlos School of Engineering, 2007.

SOUZA, Vitor Amadeu. **RS232 and RS485 communication**. Available at: <http://www.cerne-tec.com.br/Comunica%E7%E3o232485.pdf> - Accessed on: 20 out. 2010.

TECNICAL English. **Stepper motor basics**. Available at: <www.sapiensman.com/ESDictionary/docs/d6.htm> - Accessed on: 06 out. 2010.

WIKIPEDIA. **C++ standard library.** Available at: <http://pt.wikipedia.org/wiki/Biblioteca_padr%C3%A3o_do_C%2B%2B> – Accessed on 18 Sep. 2010.