Department of Computer, Informatics and Telecommunications Engineering

Thesis:

Two Axis CNC - Hardware and Software Development

Using Atmega 328p Microcontroller

By

Iakovou K. Petros

Mechanical Engineer

MS: Advance Industrial and Manufacturing Systems

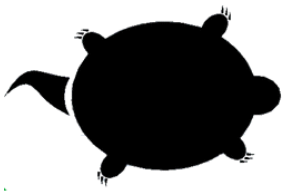
Supervisor: Dr.John Kalomiros (Professor)

September 2021

Declaration:

I, Iakovou Petros, declare that I am the author of this thesis and the entire work represented in this paper, is the outcome of personal labor. The original paper is written in Greek Language and has been deposited in the Institute's Library. Anywhere in this thesis, mentioned work which has been done by others, is cited in the reference section.

Iakovou Petros



The only limit in creativity

is your imagination

Abstract

The following paper is a thorough summary of the final project for the Department of Computer, Informatics and Telecommunications Engineering, of International Hellenic University. The project focus on:

1) The development of a CAM software, which reads CAD files, extract coordinates from geometrical shapes and generates a text file (.nc) with the corresponding G code.

2) Hardware and Software development (planning and implementation), of a two axis CNC.

3) The implementation of a linear and Circular Interpolation Algorithm.

Key Words: CAD, CAM , CNC , Numerical Control Interpolation

YouTube link: https://www.youtube.com/watch?v=j8LD028rm2o

contact: berxt@hotmail.com

Table of Contents

Table of Images

Table of Tables

Table of Mechanical Drafts

Table of Codes

Abbreviations

CAD: Computer Aided Design
CAM: Computer Aided Manufacturing
CW: Clock Wise
CCW: Counter Clock Wise
CNC: Computerized Numerical Control
CTC: Clear Timer on Compare
PWM: Pulse Width Modulation

Part 1 : From CAD TO CAM

1.1 Computer Aided Design (CAD)

The ISO 6983 **[1]** describes the structure of code which designated for numerical control machines. The standard is known as G code and (among others) contains the coordinates of geometrical structures in a linear drawing. So, the first task which must be accomplished is to produce the G code which describes a 2D drawing from the representative CAD file.

By saving a CAD drawing in its native type, a file which contains the entire information, coded in binary form, is generated (for AutoCAD is .dwg). The drawback of such files is that they cannot be utilized by other software applications. For that reason, CAD software companies provide an alternative way for saving drawings in order to make them portable (for AutoCAD is .dxf). These files contain the information defined by the user in ASCII form and so, are readable by a human. In the current application .dxf files will be used to extract, coordinates and data, of geometrical entities (line, arc, circle) from the 2D drawing which is going to be processed.

The .dxf (**D**rawing **E**xchange **F**ormat) files developed from Autodesk in order to make 2D drawings utilizable by other applications. Today is the most common way to transfer the information that describes a 2D drawing. Autodesk provides a documented paper **[2]** in which thoroughly describes how the information organized and represented in a .dxf file. Generally data that describes a specific feature, indicated by a unique integer called group code. In other words, the group code is the identity of the consequent data. The tables below illustrate the data that represented by some group codes, which will be used in the current application.

- Arc

Group Code	Data Type
100	Type (Class AcDbCircle)
10	Center of arc inX axis (OCS)
20	Center of arc in Y axis (OCS)
30	Center of arc in Z axis (OCS)
100	Type (Class AcDbArc)
50	Angle between X axis and straight line from Start point to the Center
51	Angle between Y axis and straight line from Start point to the Center

Table 1.1 - Group Code for Arc

- Circle

Group Code	Data Type
100	Type (Class AcDbCircle)
10	Center of circle in X axis (OCS)
20	Center of circle in Y axis (OCS)
40	Radius

Table 1.2 - Group Code for Circle

- Line

Group Code	Data Type
100	Type (Class AcDbLine)
10	Start point in X axis (wcs)
20	Start point in Y axis (wcs)
30	Start point in Z axis (wcs)
11	End point in X axis (wcs)
21	End point in Y axis (wcs)
31	End point in Z axis (wcs)

Table 1.3 - Group Code for Line

1.2 Coordinates mining

In order to read the .dxf file and obtain the data of geometrical entities, an executable program (named "**CAD2G Code**"), have been developed, (C++).

To store the data in variables a class "**Entity**" generated with the following structure (Code 1.1).

entity.h

```
/*
name: CAD2G code (in DEV-C++ 5.11)
by: Iakovou Petros
Date: 05-05-2021
Class entity used to store data of geometrical entities (line, cirlce, arc)
of geometrical structures in a .dxf file
circle will handle as clockwise arc with indentical start and end point
*/

#ifndef ENTITY
#define ENTITY

class entity
{
    private:
            char type[2];               //  G01 if Line, G02 if Circle, G03 if Arc
            float x1,y1,x2,y2,I1,J1;    //  x1,y1: initial entity's coordinates,
                                        //  x2,y2  final entity's coordinates,
                                        //  I1,J1  center of arc, relative to initial point (x1,y1)
    public:
            entity();                   // constructor without arguments
            entity(char type_1[],float x_1,float y_1,float x_2,float y_2,float I_1,float J_1); // constructor with arguments
            void SetLine(char type_1[],float x_1,float y_1,float x_2,float y_2);  // create an entity that describes line
            void SetArc(char type_1[],float x_1,float y_1,float x_2,float y_2,float I_1,float J_1); //create an entity that describes arc
            void PrintLine();   // display line's data
            void PrintArc();    // display arc's data
            char *Get_type();   // return the geometry of entity
            float Get_x1();     // return variable x1
            float Get_y1();     // return variable y1
            float Get_x2();     // return variable x2
            float Get_y2();     // return variable y2
            float Get_I1();     // return variable I1
            float Get_J1();     // return variable J1
};

#endif
```

Code 1.1 - Class Entity

By executing the program, the user prompted to give the .dxf drawing which is going to be processed. If the file exist the program opens and reads the given .dxf file, line by line. Every time the cursor meets a geometrical entity (this mean, that after group code 100, reads string = "AcDbLine", or "AcDbArc" or "AcDbCircle") the program creates an object of the class Entity and stores the data that describes it (in case of arc, further trigonometric calculations required). By reading the entire file ,as many objects of the class Entity have been created, as the number of geometric entities that assembly the 2D drawing.

1.2 Align Geometrical Entities

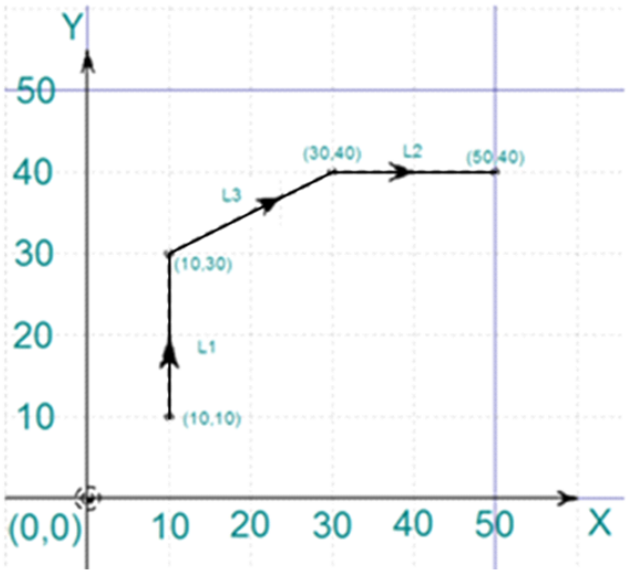
The problem: The order in which the entities have stored, by reading the .dxf file is the same as the order in which the drawing have been formed. Furthermore, as far as the arc concern, it is always illustrated as counter clockwise (G03) independently if it initially had been generated as CW (G02) or CCW(G03), while in real application both CW and CCW arcs, used.

For example, the draft 1.1 contains three geometrical shapes with the following order

1) Line 1:P1(10,10) -> P2(10,30)

2) Line 2:P1(30,40) -> P2(50,40)

3) Line 3:P1(10,30) -> P2(30,40)




```
N001 G00 X10 Y10
N002 G01 X10 Y30 F20
N003 G00 X30 Y40
N004 G01 X50 Y40 F20
N005 G00 X10 Y30
N006 G01 X30 Y40 F20
N007 M30
```

Draft 1.1 - Machine Process following the order in which the entities have been created in the 2D Design

The G code that describes the above draft consist of 6 routing commands (G00 and G01) and so the header (pen, laser, drill, nozzle or other) has to turned on three times (transition from G00 to G01) and other three times to turned off (transition from G00 to G01). In most cases this is not a recommended method because it brings out many friction wears and spoil the smoothness of part's surface (for example a pen plotter release more ink in the beginning after staying unused or a laser cutter generates a burning when starts to cut).

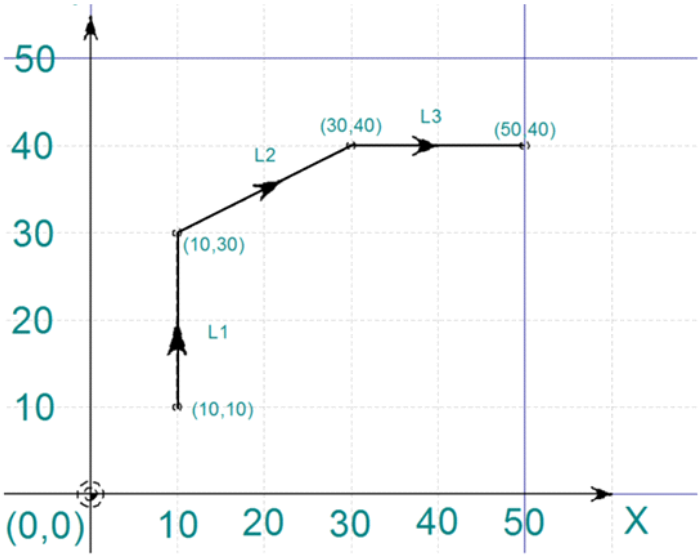
If we overlook the sequences in which the entities have been drawn and if we change the order so that to be continuous, (where this is feasible) ,it is possible to reduce the number of commands in G code and consequently the execution time and the header's on / off alterations.

In draft 1.2 Line 3 and Line 2 have been swapped.

1) Line 1:P1(10,10) -> P2(10,30)

2) Line 2:P1(10,30) -> P2(30,40)

3) Line 3:P1(30,40) -> P2(50,40)



N001 G00 X10 Y

N002 G01 X10 Y30

N003 G01 X30 Y40

N004 G01 X50 Y40

N005 M30

Draft 1.2 - Machine Process after reorder the entities position

The G code is now only 4 routing commands and the header activated and deactivated one time. In real application with thousand entities this alignment, drives to a significant operational improvement.

The following three cases have been taken under consideration:

case1: The end point of an entity E(i) coincide with the end point of another entity E(j), draft 1.3.

If E(j) is:

Straight Line:

1) Swap the start and end point of E(j)

Arc:

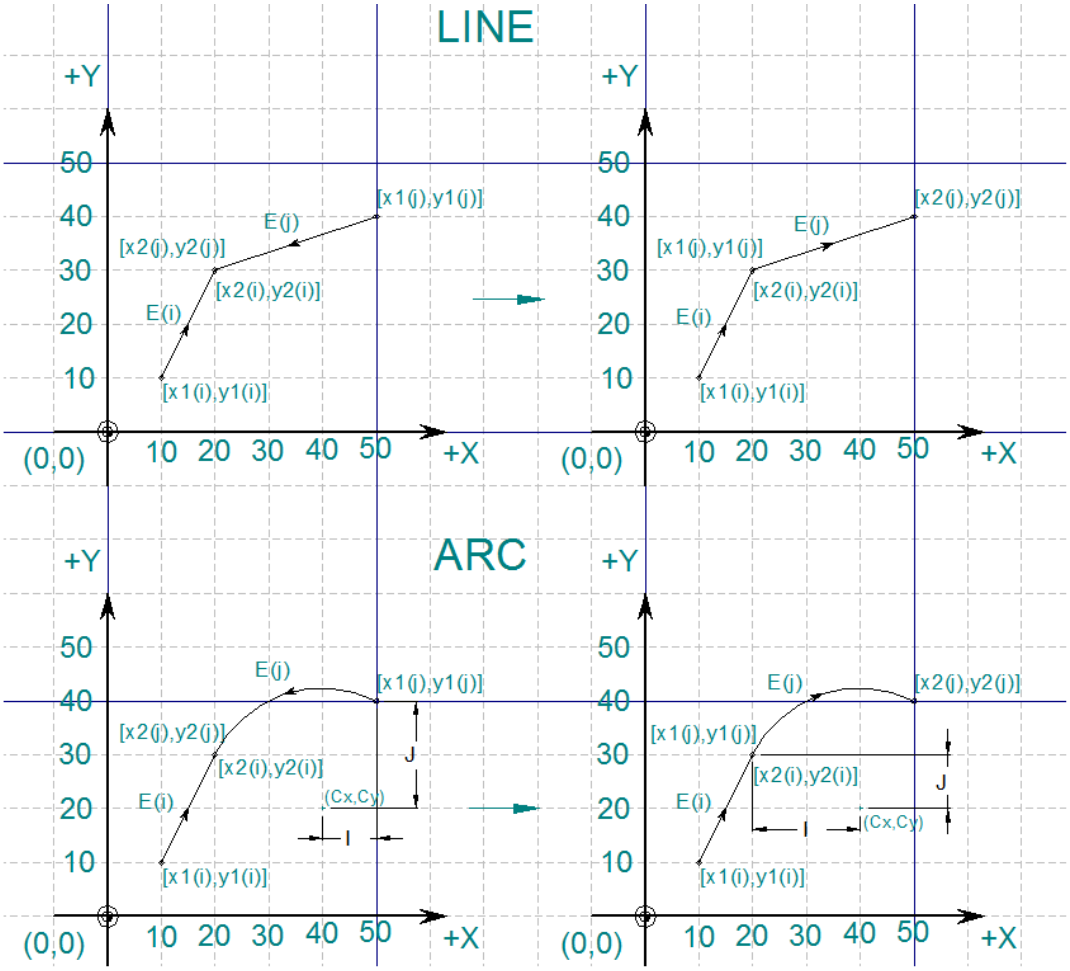
1) Change the rotation direction from counter clockwise (CCW) G03 into clockwise (CW) G02

2) Swap the start and end point of E(j)

3) Calculate the relative coordinates of the new start point in regard to arc's center by using the following equations:

I=Cx-x1

J=Cy-y1



Draft 1.3 - The end point of an entity E(i) coincide with the end point of another entity E(j)

case2: The start point of an entity E(i) coincide with the start point of another entity E(j), draft 1.4

if E(j) is:

Straight Line:

1) Swap the start and end point of E(j)

2) Swap entities E(i) and E(j)

Arc:

1) Change the rotation direction from counter clockwise (CCW) G03 into clockwise(CW) G02

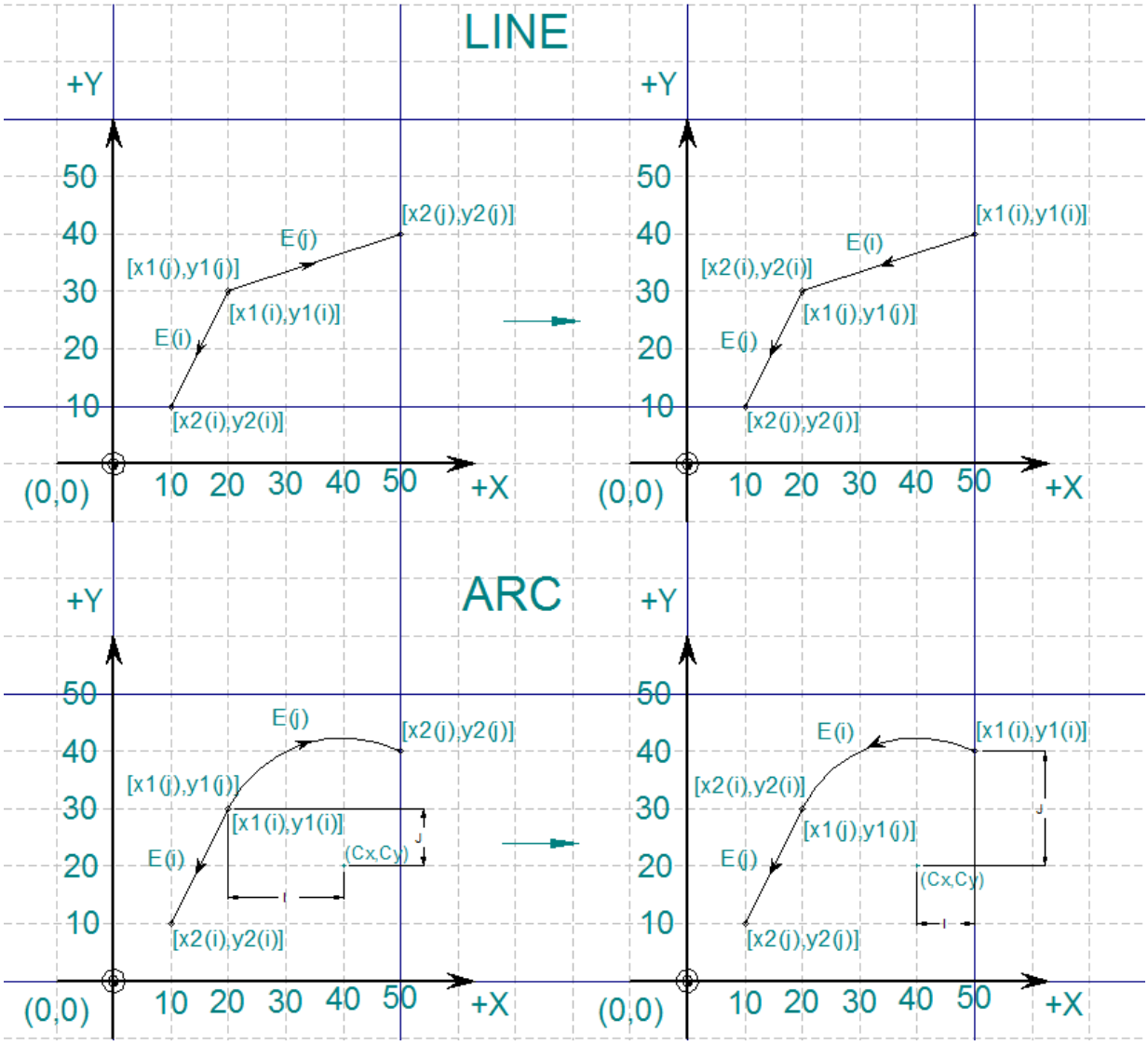
2) Swap the start and end point of E(j)

3) Calculate the relative coordinates of the new start point in regard to arc's center by using the following equations:

$$I=Cx-x1$$

$$J=Cy-y1$$

4) swap entities E(i) and E(j)

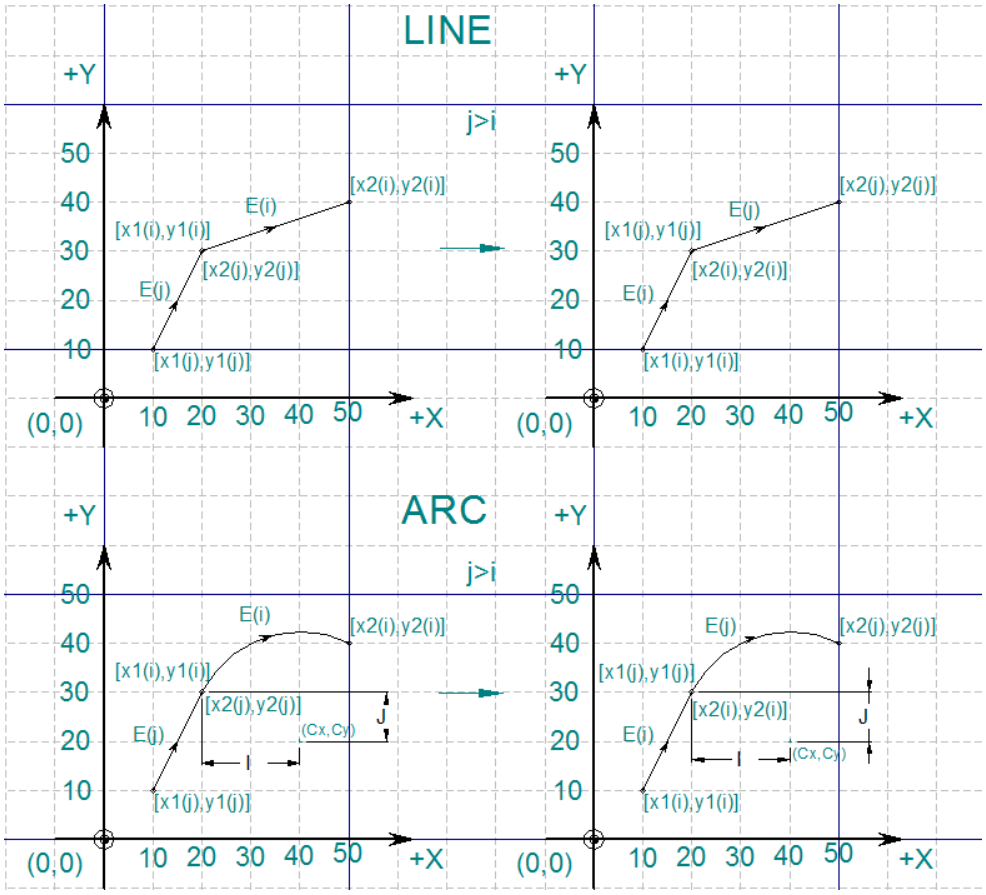


Draft 1.4 - The start point of an entity E(i) coincide with the start point of another entity E(j)

case3: The start point of an Entity E(i) coincide with the end point of a following entity E(j) (j>i), σχέδιο 1.11.

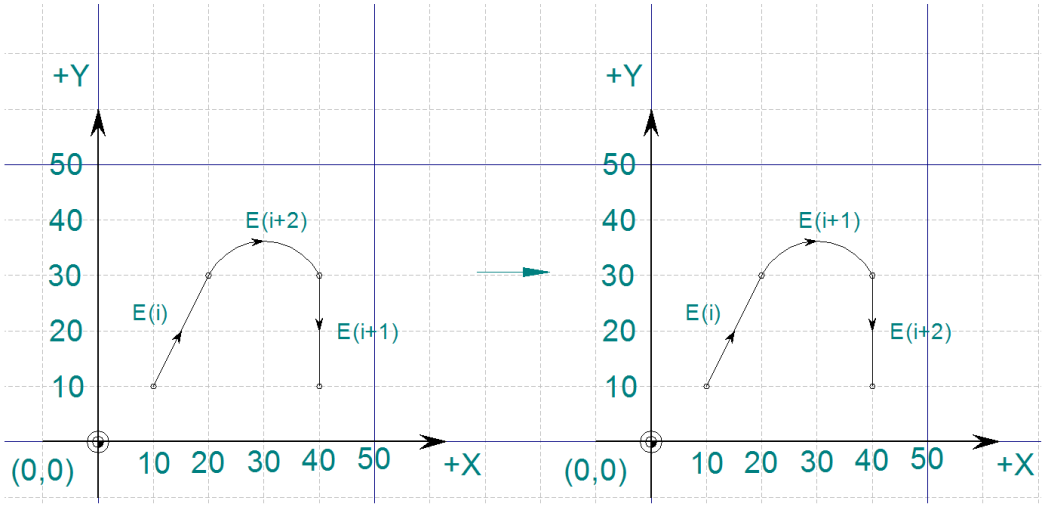
<u>always, regardless the entity's type (Straight Line or Arc):</u>

1) Swap entities E(i) and E(j)



Draft 1.5 - The start point of an Entity E(i) coincide with the end point of a following entity E(j) (j>i)

alignment: if in any of the above cases, even just one swap between two entities occurs, the program executes a scanning to the list of entities to check if it needs to reposition any of them (draft 1.6).



Draft 1.6 - align entities

The Image 1.1 is the algorithm's flow chart as described above and in page 65 is the code in C++ (Appendix A1).

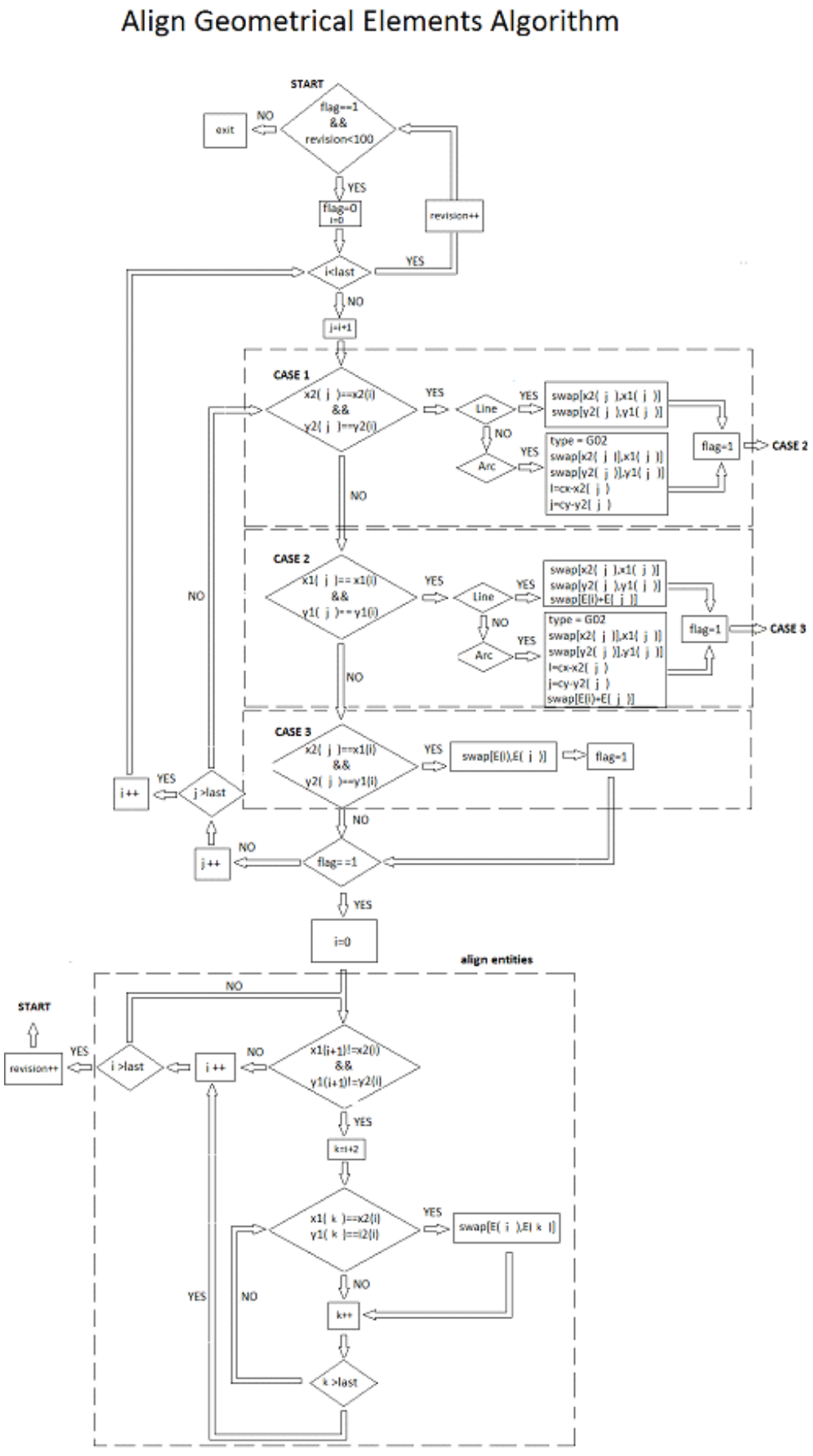


Image 1.1 - Align Entities Algorithm Flow Chart (Appendix A1)

By executing the align geometrical entities algorithm the program use the entities to create a text .nc file with the G code which represent the 2D drawing by implement the following rule:

When the last point of an entity (i) does not coincide with the start point of the next entity (i+1), it means that we have movement without process (G00 command) with the maximum velocity into the start point of i+1 entity. While if the two points are identical, it means that we have movement with process (G01 or G02 or G03), with feed rate velocity in the end point of i+1 entity.

In image 1.2 there is an actual implementation of the algorithm in a 2D draft (ex03.dxf). Nineteen entities E[xx] in this sketch have been generated in a random sequence. An index, gives the order in which the entities have been created. First E[0] and Last E[18] (E[0] → E[1] → .... → E[18]). The G code for this sketch as it is (without entity alignment) has 70 commands and many passages from G00 to G01 and reversely.

The algorithm, align the entities so to be continuous making a G code with 25 commands and only two passages from G00 to G01 and reversely. The order of entities in the new sketch has change accordingly figure (1.3)

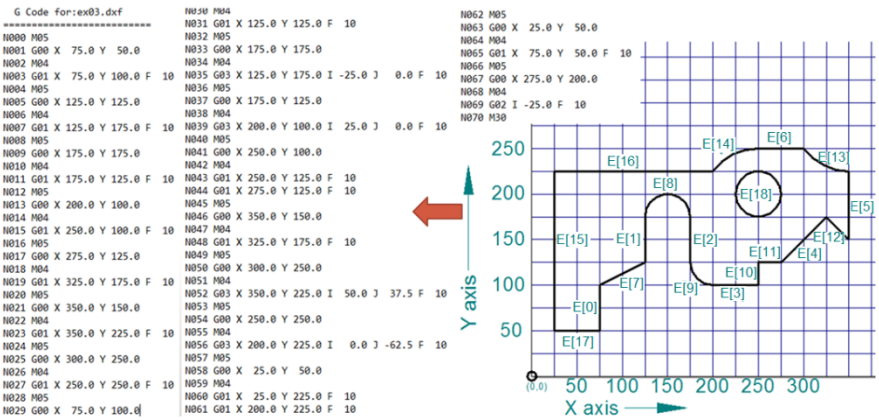


Image 1.2 - Align Entities Algorithm Example (before alignment)

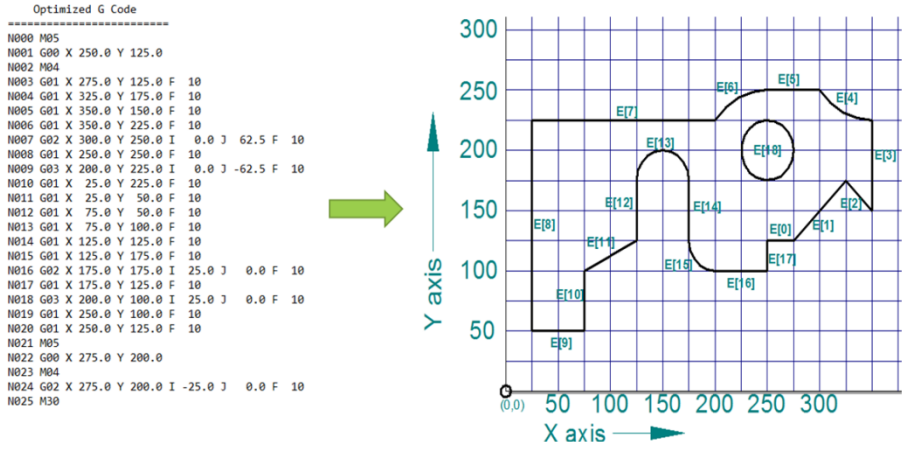


Image 1.3 - Align Entities Algorithm Example (after alignment)

Part 2: The Hardware

The hardware separated into two main parts. 1. The mechanical hardware and 2. the electronic hardware.

2.1 Mechanical hardware:

2.1.1 The transmission method

The mechanical design of a Computerized Numerical Control Machine has many different approaches. The major question that has to be answered from the very beginning is the transmission method (the mechanism that convert the motor's rotary motion into linear)

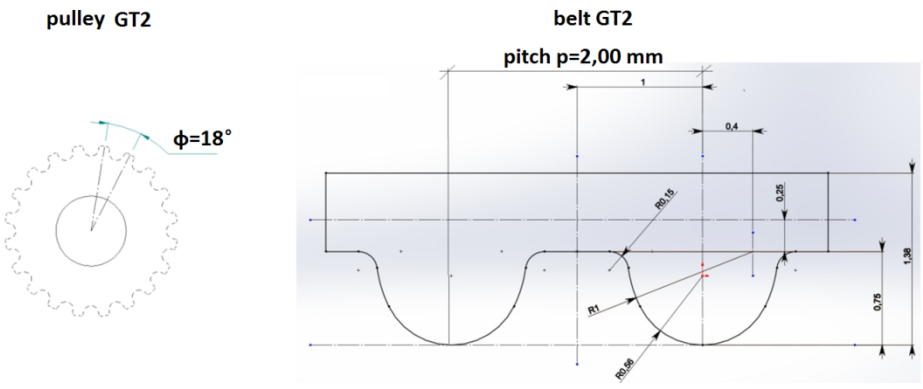
Two options are available, 1. Belt driven or 2. Screw driven. The choice depends from the application's demands. The table below, summarize the pros and cons for each method.

Belt VS Screw driven transmission method		
Feature	Belt	Screw
Velocity	+	-
Accuracy	-	+
Backslash	-	+
Cost	+	-
Design Complexity	+	-
Locking Position	-	+
Travel length	+	-
Acceleration	-	+
Load	-	+

Table 2.1 - Belt driven VS Screw transmission method

Generally screw driven method is better for applications with greater accuracy and horizontal movement while belt driven method preferred where bigger velocity and travel length required.

In the current application the transmission is the consequence of the collaboration between belt and pulley. The type of belt is GT2 with the following features (draft 2.1).



Draft 2.1 - GT2 belt and pulley

Having a pulley with 20 teeth, the angle revolution for a pitch (p=2 mm) is $\varphi=360^o/20=18^o$. This means that for linear movement equal to one pitch (and in GT2 this is 2,00 mm) the pulley has to rotate 18^o. And so the equation that gives the linear movement dx for a given angle 'a' is:

$$dx = \frac{2a}{\varphi} = \frac{2a}{18}\ (mm)\ (2.1)$$

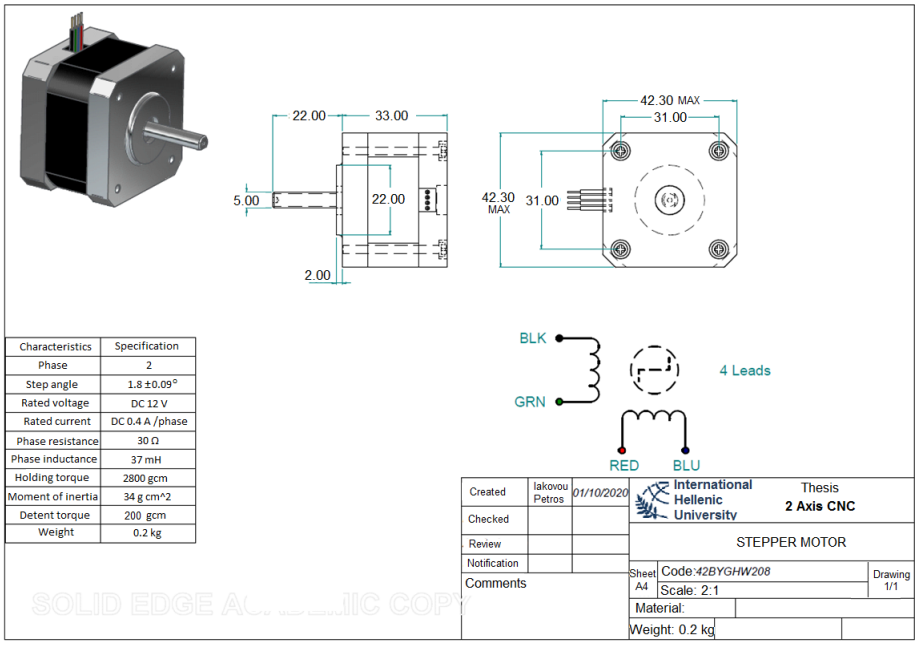
2.1.2 The type of motor

The next design decision, which is also determined by the nature of the application, is the type of motor, it will be used. The decision has to be taken between servo and stepper motor. The table below summarize some basic characteristics for each type.

Step Motor	Servo Motor
Open - loop	Closed loop
No Feedback system (more prone to error)	Internal Feedback system (less prone to error)
Less expensive	More expensive
Smaller	Bigger
High torque at low speeds	Lower torque at low speeds
Very low torque at higher speeds	High torque at higher speeds
No encoder	Encoder and gearbox for accurate control
Lower speed	Higher speed
No vibration or pulsation at standstill position	Pulsate or vibrate at standstill position

Table 2.2 - Step motor VS Servo motor

For the current application the 42BYGHW208 bipolar stepper motor **[3]** used with the following features.



Draft 2.2 - Technical specifications of Step motor used in application

The size of the motor, depends from the demand in detent torque. One parameter, for the torque's calculation, requires the knowledge of the max load, the motor has to carry. To estimate this load the mechanical design of the CNC machine required.

2.1.3 The Mechanical Design:

For the mechanical design, Solid Edge (Siemens) used as 3D software.

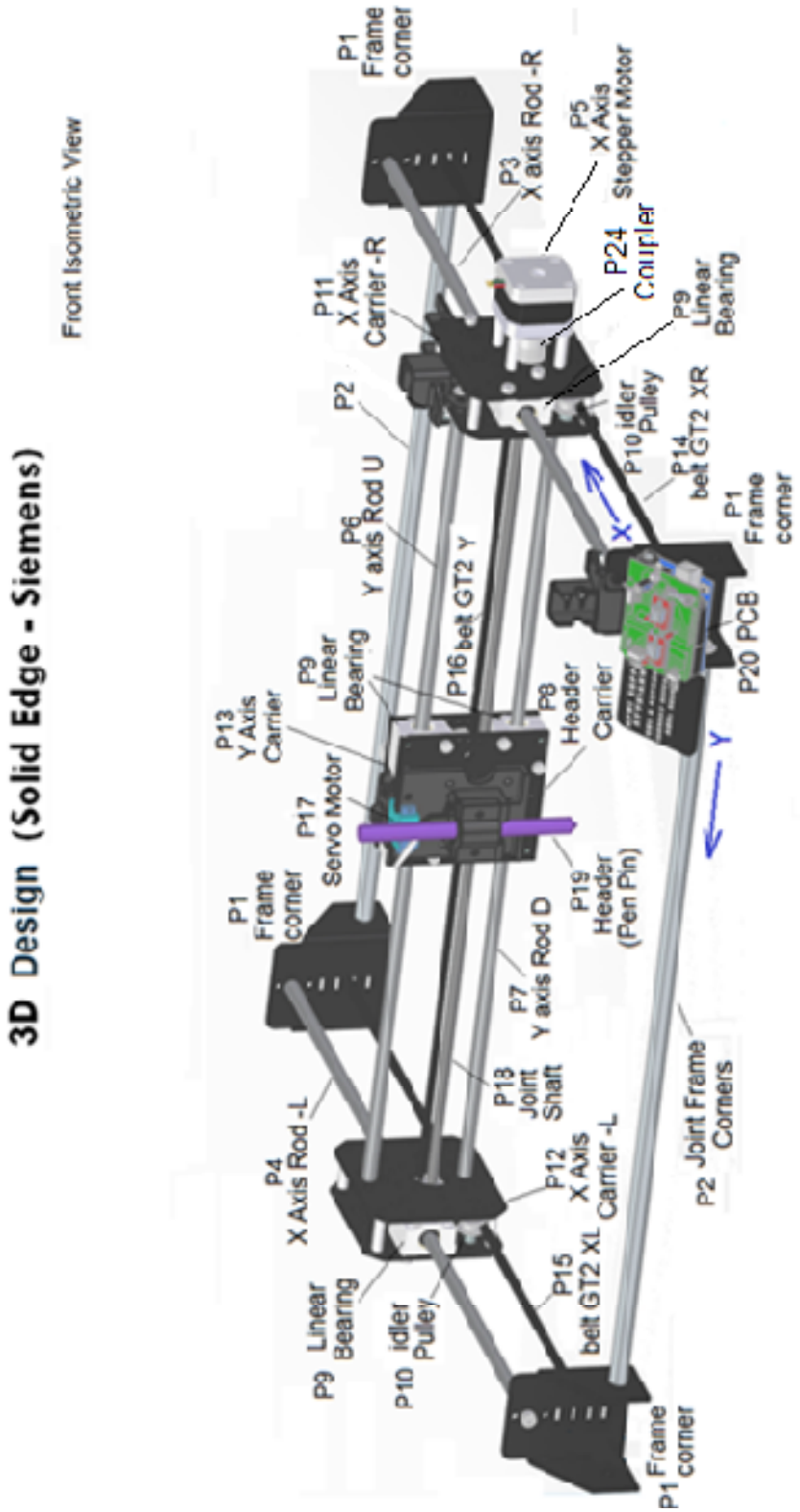


Image 2.1 - 3D Design Front Isometric View

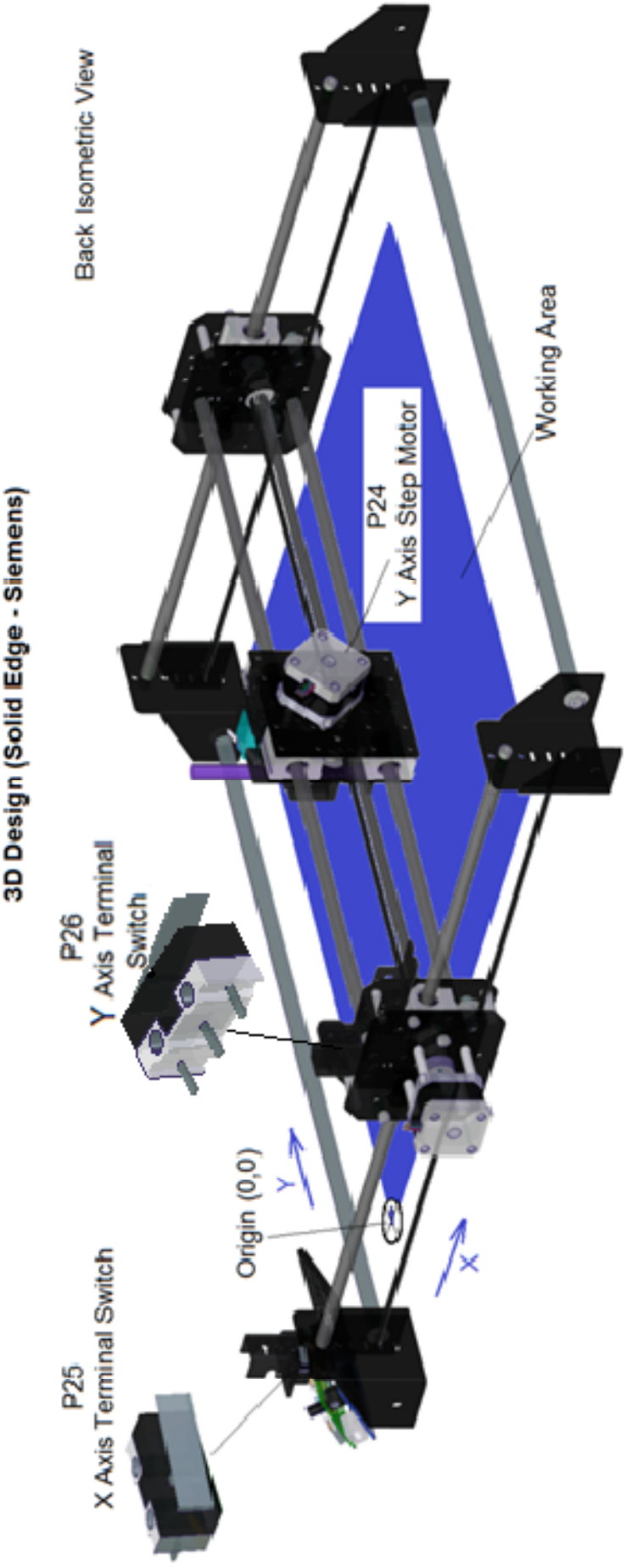


Image 2.2 - 3D Design Back Isometric View

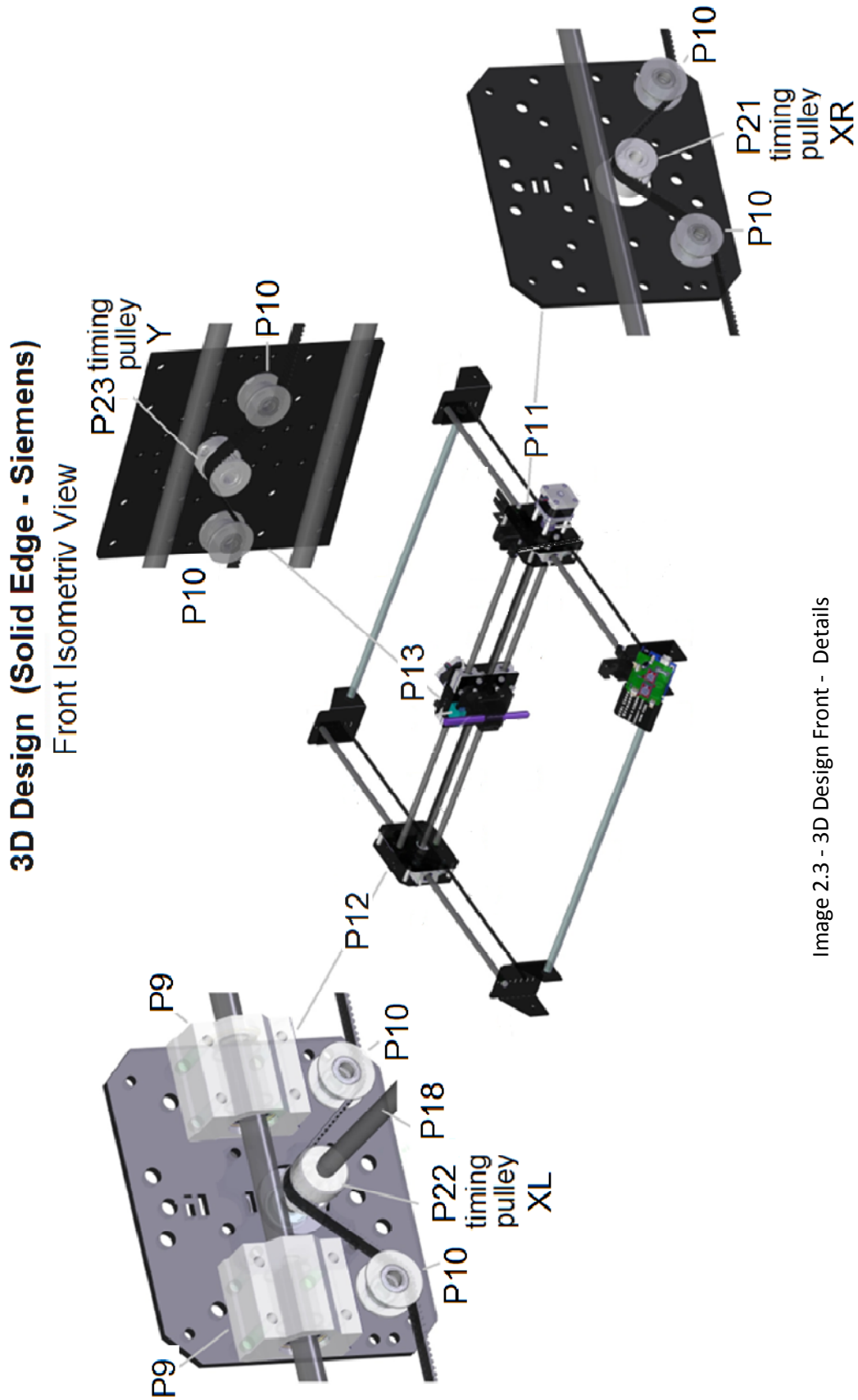


Image 2.3 - 3D Design Front - Details

A square frame assembled using, four frame corners (P(art)1). The corners connected by two Joint Frame Corners (P. 2) in Y axis direction and by two X axis Rods, -R (P.3) and -L (P. 4) in X axis direction.

Through the X axis Rods, runs the motion in X axis. For that purpose, two Carriers (P.11 and P.12) slides upon the X axis Rods via Linear Bearings (P.9) . The -R X axis carrier (P11), carries the X Axis Step motor (P.5). The motor rotates the X axis timing pulley -R (P.21) which is attached on the -X axis belt -R (P14). The belt driven pulleys P10, gives to the belt, a V shape formation in order to achieve greater overlap between the belt and the timing pulley. Through this mechanism, the X carrier -R, forced into linear motion.

The two X axis Carriers (-R and -L) joined together with the - Y Axis Rods (P.6 and P.7). In order to achieve an instant response for both X axis Carriers, a Joint Shaft (P. 18) used. The Joint Shaft attached to X axis motor Shaft from one side using a mechanical coupler (P. 24) while the other side results in a ball bearing in the X axis Carriers -L. The X Axis timing pulley -L (P.22) is attached to the Joint Shaft and with the same V shape mechanism the X carrier -L, forced into linear motion. This mechanism secures a simultaneous motion for the two X axis Carriers.

In Y axis Rods the motion in Y axis take place. The Y axis Carrier (P.13) slides upon the rods via Linear Bearings. The Y axis carrier, carries the Y Axis Step motor (P.24). The motor rotates the Y axis timing pulley (P.23) which is attached on the -Y axis belt (P16). The belt driven pulleys (P.10), gives to the belt the V shape formation in order to achieve greater overlap between belt and the timing pulley. Through this mechanism, the Y carrier, forced into linear motion. The header (laser, Pen, Drill, Nozzle or other), is attached in the Y axis carrier

At start up, a mechanical initialization take place in order to drive the header in the reference point (origin). So the first task the microcontroller execute s is to move the header to the origin (0,0). In the X axis Origin (0,x) the contact of the NO-NC X Axis Terminal Switch (P25) close and gives signal to microcontroller in order to stop the motion in the X axis direction. Similarly in the Y Axis Origin (x,0) the contact of the NO-NC Y Axis Terminal Switch (P26) close and gives a signal to microcontroller in order to stop the motion in the Y axis direction.

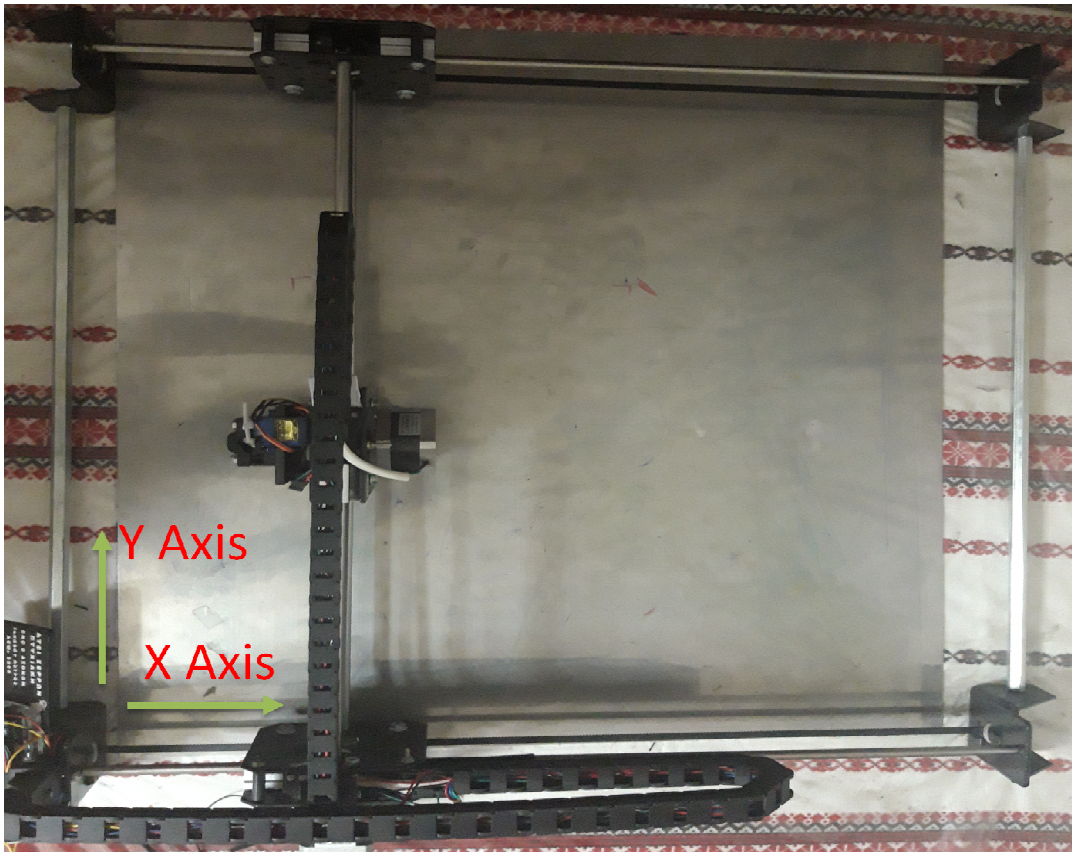


Image 2.4 - Top view (Photo)

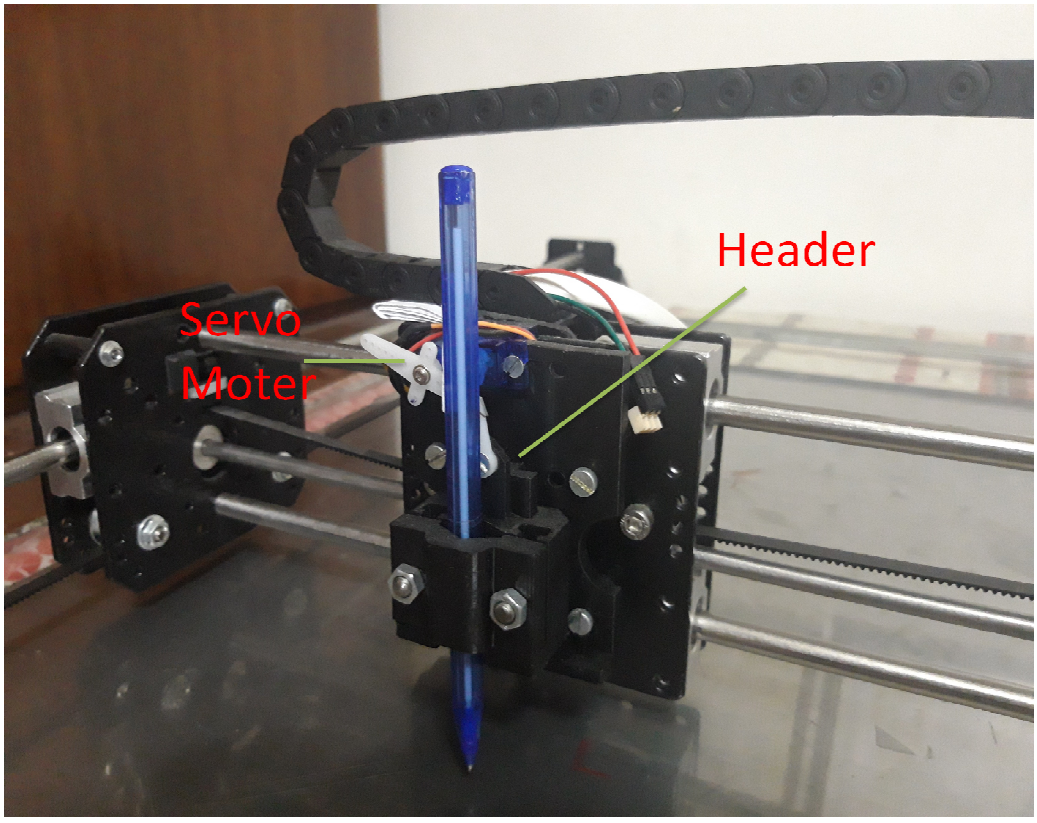


Image 2.5 - Y Axis Carriage and Header (Photo)

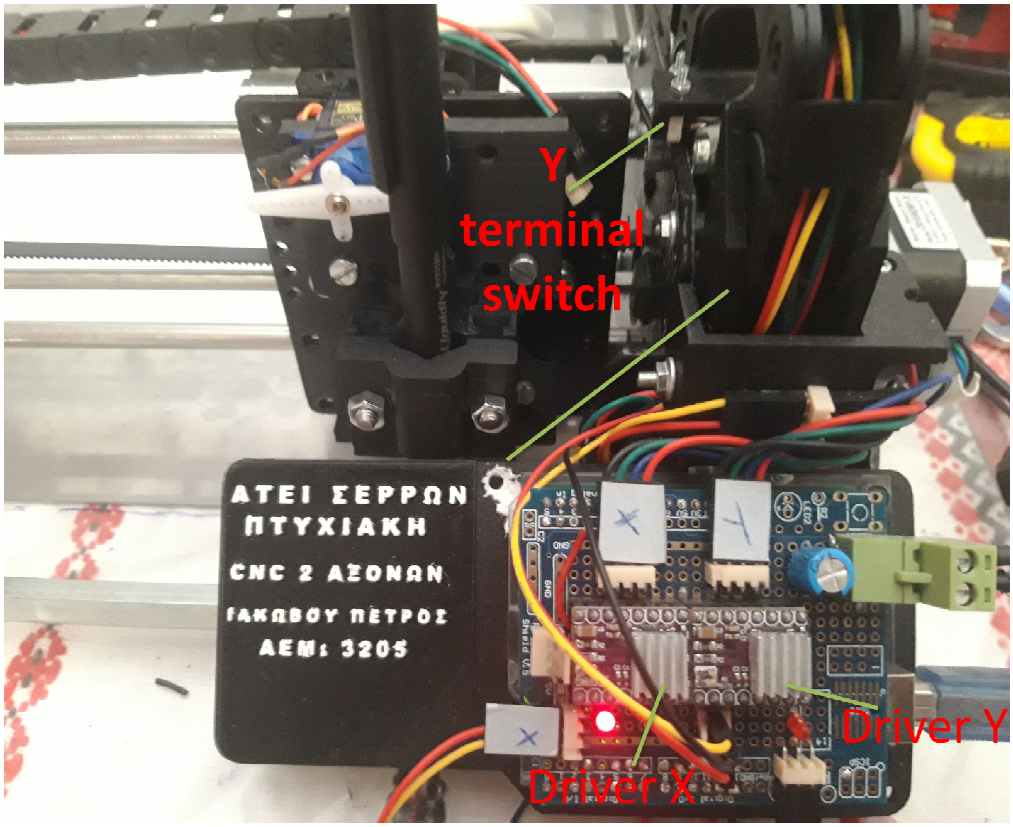


Image 2.6 - PCB (Driver X and Y), Y axis terminal switch (Photo)

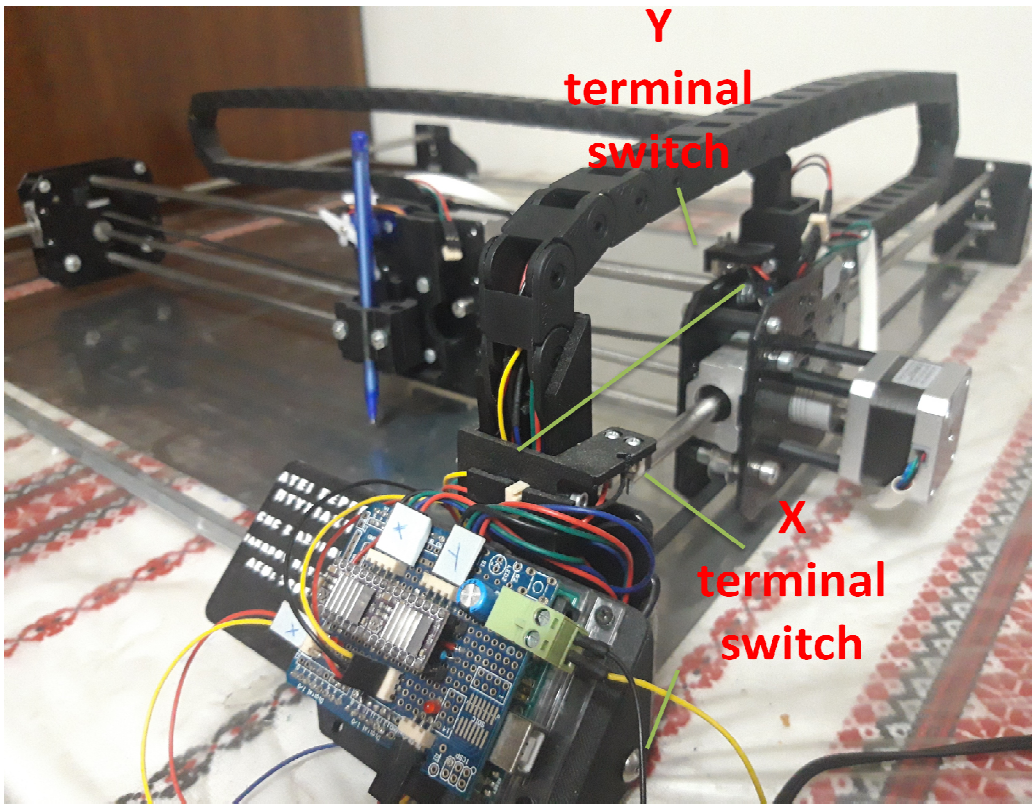
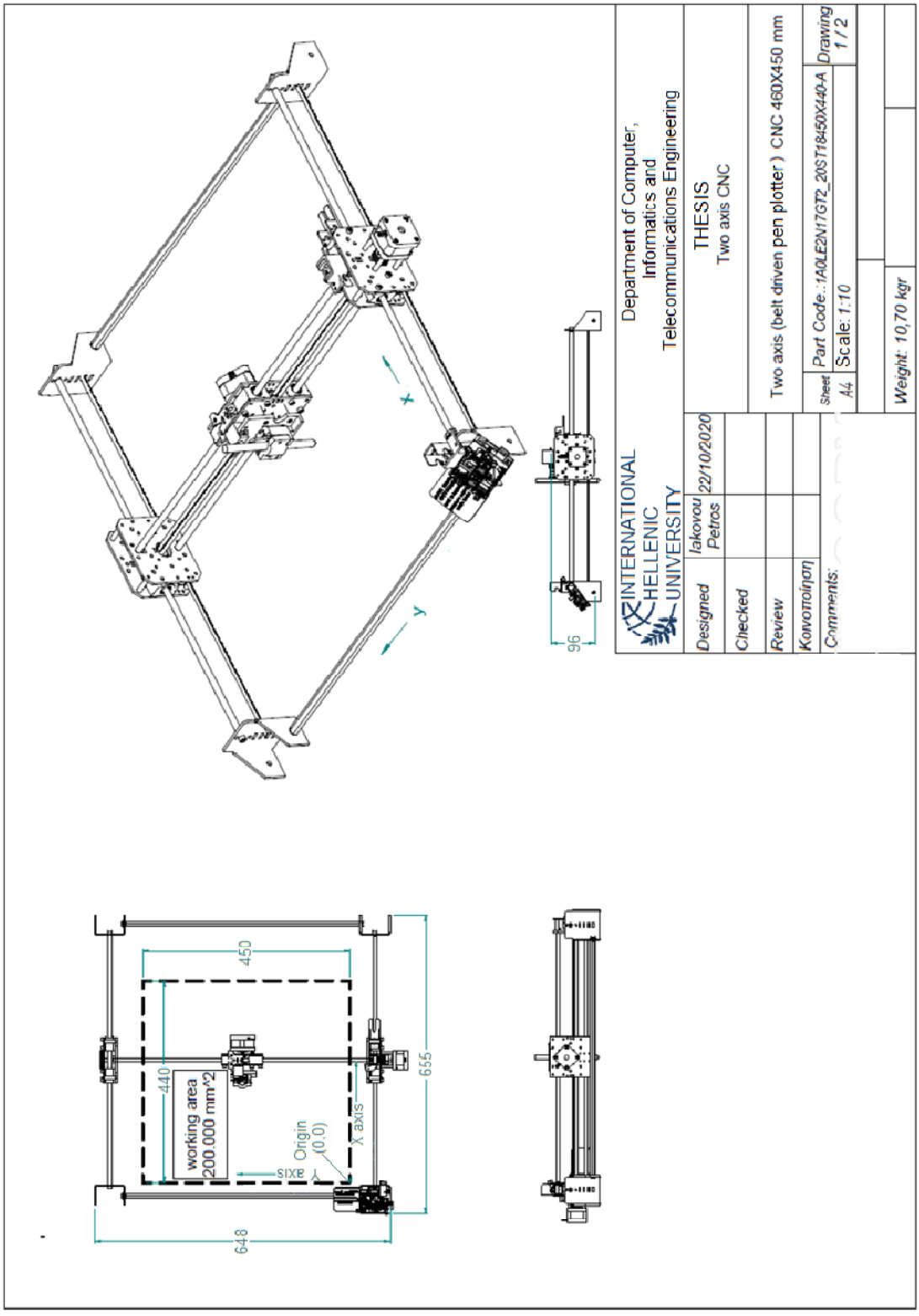


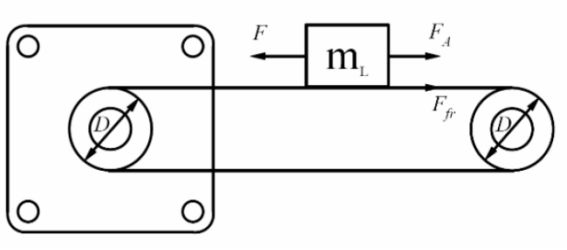
Image 2.7 - X and Y axis terminal switch, X axis Carriage (Photo)



Draft 2.3 - Mechanical Design of CNC

2.1.4 Torque Calculation for belt driven motion:

Torque calculation requires the knowledge of the total weight that has to be moved, the acceleration and deceleration ,the max velocity, and the inertia of the mass involved in the motion.



Motion in X axis demands greater torque than in Y axis and this, because motor in X axis has to carry the assembly of both X axis carriers (-R and -L) plus the assembly of Y axis carrier while motor in Y axis has to carry only the assembly of Y axis carrier.

Calculation of required torque for the X axis motor:

The total torque is the sum of the required torque Ta for acceleration until the desired velocity and the required torque T_L for moving the weight with constant velocity

$$T_T = T_L + T_a \ (2.2)$$

For Ta , the total system's inertia has to be calculated as the sum of the inertias of the components involved in the motion.

the Weight inertia:

$$J_L = \frac{1}{4} m_L D^2 \ (2.3)$$

m_L: Is the total weight (kg) of masses that the X axis motor has to move (assembly of the two X axis carriers, assembly of the Y axis carrier, the two Y axis rods, the Joint Shaft and the Y axis belt). From the 3D model this mass is m_L=3,8 (kg)

$$J_L = \frac{1}{4} \cdot 3{,}8 \cdot (0{,}01222)^2 = 1{,}4 \cdot 10^{-4} kg \cdot m^2$$

Timing pulley inertia:

$$J_P = \frac{1}{4} m_P D^2 \ (2.4)$$

m_P: pulley's weight (kg)

D: pulley's diameter (m)

$$J_P = \frac{1}{8} m_P D^2 = \frac{1}{8} \cdot 0{,}0054 \cdot (0{,}01222)^2 = 10^{-7}\ kg \cdot m^2$$

By having two pulley's involved in x axis motion (P.21 and P.22), the above amount has to be calculated twice in the total sum.

Belt inertia:

$$J_B = \frac{1}{4} m_B D^2 \ (2.5)$$

m_B: belt's weight (kg)

D: pulley's diameter (m)

$$J_B = \frac{1}{8} \cdot 0{,}01 \cdot (0{,}01222)^2 = 1{,}9 \cdot 10^{-7}\ kgm^2 (B.3)$$

Total System's inertia:

$$J_T = J_L + 2 \cdot J_P + J_B + J_M (2.6)$$

J_M: motor's inertia (kgm^2) (taken from motor's datasheet)

$$J_T = 1{,}4 \cdot 10^{-4} + 2 \cdot 10^{-7} + 1{,}9 \cdot 10^{-7} + 1{,}5 \cdot 10^{-6} = 1{,}42 \cdot 10^{-4}\ kgm^2$$

Normally the inertia of the four idler pulleys (P.10) has to be added, but (in the current application) it is obvious that the total inertia determined mainly from the total weight of the moving mass and the contribution from the other systems is negligible.

Required torque for acceleration:

$$T_a = J_T a = J_T \frac{\omega_1 - \omega_0}{\tau} \ (2.7)$$

$$T_\alpha = J_T \alpha = J_T \frac{\omega_1 - \omega_0}{t} (B.4)$$

ω_0: initial velocity (rad/sec)

ω_1: final velocity (rad/sec)

An initial approach is for max velocity 80 mm/sec (=13 rad/sec for D=0,01222 m) and the time to take this value t=1 sec.

$$T_\alpha = 1{,}42 \cdot 10^{-4} \frac{13 - 0}{1} = 18{,}5 \cdot 10^{-4}\ Nm$$

Required torque for moving the weight with constant velocity:

$$T_L = \frac{m_L\, g D_\mu}{2n} \quad (2.8)$$

g: gravity acceleration (9.8 m/s^2)

μ: friction coefficient

n: performance coefficient (0,85-0.95)

$$T_L = \frac{3{,}8 \cdot 9.8 \cdot 0{,}01222 \cdot 0{,}5}{2 \cdot 0.90} = 0{,}13Nm$$

Total required torque:

$$T_T = T_L + T_a \quad (2.9)$$

$$T_T = 0{,}13 + 18{,}5 \cdot 10^{-4} = 0{,}1318\ Nm$$

In the current application the total required torque depends from the required torque to move the total weight with constant velocity T_L. The X axis torque calculated from the above amount incremented with a safety coefficient K_s=1,5

X Axis Motor torque:

$$T_M = K_S + T_T \quad (2.10)$$

$$T_M = 1{,}5 \cdot 0{,}1318 = 0{,}20Nm$$

The same motor will be used for the motion in the Y axis, although it is not an economic decision.

2.2 The electronic hardware:

2.2.1 Driver DRV8825 [4]

The electronic circuit that control the step motor is the driver DRV8825. It uses two H bridges to alter the current direction in the motor's windings (one for each winding). The driver has 8 contacts (pins).

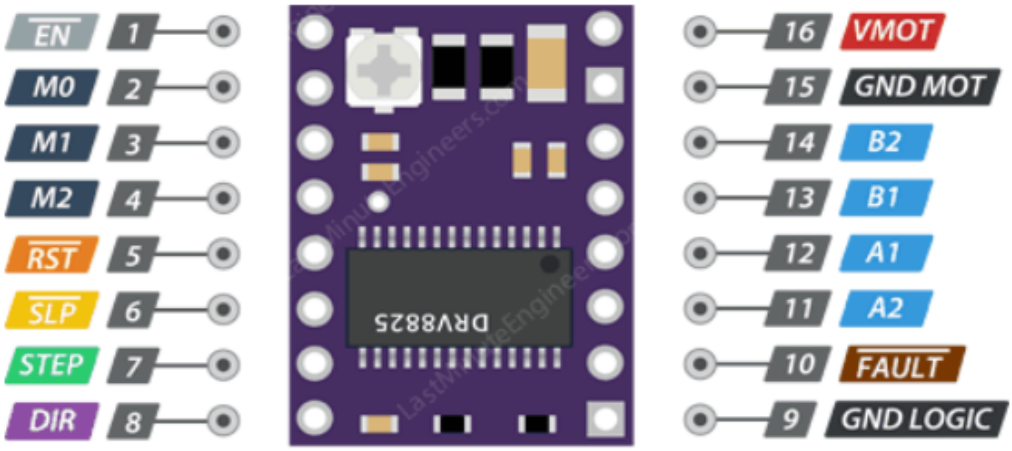


Image 2.8 - The DRV8825 Driver

1(**En)**able : This pin is the driver's on-off switch. By Clear this bit (logic "0": Enable) the driver 's outputs (A & B) act in motor's wings, while by Set this bit (logic "1"), the driver disabled.

2 (**M0**) Mode0, 3 (**M1**) Mode1, 4 (**M2**) Mode2: The value in these pins determines the output current distribution in motors wings according to the table below. By this way it is possible to subdivide the basic motor's step angle.

MODE2(M2)	MODE1(M1)	MODE0(M0)	STEP MODE
0	0	0	full step (two-phase)
0	0	1	1/2 step (1-2 phase)
0	1	0	1/4 step
0	1	1	8 microstep/step
1	0	0	16 microstep/step
1	0	1	32 microstep/step
1	1	0	32 microstep/step
1	1	1	32 microstep/step

Table 2.3 - Driver's Operation Modes

From the motor's technical specification a full step is for an angle a= 1,8^{o} which gives (from equation 2.1) a linear movement:

$$dx = \frac{2 \times 1{,}8}{18} = 0{,}20 \ mm$$

By Manipulating the M0, M1 and M3 bits it is possible to subdivide the basic movement of a full step according to the table above.

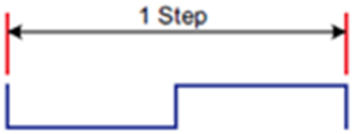
The maximum resolution is:

$$dx/32 = 1/32 * 0,2 = 0,00625 \text{ mm resolution.}$$

Although increasing the resolution presumes higher accuracy, it has negative impact in torque's capability.

Furthermore, the smaller the movement 'dx' a microstep can give, the greater the number of microsteps that needed to accomplish a total movement 'Δx'. And in case of poor hardware capabilities (for example 8 bit registers) this increment could lead to greater software's complexity (§ 6.1).

7(STEP) : Producing a pulse in this pin, rotates motor's shaft one microstep.



8(DIR) : the value in this pins determines the motor's rotation direction (depends from the motor's wiring connection to A & B driver's pins).

11(A2),12(A1),13(B1),14(B2): Output to motor's wings

15(GND MOT): Motor's ground

16(VMOT): Motor's voltage

2.2.2 AVR 328P microcontroller [6]

The Atmega 328P (Atmel) microcontroller is the brain of the CNC. It takes data that arise from G code and signals from terminal switches and produce the appropriate signals to drivers and other peripherals devices (for example servo motor in case of pen plotter).

It uses Harvard architecture (separate bus for data and program). The command set (131 commands in total) is based on RISC architecture (one command execution for every clock cycle). It has 32, 8bit, general purpose registers which are direct accessible from the ALU in one clock cycle. It has two 8 bit timer-counters, one 16bit timer-counter and among many other functions can support USART communication.

The figure below illustrates the microcontroller by naming its pins.

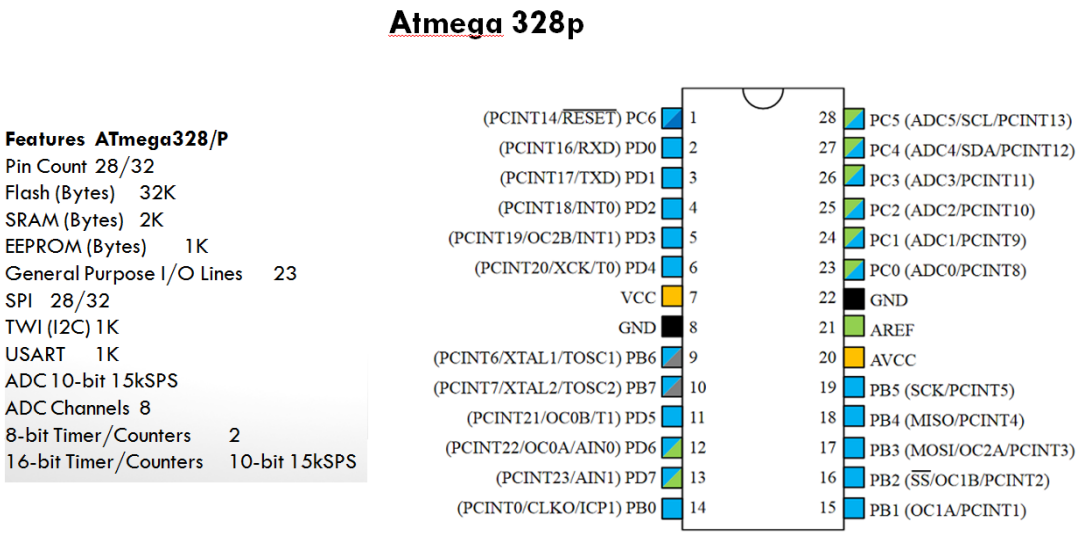


Image 2.9 - The Atmega 328p Microcontroller

To control the motor, the function CTC (**C**lear **T**imer on **C**ompare) of timer 0 and timer 2 will be used.

By setting up the timers in CTC (Clear Timer on Compare) mode, it is possible to alter the value of a specific pin when the timer's value match a pre-defined value in a dedicated for that purpose register. This pin for timer0 is pin12 (PD6) while for timer2 is pin 17 (PB3).

timer0: (pin12 ->PD6:OC0A) for x Axis motor and

timer2: (pin17 ->PB3:OC2B) for y Axis motor.

In the wiring the pin PD6 connected to pin STEP of driver X and pin PB3 to pin STEP of driver Y.

- Pin 328P Mapping

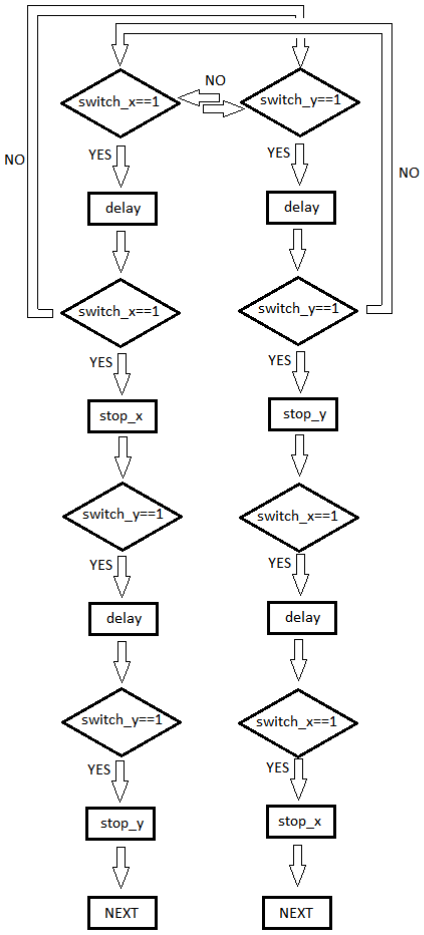
PD2 (Input): This pin is connected in the 'NC' (**N**ormal **C**lose) contact of the X AXIS terminal switch (X_MSW). The 'C' (**C**ommon) switch's contact is continuously at 5V. The switch (image 2.3) is placed near the origin with such way so it's activation button to be upon the X axis. At start up (before the microcontroller starts to receive data from the G-Code), the CNC, drives the header to the origin. When the header reach the origin, the X axis carriage push the activation button and so the logic "1" appears in the PD2 pin. By reading the microcontroller logic "1" in PD2 pin, disables the X axis driver (PB2→"1").

PD3 (Input): Used to disable Y axis driver when header reach origin, at start-up, with similar way as PD2 does. The Y axis terminal switch Y_MSW (figure 2.3) is placed on X axis carriage -R and activated when the Y axis carriage push its activation

button. At this time PD3 pin comes to logic "1" and the microcontroller disable Y axis driver by making (PB4→"1").

PD2 and PD3 pin are assigned as input.

The below code (assembly) in Code 2.1 audits the activation of X and Y terminal switches. The inspection run pseudo-parallel for the two switches. When one of the them activated, the code recheck the value after a small delay. If the activation is confirmed the code will jump in a loop waiting the activation of the other switch (including a second confirmation again , after a small delay). The roll of the seconds confirmations is to skip error signals due to scrambles (a kind of software filter).




```
ORIGIN:
X_AXIS_ORIGIN:
                SBIS    PIND,2            //Skip if bit PD2 (switch_x) is set
                RJMP    Y_AXIS_ORIGIN
                RCALL   Delay
                SBIS    PIND,2            //Skip if bit PD2 (switch_x) is set
                RJMP    Y_AXIS_ORIGIN
                SBI     PORTB,PB2        //Stop x_axis motor (PB2=1)
        Y0:     SBIS    PIND,3          //Skip if bit PD3 (switch_y) is set
                RJMP    Y0
                RCALL   Delay
                SBIS    PIND,3         //Skip if bit PD3 (switch_y) is set
                RJMP    Y0
                SBI     PORTB,PB4      //Stop y_axis motor (PB4=1)

                RCALL   Delay 1s

                RJMP    Initialize_USART

Y_AXIS_ORIGIN:
                SBIS    PIND,3            //Skip if bit PD3 (switch_y) is set
                RJMP    X_AXIS_ORIGIN
                RCALL   Delay
                SBIS    PIND,3           //Skip if bit PD3 (switch_y) is set
                RJMP    X_AXIS_ORIGIN
                SBI     PORTB,PB4        //Stop y_axis motor (PB4=1)
        X0:     SBIS    PIND,2           //Skip if bit PD2 (switch_x) is set
                RJMP    X0
                RCALL   Delay
                SBIS    PIND,2           //Skip if bit PD2 (switch_x) is set
                RJMP    X0
                SBI     PORTB,PB2        //Stop x_axis motor (PB2=1)

                RCALL   Delay_1s

                RJMP    Initialize_USART
```

Code 2.1 - Go to Origin Inspection (for Mechanical initialization)

PD6 (Output): It is connected with the pin STEP_X of driver X. Generating one pulse in this pin cause the X axis motor to rotate one microstep. The number of microsteps is the total movement (dx), while the frequency of pulse is the velocity. To produce the pulse the function CTC (**C**lear **T**imer on **C**ompare) of timer/counter 0 is used.

By setting up the timer/counter 0 in CTC mode and the PD6 altered, when the value in the counter identified with the value in OC0A register. After the identification the

counter nullify. From microcontroller's datasheet the frequency f_{OC0A} in which the compare match take place is given by the following equation:

$$f_{OC0A} = \frac{f_{clk_I/O}}{2 * N * (1 + OC0A)} \quad (2.11)$$

where:

$f_{clk\_I/O}$=16.000.000 Hz is the input-output clock frequency.

OC0A is the value of OC0A register in decimal system and

N is the timer/counter prescaler which determines after how many clock pulses the counter increase or decrease its value. It can take the following values N=(1, 8 , 64, 256, 1024)

The type of motion is linear with stable velocity and so the equation between velocity $\vec{v}$ and movement $\overrightarrow{dx}$ is

$$dx = v * dt \Leftrightarrow t = \frac{dx}{v} \Leftrightarrow \frac{1}{t} = \frac{v}{dx} \rightarrow f_{OC0A} = \frac{v}{dx} \quad (2.12)$$

From the above equations, the value of OC0A register for a given displacement and velocity is:

$$OC0A = \frac{f_{clk_I/O}}{2 * N * (\frac{v}{dx})} - 1 \quad (2.13)$$

The equation says that the velocity is inverse proportional with the value in OC0A register. Furthermore it is also inverse proportional with the prescaler N. Greater resolution can achieved with small values in prescaler N. But small values in N gives large values in OC0A register and with 8 bit register this is a problem.

PB0 (Output): It is connected to pin DIR_X of driver X. The value in this pin determines the direction of motor's shaft rotation. The directions depends on motor's wiring in the A and B driver's output. Here, when it is in logic "0" the motor's shaft rotated clockwise (CW) while in logic '1' rotated counter clockwise CCW. By the current design CW motion drives the header in higher values to X axis while CCW motion to lower values.

PB1(Output): This pin is connected to pin SIGNAL of a Servo Motor (in case of a Pen plotter a Servo motor used to control the pen's position). The type of Servo motor in this application **[5]** can rotate between three fixed positions (-90^o, 0^o, +90^o). The position determined by a control circuit which is attached to the servo motor. The control circuit act by abstract the income pulse which takes from the SIGNAL pin with a three standard pulses which designate the tree positions (given in servo motors data sheet).

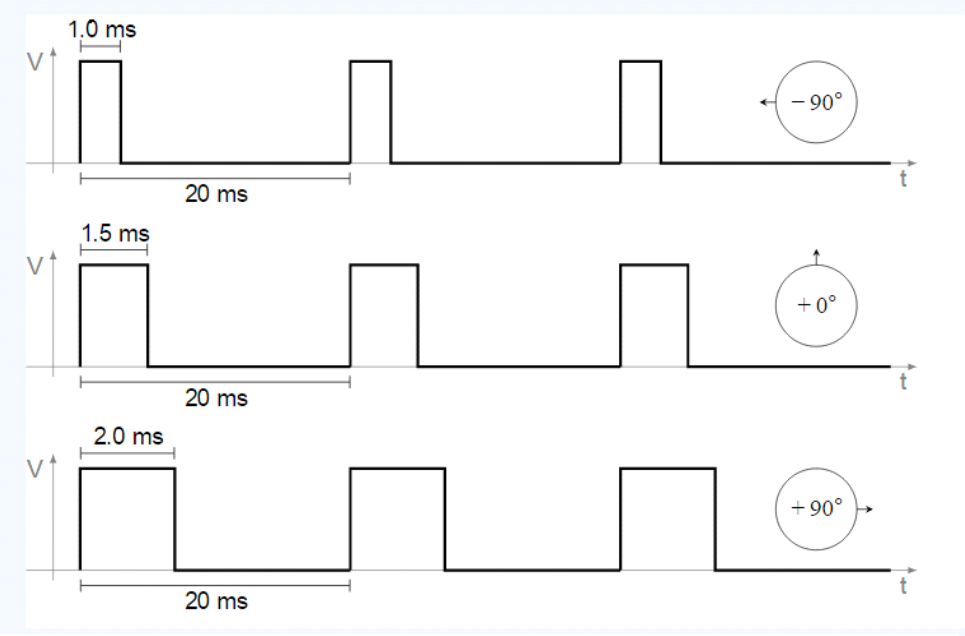


Image 2.10 - Servo Motor Pulse - Position relationship

To generate a PWM (Pulse Width Modulation) with specific frequency, as output in PB1 pin, the timer/counter 1 will be used. For that purpose it needs to set up the timer / counter 1 to Fast PWM mode (mode 14). In this mode the pin OC1A (PB1) comes to '1' when the counter is at bottom and change to '0' when a compare match occurs between timer / counter 1 and OCR1A register. The counter increase until a defined TOP value and then returns to BOTTOM.

The pulse frequency determined by the TOP value and should be f= 50 Hz (20 ms) (datasheet). And with the timer/counter prescaler N=8 this value should be

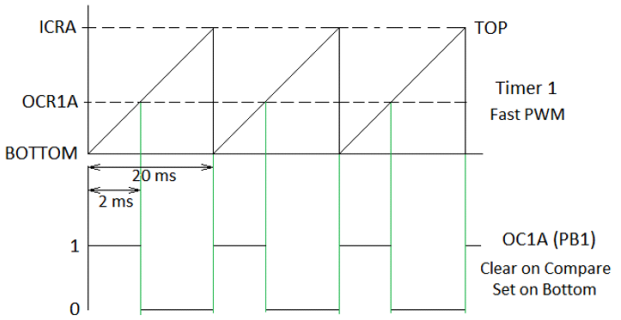
$$f = \frac{f_{clk_I/O}}{2 * N * (1 + ICR1A)} \rightarrow ICR1A = \frac{16{,}000{,}000\ Hz}{2 * 8 * 50\ Hz} - 1 \rightarrow ICR1A = 19{,}999$$

The value in the compare register $OCR1A_{(0)}$ to generate the signal (duty cycle) 1,5 ms (f(0)=666,666 Hz → position 0^o) is

$$OCR1A_{(0)} = \frac{16{,}000{,}000\ Hz}{2 * 8 * 666{,}666\ Hz} - 1 \rightarrow OCR1A_{(0)} = 1{,}499$$

The value in the compare register $OCR1A_{(0)}$ to generate the signal (duty cycle) 2 ms (f(0)=500 Hz → position 90^o) is

$$OCR1A_{(90)} = \frac{16{,}000{,}000\ Hz}{2 * 8 * 500\ Hz} - 1 \rightarrow OCR1A_{(90)} = 2{,}000$$




```
.equ    Pen_Down            =   1500
.equ    Pen_UP              =   2000
;====================================

//SET UP TIMER 1 . PEN_Header
LDI     TEMP,0b01_0000_00     //toggle OC1A DP->9 PB1 ON COMPARE MATCH
STS     TCCR1A,TEMP
LDI     TEMP,0b000_01_010   // prescaller 8
STS     TCCR1B,TEMP
;=========================================Pen_Up=2000==============
//By giving the value 2000 to OCR1AH a 2 ms pulse produce in PB1 pin
//and the servo moter rotates to +90 degrees position
Pen_U:
                LDI XH,HIGH(Pen_Up)
                STS OCR1AH,XH
                LDI XL,LOW(Pen_Up)
                STS OCR1AL,XL

                RET
;=========================================Pen_Down=1500================
//By giving the value 1500 to OCR1AH a 1.5 ms pulse produce in PB1 pin
// and the servo moter rotates to +00 degrees position
Pen_D:
                LDI XH,HIGH(Pen_Down)
                STS OCR1AH,XH
                LDI XL,LOW(Pen_Down)
                STS OCR1AL,XL

                RET
;=================================================================
```

Code 2.2 - Servo Motor Calibration

PB2(Output): It is connected with pin EN_X of driver X, for enable/disable the driver. When it is in logic "1" the driver is disabled, while in logic "0" is enabled.

PB3(Output): It is connected with the pin STEP_Y of driver Y. Generating one pulse in this pin cause the Y axis motor to rotate one microstep. The number of microsteps is the total movement (dy), while the frequency of pulse is the velocity. To produce the pulse the function CTC (**C**lear **T**imer on **C**ompare) of timer/counter 2 is used. The operation principle is the same as timer/counter 0 acts in PD6 pin.

PB4(Output): It Connected with pin EN_Y of driver Y, for enable/disable the driver. When it is in logic "1" the driver is disabled, while in logic "0" is enabled.

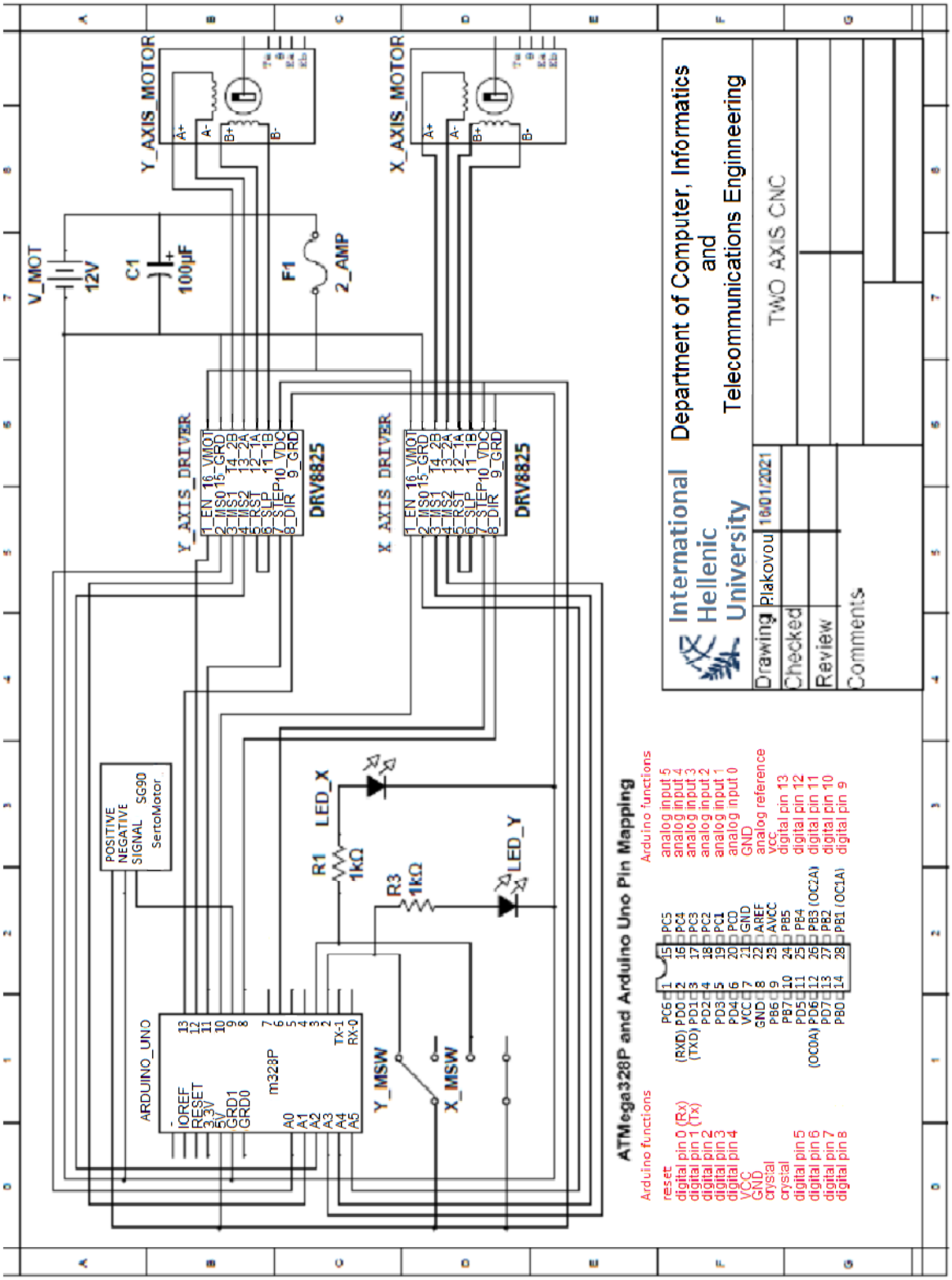
PB5(Output): It is connected with pin DIR_Y of driver Y. The value in this pin determines the direction of the shaft rotation. The direction depends on motor's wiring in the A and B driver's output. Here when it is in logic "0" the motor's shaft is rotated clockwise (CW) while in logic '1' it is rotated counter clockwise CCW. By the current design CW motion drives the header in higher values to Y axis while CCW motion to lower values.

PC0 / PC1 / PC2(Outputs): These pins are connected in correspondence with the M0, M1, M2 pins of driver Y and determine the operation mode of driver according to table 2.3.

PC3 / PC4 / PC5(Outputs): These pins are connected in correspondence with the M2, M1, M0 pins of driver Y and determine the operation mode of driver according to table 2.3

2.2.3 Electronic Design

The electronic design below illustrates the wiring connection between the electrical components. The microcontroller is mounted in the arduino development board. In the down-left corner there is a pin mapping between the m328p and arduino's outputs.



Draft 2.4 - Electrical Design

The microcontroller has tree 8-bit ports A,B and C and gives access to the 8 - bits of each port with a single command. The wiring connection between microcontroller and drivers has anticipated the potential to determine both driver's operation step mode (driver prescaler - table 2.3) with a single command by connecting the pins M0, M1, M2 of both drivers to port C.

Similarly one byte in PORT B controls the rotation direction, the enable / disable state of both motors through drivers and the state (enable / disable) of header. The port B named "**START Byte**".

	PB7(IN)	**PB6(IN)**	**PB5**	**PB4**	**PB3**	**PB2**	**PB1**	**PB0**
PORTB	'1'	'1'	Y DIR	Y EN	Y STP	X EN	HEADER	X DIR

	PC7(IN)	**PC6(IN)**	**PC5**	**PC4**	**PC3**	**PC2**	**PC1**	**PC0**
PORTC	'1'	'1'	Y MS1	Y MS2	Y MS3	X MS3	X MS2	X MS1

	PD7(IN)	**PD6**	**PD5(IN)**	**PD4(IN)**	**PD3**	**PD2**	**PD1**	**PD0**
PORTD	'1'	X STP	'1'	'1'	Y SW	X SW	TXD	RXD

Example:

- if dx<0 , through Port B, the pin ENABLE of driver x becomes '0' to enable the driver and the pin X DIR becomes '1' for CCW rotation

	PB7(IN)	**PB6(IN)**	**PB5**	**PB4**	**PB3**	**PB2**	**PB1**	**PB0**
PORTB	'1'	'1'	Y DIR	Y EN	Y STP	'0'	HEADER	'1'

- if dx>0 , through Port B, the pin ENABLE of driver x becomes '0' for enable the driver and the pin X DIR becomes '0' for CW rotation

	PB7(IN)	**PB6(IN)**	**PB5**	**PB4**	**PB3**	**PB2**	**PB1**	**PB0**
PORTB	'1'	'1'	Y DIR	Y EN	Y STP	'0'	HEADER	'0'

-and if dx=0 , through Port B, the pin ENABLE of driver x becomes '1' to disable the driver and stop rotation

	PB7(IN)	**PB6(IN)**	**PB5**	**PB4**	**PB3**	**PB2**	**PB1**	**PB0**
PORTB	'1'	'1'	Y DIR	Y EN	Y STP	'1'	HEADER	X DIR

For every command in G code, movement and velocity, are calculated for each axis and the data is stored in some defined working registers. Finally the START byte is sent to Port B making the motor to act accordingly.

Example of START Byte (PORT B) for dx>0 & dy=0

	PB7(IN)	**PB6(IN)**	**PB5**	**PB4**	**PB3**	**PB2**	**PB1**	**PB0**
PORTB	'1'	'1'	-	'1'	Y STP	'0'	HEADER	'1'

Bit manipulation

In many cases the demand comes up to give a value to a single bit in START byte, leaving the rest bits unchanged. As it not possible to know the initial value of START Byte each time, a method is required to give a value to a single bit without affect the others.

- Set a single bit '1' in a binary word **(logical OR):**

The logic OR is a simple way to secure that a particular bit in a binary word has taken the value '1' leaving the others unchangeable. For example in the byte XXXXXXXX (the value X could be either '0' or '1') , the 0-bit must be '1'. This is achieved by perform the logic OR between the byte and the mask '00000001'

XXXXXXXX || 00000001 = XXXXXXX1

- Clear a single bit '0' in a binary word **(logical AND):**

The logic AND is a simple way to secure that a particular bit in a binary word has taken the value '0' leaving the others unchangeable. For example in the byte XXXXXXXX , the 4-bit must be '0'. This is achieved by perform the logic AND between the byte and the mask '11101111'

XXXXXXXX&&11101111 = XXXX0XXXX

With this method it is possible to ensure the value, for more than a single bit, in a byte by using the appropriate mask. For example in byte XXXXXXXX the 0 and 3 bit to be '1' and the 6 and 7 bit to be '0'

XXXXXXXX || 00001001 = XXXX1XX1

XXXX1XX1 && 00111111 = 00XX1XX1

2.2.4 Printed Circuit Board (PCB)

The m328p microcontroller is attached to the arduino UNO development board. An add-on shield has been developed (image 2.11 front) in order to wire the peripherals with the microcontroller. The shield carries on the:

1. the X and Y motor drivers

2. One 2 pin connector for the 12 V supply (from a transformer) to the motors

3. Two 4 pin connectors for output to the motors winding

4. Two 2 pin connectors for input from the terminal switches

5. Two red-color leds (one for each axis) as display when the terminal switches are activated (header to the origin)

The shield attached to the arduino UNO board and the components, wired between each others, using flexible cables image 2.11 Back photo. The right photo is a semi finished attempt to develop the shield in the Eagle Software.

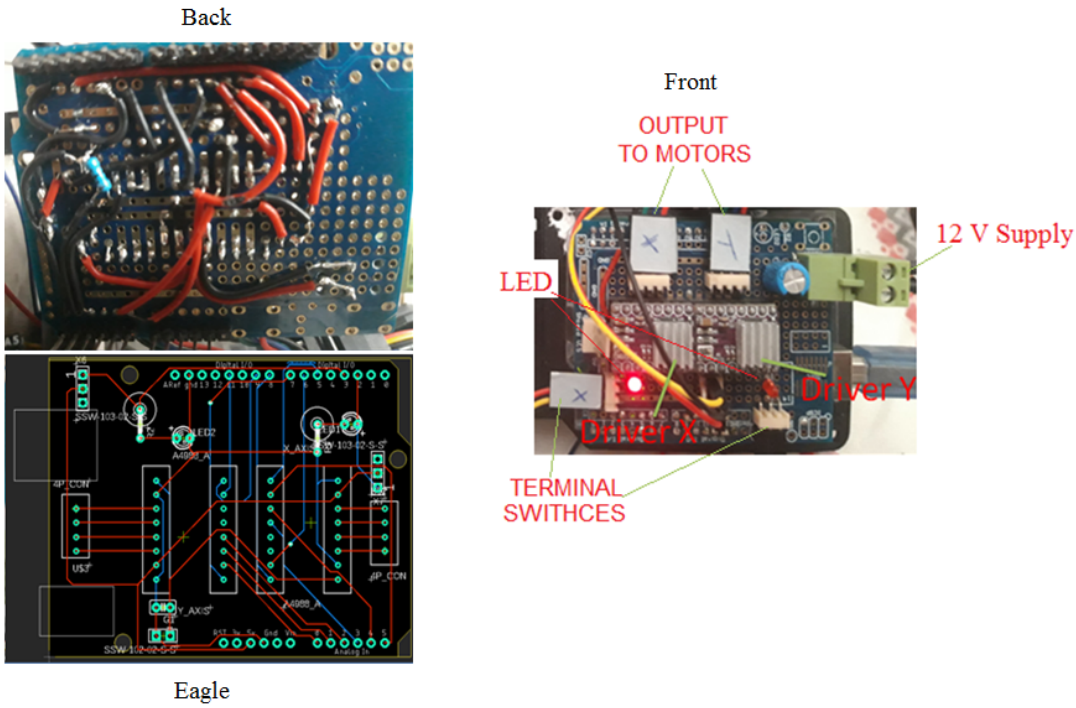


Image 2.11 - Add on Motor Driver's Shield

Part 3: Graphical Interface Environment

3.1 Connection between the CAD geometry and physical working area

One issue is the need of a graphical interface for the interconnection between the 2D design and the geometry of the physical working space.

To create such an interface, the Top View of the CNC (with the header positioned to the origin) is placed (in 1:1 scale) in the background of a 2D design layer, so that the header to be coincident with the layer's origin (figure 3.1). In the same layer, a virtual 2D orthogonal area has been drawn with the X and Y axis calibrated to simulate the geometry of the real working area with regard to the reference point (origin (0,0)).

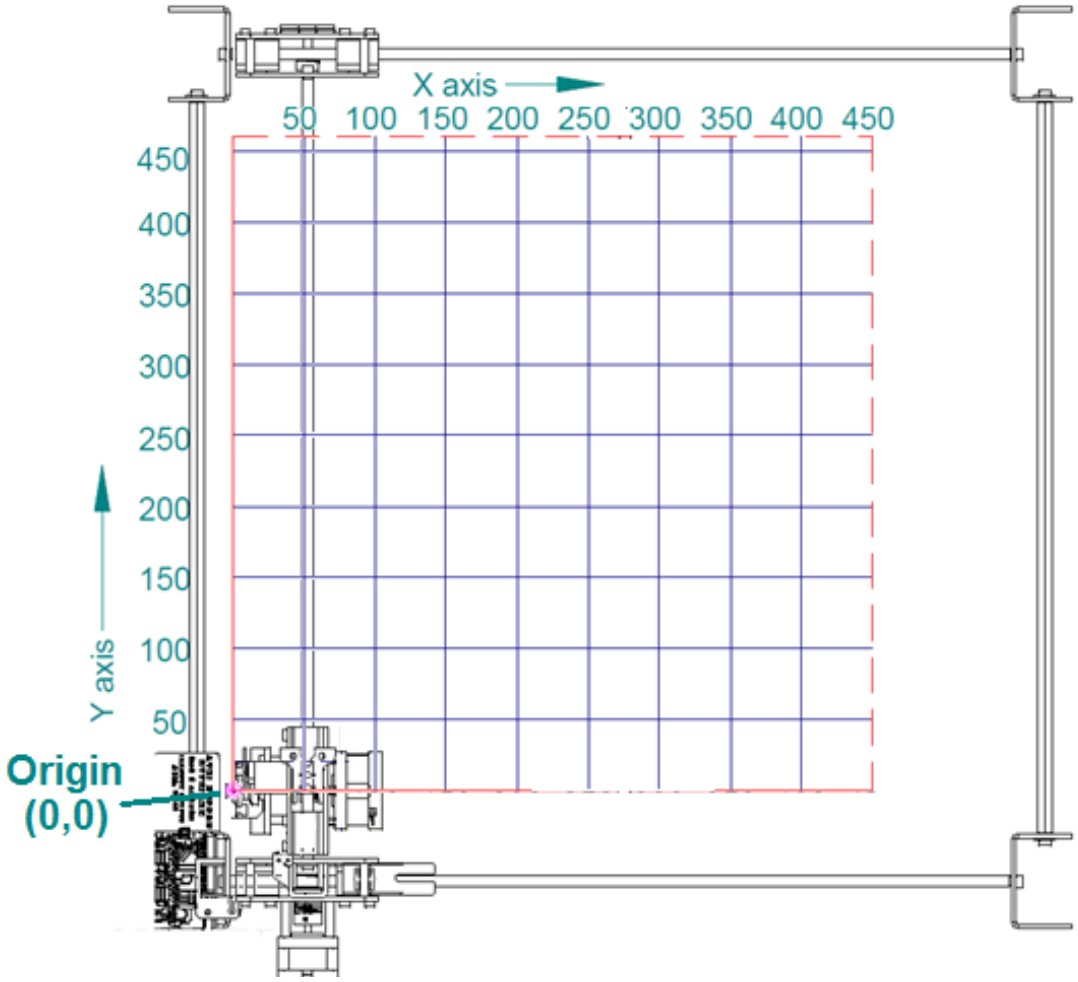


Image 3.1 - Graphical Interface Environment

By using the above interface it is possible to place the 2D design in a specific position in the design working area and relate it with the physical working area. It is possible to place an existing 2D drawing and work on it (if this is necessary) or to generate a 2D drawing from scratch. By having the drawing in the final form and position, it is saved (after the user disappears the background) in .dxf format and it is ready for reading, from the application program (CAD2G).

Part 4: Communication

4.1 USART (Universal Synchronous Asynchronous Receiver Transmitter)

At start up the CNC executes a mechanical initialization, driving the header to the origin(0,0) which is the reference point. In the origin the microcontroller waits to receive data in order to start executing the G code commands one by one.

In parallel the executable CAM program reads the .dxf file and generates a txt file (.nc) with the G code which describes the geometry of the 2D drawing. After accomplishing this task, the program opens the .nc file and reads the G code commands one by one. For every G or M command the program generates raw data in binary format. This data has to be sent to the microcontroller in order to control peripheral devices (motors) .

Note:

The communication to transfer the data is time consuming and the time it needs for the command execution affects the machines accuracy (the most critical parameter in a CNC machine). It is faster for the microcontroller to read directly the G code (from a sd card for example) and then to execute the calculations to generate the data.

In the current application the data, is arisen from the G code, by making some mathematical and trigonometrical calculations. In order to execute these calculations in assembly language the microcontroller must have dedicated floating point unit. The m328p luck of such unit and for that reason the calculations are executed in application program and data is sent to m328p.

Programming the microcontroller in C wiring gives the potential to execute mathematical and trigonometrical calculations even without a special unit (the compiler undertakes to accomplish this task).

To transfer the data from the executable program to the microcontroller a communication between them, has to be established. For that purpose the UART (**U**niversal **A**synchronous **R**eceiver - **T**ransmitter) communication will be used.

To run the USART communication successfully some basic initialization has to be done for both the microcontroller and the application program. The initialization mainly concerns the baud rate, the number of bits will be transferred, the parity bit and the stop bit. These parameters should be the same for both sides to run successfully the communication.

The initialization and the exchange data functions in the application program take place through the windows API.

After the USART communication successfully established in both sides the data can be sent from the executable program to the microcontroller. In order to have even circulation in data flow it needs a confirmation message (for successfully data reception) from both sides before new transfer takes place.

The executable program:

1. reads a G code command from the .nc file,

2. makes calculations to generate the raw data in binary format,

3. sends the data to microcontroller,

4. waits confirmation from microcontroller for receiving the data,

5. returns to step 1.

The microcontroller:

1. waits for the data,

2. receives the data and puts them to defined working registers,

3. executes (according the G code routing command) the appropriate service routine,

4. send confirmation for receiving the data,

5. return to step 1.

The last command in G code is the M30 which indicates the end of the program. By reading the application program M30, terminated, while the microcontroller disable the header and the drivers and gets into sleep mode.

USART communication is explained thoroughly in the respective capital in main thesis.

Part 5: Linear and Circular Interpolation Algorithms

The control procedure in a CNC machine, concerns, the motion in each axis upon coordinates, so that the header follows specific path in relation to the working part. The concept is to read the coordinates from the G code and the implementation, of an appropriate, code's interpretation algorithm (called interpolation algorithm). The algorithm reproduces the relevant signals that describe the sketch geometry. These signals go to the drivers and the stepper motors rotates accordingly.

The total path, the machining tool has to follow, it can be analyzed in smaller linear and circular paths and finally the control program is a combination of linear and circular interpolation algorithms. These algorithms usually are written in assembly language in order to have as much as possible instant response. According the interpretation approximation of G code discriminated two categories of interpolation algorithms

The first one (Reference Pulse Algorithm **[7]**) returns a discrete pulse for every axis. Every time the algorithm executed uses as data the current axis's position and based on the ending coordinates reproduce a pulse for each axis. This pulse will rotate the step motor for one microstep. Next the algorithm goes over from scratch having now as data the new positions in each axis. The last execution takes place when the new position is identical with the ending point. After this the algorithm is terminated. The frequency that the pulse generated is the velocity and it is related directly with the time the algorithm needs to be executed.

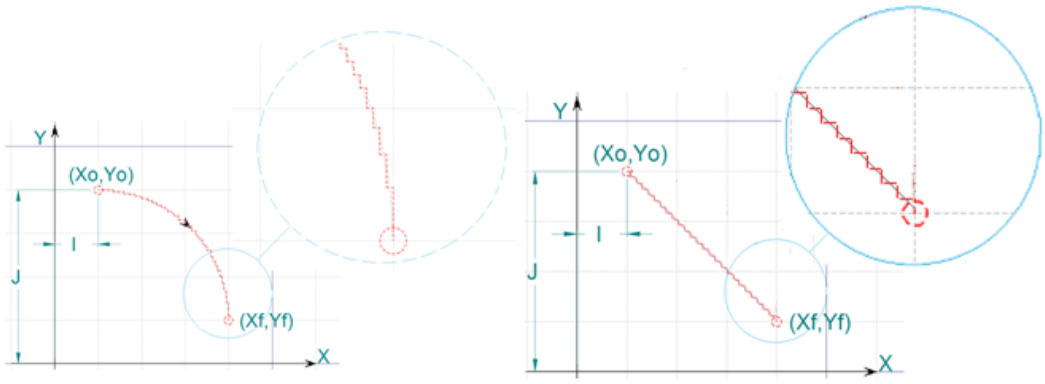


Image 5.1 - Reference Pulse Algorithm

Many of these algorithms are used also in digital image processing where the microstep rotation is corresponded in pixels. The Digital Differential Analyzer, the Stairs Approximation Method and the Direct Search Method are some algorithms belong in this category.

The second category (Reference Word Algorithm **[8])** instead of a single pulses sequence, returns one n-bit binary word for the displacement and another one m-bit for the velocity for each axis. The binary words arise from equations of linear motion with stable velocity having as initial data the start and end point, and the desirable velocity. For the velocity, a value is stored in a dedicated for that purpose register (named Velocity Counter). Each second time the velocity counter nullify a pulse generated to provoke one microstep rotation. In a second register (Displacement Counter) stored a value for the displacement. For every microstep rotation the displacement counter is reduced until nullify (in this point motion stops).

Which category will be used depends on the application requirements and the available hardware. The reference pulse based algorithms are simple in programming but they put restrictions in max velocity making them unsuitable for application with great demand in velocity. The reference word algorithms does not put restriction in velocity but they are more complex and not so accurate.

In the current application reference world algorithms will be used

Reference Word, Linear and Circular Interpolation Algorithm

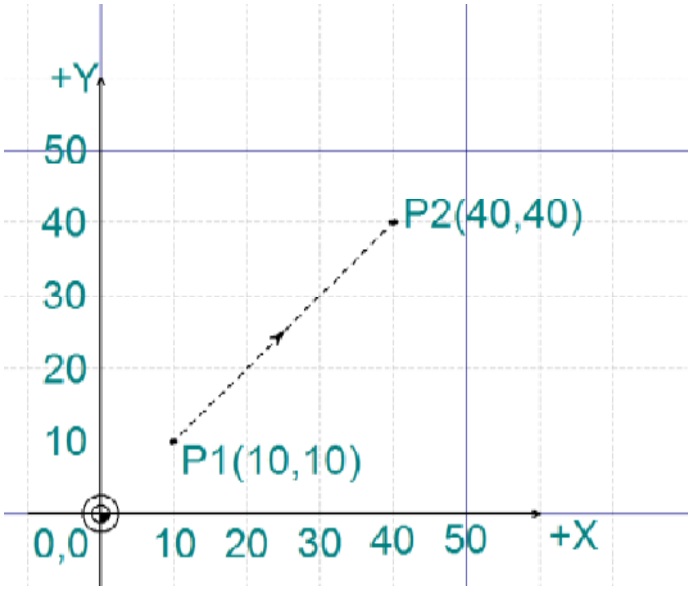
The application program reads the G code from the text file .nc and according the command generates one 8-bit acknowledge code. Then calls the suitable function to run the appropriate interpolation algorithm (equations of motion with steady velocity). The acknowledge code is sent to the microcontroller in order to call the appropriate Service Routine to serve the interpolation algorithm.

G Code Command	8-bit Acknowledge code	Interpolation Algorithm Function	Microcontroller's Service Routine
G00	'00000000'	Linear motion with max velocity	Linear motion with max velocity
G01	'00000001'	Linear motion with feed rate	Linear motion with feed rate
G02	'00000010'	CW Arc	Circular motion
G03	'00000011'	CCW Arc	
M30	'00111011'	Exit	Stop

Table 5.1 - Acknowledge Codes for G code Commands

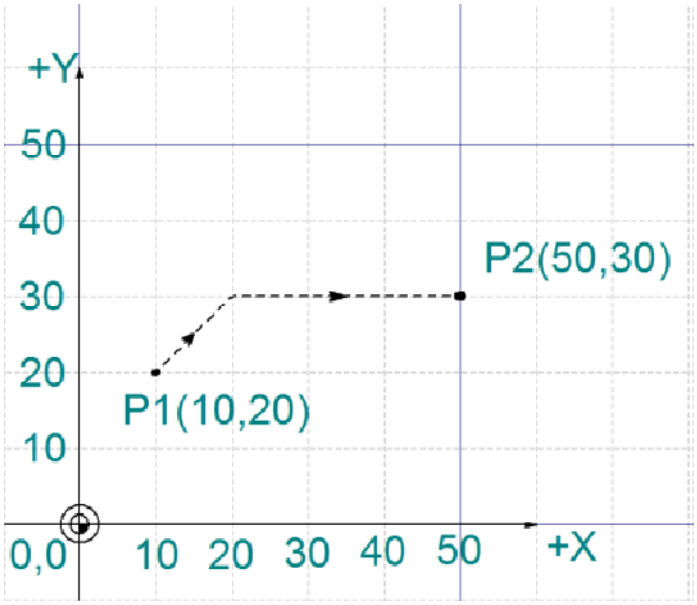
5.1 Linear motion with max velocity in both Axis - G00 command

The ISO wants max velocity for the axis with the larger displacement and the velocity in the axis with the smaller displacement to adjust accordingly so finally the two axis will reach the end point simultaneously.



Draft 5.1 - G00 with max velocity in the axis with the bigger displacement

However, the approximation in the current application is: motion with max velocity (Vmax) in both axis. This means that the axis with the smaller displacement will arrive at the end point first and then will follow the axis with the larger displacement.



Draft 5.2 - G00 with max velocity in both Axis

The displacement for each axis:

$$dx = x_2 - x_1 (5.1\ a)$$

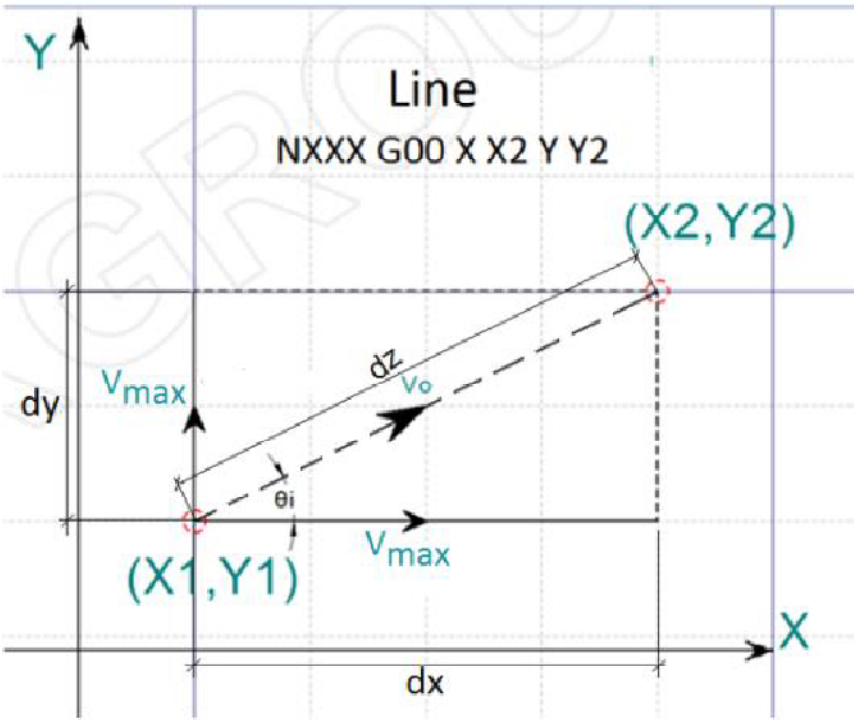
$$dy = y_2 - y_1 (5.1\ b)$$

the sign in 5.1a determines the value in the pin PB0 in PORT B (Start Byte) which is connected to the pin DIR in X axis driver and specifies the rotation direction. Similarly the sign in 5.1b specifies the rotation direction of Y axis motor through PB5. If the result is zero the PORT B disables the correspondent driver (EN(able) bit).

Conversion from mm to microsteps:

$$stepx = \frac{|dx|}{microstep} (5.2\ a)$$

$$stepy = \frac{|dy|}{microstep} (5.2\ b)$$



Draft 5.3 - G00 motion

Velocity representation in binary word:

$$OCRnA = \frac{f}{2N(\frac{v_{max}}{micro\_step})} - 1 \ (5.3)$$

the values for the displacement stepx and stepy are stored in predefined registers and the correspondent velocities to OCR0A and OCR2A. Last follows the Start Byte in

Port B, which determines the motor's rotation direction, the disable of header, and the enable or disable of drivers.

G00 Service Routine (Assembly) (Appendix A2)

the assignment of Start Byte in PORT B triggers the start of motion. The microcontroller manipulates with different procedures the position in x axis rather than in y axis

y axis position control:

Every time the timer 2 identified with the compare register OCR0A the flag TIFR2 on OCF2A register becomes '1'. A loop examines continuously this flag and every second identification the stepy counter reduced one microstep. The reason why the reduction takes place after two compare matches is that in order to stimulate a microstep rotation it needs one pulse in the STEP pin of driver. The inspection goes on until the counter stepy nullify. In this point the motion in y axis stops (Set '1' Enable pin of driver Y)

the y axis position control is interrupted temporarily from the compare match interruption of timer 0 when the identification with the compare register OCR0A takes place. The interrupt calls the routine for the x axis position control

x axis position control:

Every time the timer 0 is indentified with the compare register OCR0A, an interrupt is activated and it audits one user's defined flag. If the flag is '0' it changes it to '1' and it returns to the program (y axis position control). if the flag is '1' reduces the stepx counter one microstep, and returns to the program (y axis position control). When the stepx counter nullifies the motion in y axis it stops.

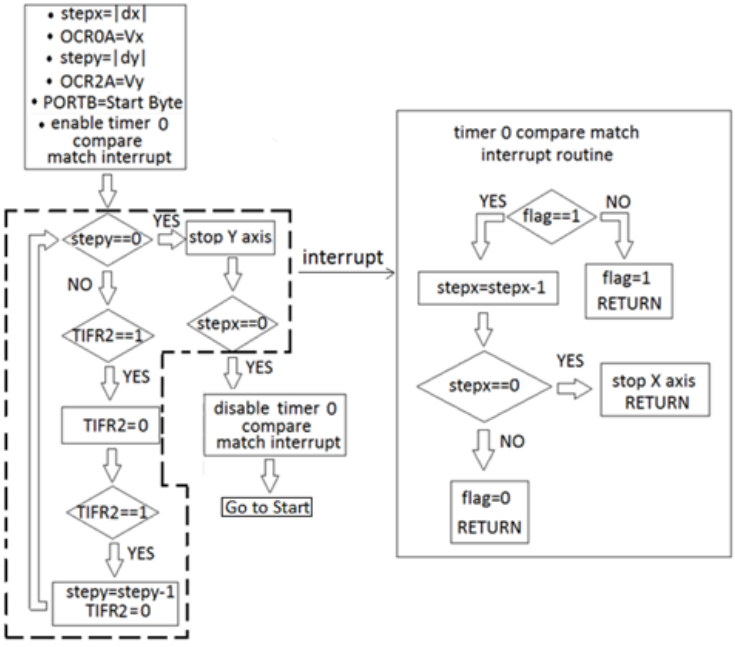
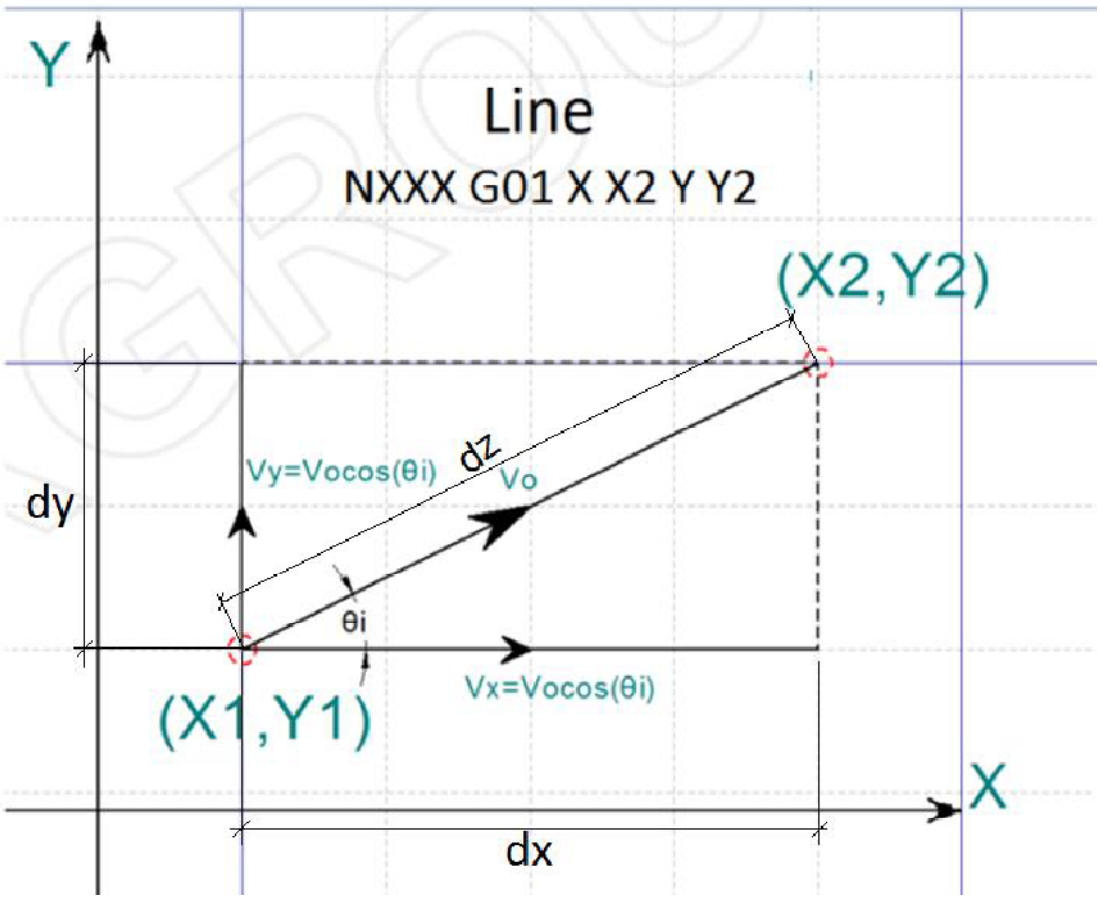


Image 5.2 - G00 Service Routine Flow Chart (Appendix A2)

5.2 Linear motion with feed rate - G01 command (Appendix A3)

The motion in both axis is linear with stable velocity. The velocity's value in each axis is estimated in order the resultant velocity Vo to be equal with the desirable feed rate.



Draft 5.4 - G01 motion

the displacement in each axis is:

$$dx = x_2 - x_1 (4.71\ a)$$

$$dy = y_2 - y_1 (4.71\ b)$$

the sign in 4.71a and 4.71b determines the motor's rotation direction and the enable - disable of drivers with the same way as explained in G00 command.

From the Pythagorean theorem the resultant displacement is:

$$dz = \sqrt{dx^2 + dy^2} (5.4\ )$$

In order to:

1. reduce the time consumed in communication between the application program and the microcontroller and

2. to minimize the microcontroller's tasks.

The position control will be executed only in the x axis. This means that the data for the displacement in y axis does not need to be transferred in microcontroller and so there in no routine for position control in y axis. While the motion is linear with steady velocity what it needs is the right values of velocities in both axis. With the right values in velocities the two axis will reach the same time in the end point. Knowing this, the displacement control only takes place in one axis (X axis) and when it reaches the end point (X2,Y2) the motion stops in both axis.

Note

The operation in G01 command by using only the position control in one axis (for example x axis) cannot work when the displacement is this axis is zero (In vertical motion: dx=0→vx=0→OCR0A=0). To surpass this obstacle the displacement and velocity data are taken from the other axis (with no zero displacement). (dx=dy→stepx=stepy | vx=vy→OCR0A=OCR2A | driver x disable)

The displacement in Y axis needs to calculate the velocity in Y.

$$dz = v_o \times dt \rightarrow t = \frac{dx}{v_0} \quad (5.5)$$

and

$$dx = v_x \times t \rightarrow v_x = \frac{dx}{t} \overset{4.}{\rightarrow} v_x = \frac{dx}{dz} v_o \quad (5.6\ a)$$

$$dy = v_y \times t \rightarrow v_y = \frac{dy}{t} \overset{4.}{\rightarrow} v_y = \frac{dy}{dz} v_o \quad (5.6\ a)$$

Conversion of dx from mm to microsteps (4.72 a). And the values in OCR0A and OCR2A

$$OCR0A = \frac{f}{2N(\frac{v_x}{micro\_step})} - 1 \quad (5.7\ a)$$

$$OCR2A = \frac{f}{2N(\frac{v_y}{micro\_step})} - 1 \quad (5.7\ b)$$

the value in the register OCR0A (8-bit) is analog to the signal's frequency in the pin STEP of drivers. So big values in OCR0A leads to small velocities while small values gives bigger velocities.

The values for the dx displacement and the velocities OCR0A (X Axis) and OCR2A (Y Axis) stored in dedicated registers. Last goes the Start Byte in PORT B, where determines the revolution direction, enables the header and the enable or disable the drivers.

G01 Service Routine (Assembly)

The assignment of Start Byte in PORT B triggers the beginning of motion. The microcontroller inspect only the displacement in x axis with the same mechanism as in G00 command in y axis control. After the counter nullify the motion terminated in both axis.

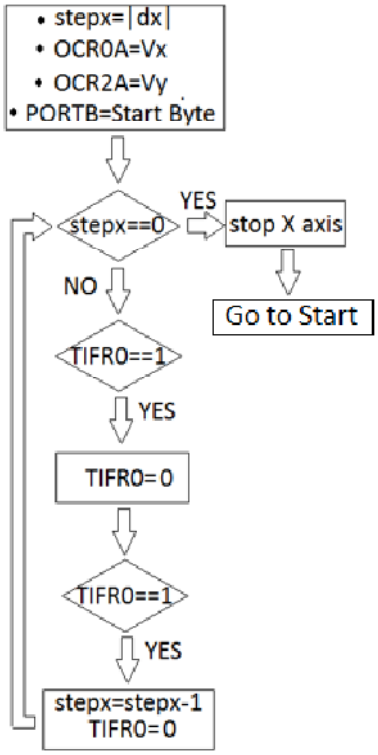
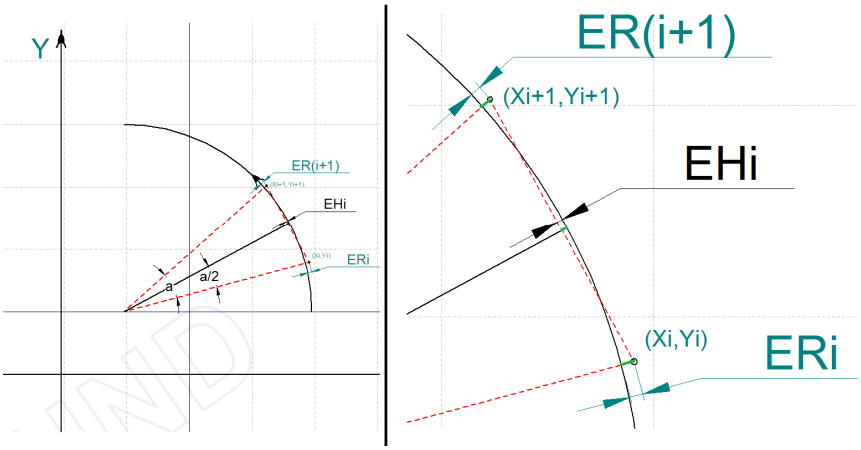


Image 5.3 - G01 Service Routine Flow Chart (Appendix A3)

5.3 CCW Arc with feed rate - G03

The arc is approximated as a part of a polygon which is enrolled in it. By this way the motion is analyzed in a number of very small linear segments (which actually is the polygon's sides). This arc's approach is evaluated by two types of errors, the Chord Error (EH) and the Radius Error (ER) **[8]**.

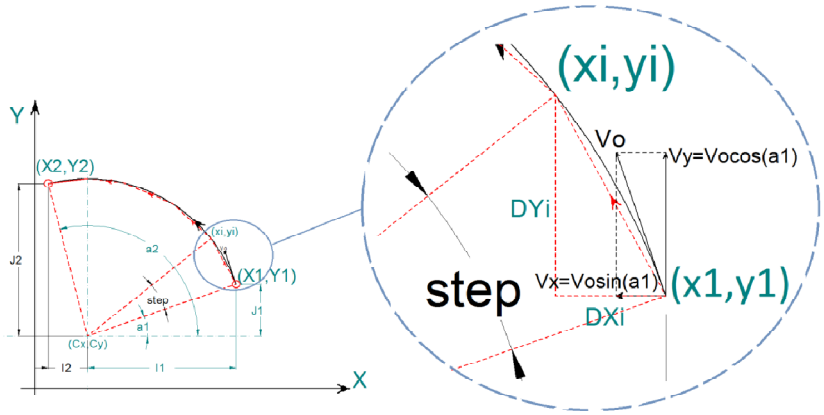


Draft 5.5 - Chord Error (CR) and Radius Error (RE)

The motion for moving upon this tiny linear segments is linear with steady velocity and the equations that describe this motion are the same as described previously in G01 command. The extra calculation the algorithm has to perform in this case is to split the arc in [i] tiny linear segments and generate data for every segment. For that reason an angle 'a' is chosen as step (step angle) in order to find the end point of every [i] segment.

Smaller step angles 'a' means smaller segments and so smaller the chord error and the polygon approximates the arc with greater accuracy. For bigger angles the chord error increases and so, the deviation from the real arc, is making the visual result more polygon than arc.

So it is desirable, as much as possible, smaller step-angle 'a' . How small the angle a could be, depends on the time needed to solve the equations and the data to be transferred.



Draft 5.6 - polygon enrolled in it arc

The arc radius is calculated:

$$R = \sqrt{I_1^2 + J_1^2} \ (5.8\ )$$

The arc's center coordinates:

$$C_X = X_1 + I_1 \ (5.9\ a)$$

$$C_Y = Y_1 + J_1 \ (5.9\ b\ )$$

and the arc's end point coordinates relative to arc's center:

$$I_2 = C_X - X_2 \ (5.10\ a\ )$$

$$J_2 = C_Y - Y_2 \ (5.10\ b\ )$$

From the above, the angle a1 between the start point and the X Axis and the angle a2 between the end point and the X Axis:

$$a_1 = \text{atan}\left[\left(\frac{J_1}{I_1}\right) \times \frac{180}{\pi}\right] \ (5.11\ \alpha)$$

$$a_2 = \text{atan}\left[\left(\frac{J_2}{I_2}\right) \times \frac{180}{\pi}\right] \ (5.11\ b)$$

The end point coordinates of the first segment is for an angle a1 increased one step angle a1=a1+a. The end point coordinates for [i] segment is for angle a1 increased [i] steps angle, a1=a1+a*i. Every time the end point coordinates of the [i] segment can be found by the following equation:

$$x_{2i} = \cos\left[\frac{a_1 \times 180}{\pi}\right] \times R \ (5.12\ b)$$

$$y_{2i} = \sin\left[\frac{a_1 \times 180}{\pi}\right] \times R \ (5.12\ b)$$

By this way the end coordinates of the [i] segment calculated. And by knowing the start and end point the data for the microcontroller arises as described above in G01 command.

The bit in DIR pin in driver which determines the motor's direction in Start Byte depends on the quadrant runs the motion according the following table.

Motor's rotation direction according the quadrant for ccw arc (G03)			
Quadrant	Angle a1	X	Y
	0°<α≤90°	-	+
	90°<α≤180°	-	-
	180°<α≤270°	+	-
	270°<α≤360°	+	+

Table 5.2 - Motor's rotation direction according to quadrant for ccw arc (G03)

The algorithm is repeated as long as the relation a1≤a2 is confirmed. When a1>a2 the dedicated bits for enable-disable the drivers in Start Byte goes to '1' (Disable drivers) and motion stops.

(G02 & G03) Service Routine (Assembly) (Appendix A4)

In order to achieve as much as possible small step angle, the position control runs parallel with the data transmission for the next polygon's linear segment. The data for the first segment is transferred to microcontroller and the motion begins.

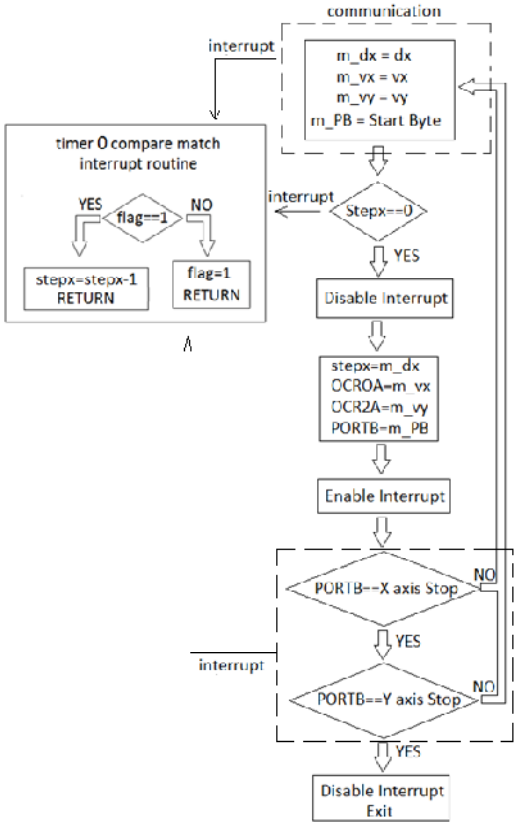


Image 5.4 - G02 and G03 Service Routine Flow Chart (Appendix A4)

The data for the next polygon's linear segment (displacement in x axis, velocity in x axis (vx), velocity in y axis(vy) and start Byte (Port B)) is stored in memory locations other than the dedicated registers which are used from the position control routine.

As long as the position control of [i] segment takes place, the new data for the next segment [i+1] is transferred to memory. When the timer 0 is identified with the value in compare register OCR0A, the compare match interrupt is executed (the data transfer temporarily stops) and calls the position control routine for position control. The position control routine inspects a user defined flag. If the flag is '0' changes to '1' and data transfer is continued. If the flag is '1' the displacement is reduced one microstep, the flag changes to '1' and the data transfer is continued.

In this way the position control runs parallel with the data transfer. When the displacement counter nullifies, the motion stops, the compare match interrupt is disabled, and the new data for the next (i+1) segment is moved to the dedicated registers.

Note:

During this period the motion goes on. So until the compare match interrupt enable again a small position control error occurs (Radius Error).

This transportation takes place almost instantaneously because it takes place internal in the microcontroller. After the dedicated registers take the new data the compare match interrupt is enabled again in order to run the position control.

In any case the data transfer should be completed before the axis goes to the end point.

Before the program returns to read the data for the next segment, it examines the PORTB. If the bits for the enable - disable drivers are '1' (drivers disable) it means that the arc is finished so the exit from the routine follows and returns to the main program to take the new G command. Otherwise Circular Interpolation Service Routine runs from scratch.

5.4 CW Arc with feed rate - G02

The case of CW arc is the same as in CCW. The only differences are that the application program runs until a2<a1 and that the bits which determine the revolution direction in Start Byte are given in the following table.

Motor's rotation direction according the quadrant for cw arc (G02)			
Quadrant	Angle a1	X	Y
	0°<α≤90°	+	-
	90°<α≤180°	+	+
	180°<α≤270°	-	+
	270°<α≤360°	-	-

Table 5.3 - Motor's rotation direction according to quadrant for cw arc (G02)

In the assembly the service routine is the same independently if the arc is CW -> G02 or CCW -> G03 (Appendix A4)

5.5 End of process - M30

The M30 command terminates the application program, disable the header and put microcontroller in sleep mode.

Part 6: Conclusions

6.1. Accuracy and Total Travel Distance

The smallest unit displacement (microstep) depends from the driver's prescaler:

Prescaler	Microstep (mm)
1	0.2
1/2	0.1
1/4	0,05
1/8	0,025
1/16	0,0125
1/32	0,00625

Table 6.1 - Driver Prescaler - Linear

The greatest accuracy is given by the smallest microstep. But the smaller the microstep is, the bigger the number of microsteps which comprise the total displacement ds are, and thus the value in the displacement register. For that reason two 8 bit registers used to store the displacement in each axis. With 16 bit available the maximum total distance that the machine can support is:

ds=216 * microstep

Prescaler	dsmax (mm)
1	13107,2
1/2	6553,6
1/4	3276,8
1/8	1638,4
1/16	819,2
1/32	409,6

Table 6.2 - Driver prescaler - max displacement correlation for 16bit register

From the mechanical design the maximum distance is dsmax=460 mm. This means that with a driver's prescaller = 1/32 a displacement greater than 409,60 mm will cause problem because the displacement register will overflow.

One solution is to use three 8-bit register for the displacement. This will increase the complexity in software (one more register to examine)

6.2. Accuracy and Hardware Capabilities

The equation $OCR0A = \frac{f}{2N(\frac{v_x}{micro\_step})} - 1$ (6.0) converts the velocity in binary number in the compare register. The factor N is the timer's prescaler. With N=1 the timer is changed for every clock pulse. If N=8 it needs eight clock pulses to change the value in timer.

Example: Velocity v=10,45 mm/sec and N=1024

$$OCR0A = \frac{16000000}{2 \times 1024 \times (\frac{10.45}{0.00625})} - 1 = 3.67 \text{ (6.1)}$$

Converting the decimal 3.67 in binary we miss the fractional part and finally the value in OCR0A register is:

$$OCR0A=(3)10=(11)(2\ bit)$$

This value gives a actual velocity:

$$v_R = \frac{f \times microstep}{2N(OCR0A+1)} = \frac{16000000 \times 0.00625}{2 \times 1024(3+1)} = 12{,}20\ mm/\text{sec} \quad (6.2)$$

The deviation here is significant:

$$dv=vR-v=12{,}20-10{,}45=1{,}75\ mm/\text{sec} \text{ (6.3)}$$

Changing the timer's prescaler to N=1 and

$$OCR0A = \frac{16000000}{2 \times 1 \times (\frac{10.45}{0.00625})} - 1 = 4784{,}68 = (4784.68)_{10} = 1001010110000_2(13\ bit) \text{ (6.4)}$$

the actual velocity:

$$v_R = \frac{f \times microstep}{2N(OCR0A+1)} = \frac{16000000 \times 0.00625}{2 \times 1(4784+1)} = 10.449\ mm/\text{sec} \quad (6.5)$$

The deviation here is acceptable:

$$dv=vR-v=10{,}449-10{,}45=-0{,}001\ mm/\text{sec}$$

So for greater accuracy is better to use a small timer prescaler. But a very small value in timer prescaler will give very large value in OCR0A register and it is only 8 bit.

Of course it is also possible to retain the accuracy by increasing software complexity.

It is possible to use a smaller value for timer prescaler (for ex. N=256, and improve the accuracy in velocity) and for correction to use a bigger driver prescaller (for ex. 1/8 and reduce the accuracy in displacement). In this way the ratio N/micro_step in the 6.0 equation maintain the same (1024/32=256/8)

6.3 Solution for Improvement

There are many points for further improvement so to software (CAM , Interpolation algorithms, Assembly) as to hardware (mechanical design , PCB). The most highlighted points are

Interpolation algorithms:

1) Approximation of the decimal values in OCR0A register (velocity in x axis) in the nearest integer. So for example the decimal value X,99 turns to X+1 (deviation 0.01) instead of X (deviation 0.99). So the actual velocity is approximated with greater accuracy
because the maximum error is limited up to 0.5 instead of 0.99.
2) Recalculation the new actual velocity in x axis using the approximated value of OCR0A. Calculation of the velocity in y axis take under consideration the new velocity of x axis using the equation:

$$v_y = \frac{dy}{dx} v_x$$

3) Approximation the decimal values in OCR2A register (velocity in y axis)

Hardware:

Instead of using timer 2 for the velocity in y axis , it could be better to use timer 1. The compare register OCR1A is 16 bit. Having 16 bit to store the value of velocity it gives the potential to set the timer's prescaler to N=1. This will increase to maximum the resolution of velocity in y axis.

This change presuppose , modification to the PCB. Wiring the STEP pin of driver Y to pin OC1A (PB1) of microcontroller and the pin and the SIGNAL line of servo motor to pin OC2A (PB3) of microcontroller. The microcontroller's software should adjust in these changes.

Displacement Control

The driver needs the generation of a pulse in his STEP pin to trigger a microstep rotation in motor. The microcontroller alters the value in STEP pin in every timer's compare identification with the compare match register. So two identifications are needed in order to produce a pulse. Every identification notified by examining a flag in a counter's register. For that reason, for the displacement control the assembly routine examines this flag and every second identification , it decreases 1 microstep the displacement register
If the number of microsteps (calculated in application program) are multiplied by two, it will make possible to take the right displacement by decreasing the displacement counter in assembly after every identification (not two). In this way it is possible to reduce the software complexity because the reduction in displacement counter will take place after one flag 's notification instead of two. On the other hand

this duplication in the number of steps leads in greater space in the displacement counter.

References

[1] ISO 6983-1:2009: Automation systems and integration - Numerical control of machines - Program format and definitions of address words - Part 1: Data format for positioning, line motion and contouring control systems. International Organization for Standardization

[2] Autodesk Inc. , February 2011 , DXF Reference, USA , https://images.autodesk.com/adsk/files/autocad_2012_pdf-reference_enu.pdf&ved

[3] Open Pulse, 42BYGHW208 Stepper Motor Datasheet

[4] Texas Instrument, DRV8825 Stepper Motor Controller IC datasheet [Rev. July 2014]

[5] Tower Pro, Servo Motor SG90 Datasheet

[6] Atmel, "8-bit AVR Microcontrollers", ATmega328/P _ Datasheet _ Complete, [Rev. Dec. 2016]

[7] O. Masory, Y. Koren, 1981, "Reference-Pulse Circular Interpolations for CNC Systems, Journal of Engineering for Industry, 103, 131-136, https://ykoren.engin.umich.edu

[8] O. Masory, Y. Koren, 1982, "Reference-Word Circular Interpolations for CNC Systems, Journal of Engineering for Industry, 104, 400-405, https://ykoren.engin.umich.edu

Appendices

A. Codes

A1. Align Geometrical Entities Algorithm (C++)

- (case 1)

```
//Align Entities Algorithm
//Written in Dev-C++ 5.11
//Author : Petros Iakovou
//Date: 02/02/2021

flag=1;
int revisions=0;                // to count how many loop will make
while(flag==1 && revisions<100) // normally the revisions<100 does not need. but i will use it for safe is for some reason will have continious loop
{
    flag=0;                    // make the flag =0 if no any changes happens flag will remain zero and will not repeat the loop
    for (j=0;j<=i_E;j++)
    {
        for (k=j+1;k<=i_E;k++)  //compare with another (k) entity (line or arc)
        {

//============CASE 1 : if the end point of the current entity k is the same with end point of another k entity===========

            if(E[k].Get_x2()==E[j].Get_x2() && E[k].Get_y2()==E[j].Get_y2())
            {

                s1=E[k].Get_type();      // take the type of (k) entitity

                if(s1=="G01")    // if it is line
                {
                E[k].SetLine((char*)"G01",E[k].Get_x2(),E[k].Get_y2(),E[k].Get_x1(),E[k].Get_y1());     //reverse line's coordinates
                }
                if(s1=="G03")      // if it is arc
                {
                    rx=E[k].Get_x1()+E[k].Get_I1();      // find arc's center x coordinate
                    ry=E[k].Get_y1()+E[k].Get_J1();      // find arc's center y coordinate
                    I=rx-E[k].Get_x2();                 //  find I2
                    J=ry-E[k].Get_y2();                 // find J2
                    E[k].SetArc((char*)"G02",E[k].Get_x2(),E[k].Get_y2(),E[k].Get_x1(),E[k].Get_y1(),I,J);   //reverse arc's coordinates
                }
                flag=1;      // raise flag
            }
```

- (case 2 and case 3)

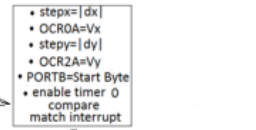
```
//=============CASE 2: if the start point of the current entity j is the same with start point of another k entity===========

        if(E[k].Get_x1()==E[j].Get_x1() && E[k].Get_y1()==E[j].Get_y1())
        {
            s1=E[k].Get_type();

            if(s1=="G01") // if it is line
            {
                E[k].SetLine((char*)"G01",E[k].Get_x2(),E[k].Get_y2(),E[k].Get_x1(),E[k].Get_y1());   //reverse line's coordinates

            //swap entities
                temp=E[j];
                E[j]=E[k];
                E[k]=temp;
            }
            if(s1=="G03")   // if it is arc
            {
                rx=E[k].Get_x1()+E[k].Get_I1();      // find arc's center x coordinate
                ry=E[k].Get_y1()+E[k].Get_J1();      // find arc's center y coordinate
                I=rx-E[k].Get_x2();                 //  find I2
                J=ry-E[k].Get_y2();                 // find J2

                E[k].SetArc((char*)"G02",E[k].Get_x2(),E[k].Get_y2(),E[k].Get_x1(),E[k].Get_y1(),I,J);  //reverse arc's coordinates

            //swap entities
                temp=E[j];
                E[j]=E[k];
                E[k]=temp;
            }
            flag=1;    // if it is line
        }

//=============CASE 3: if the start point of the current entity j is the same with the end point of another k entity===========

        if(E[j].Get_x1()==E[k].Get_x2() && E[j].Get_y1()==E[k].Get_y2())
        {
            //swap entities
                temp=E[j];
                E[j]=E[k];
                E[k]=temp;

            flag=1;
        }
```

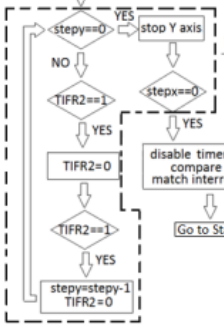
- alignment

```
//=============swap entities algorithm====================
//-----if one swap in coordinates happens in previous step we should rearrange the lines order to avoid a continious loop in swap in coordinates (journal 01/01/2021)
            if(flag==1)    //while flag =1 it means that in previous step we make one revesion in entities coordinate
            {
                for (j=0;j<=i_E;j++)
                {
                //if the next (j+1) entity is continious with the current (j) entity move to the next j+1 line
                    if(E[j+1].Get_x1()!=E[j].Get_x2() || E[j+1].Get_y1()!=E[j].Get_y2())
                    {
                        for(k=j+2;k<=i_E;k++)           //  start the comparison with the (k=j+2) entity
                        {
                            if(E[k].Get_x1()==E[j].Get_x2() && E[k].Get_y1()==E[j].Get_y2())  // if k line is continious with the first swap j+1 with k
                            {
                              //swap entities
                                temp=E[j+1];
                                E[j+1]=E[k];
                                E[k]=temp;
                            }
                        }
                    }
                }
            }
//================end of swap entities algorithm===============
        }
    }
revisions++;            // how many revision will need to allign entities?
}
```

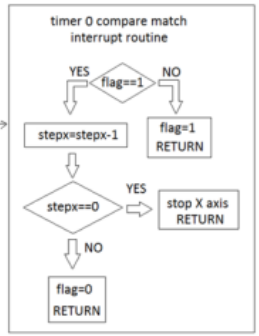
A2. G00 Service Routine (Assembly)

```
G_00:                                   Dicrease Displacement  in G00
RCALL    Pen_U                          Date:06/05/2021
                                        Written in AtmelStudio
Communication to Give                   Author:Petros Iakovou
   Data to Registers                    Version: V38
                                        C++ executable: CAD2G V42
//6th    PORTB
RCALL    Transmit       //confirm
RCALL    Receive        //receive  motion parameters
OUT      PORTB,TEMP     //start motion

//Enable Timer 0 Compare Match OCR0A Interrupt
LDI      TEMP,0b00000010
STS      TIMSK0,TEMP

SEI //Enable Global Interrupts

//RCALL  Accelarate

RCALL    Count_Steps2 // Call Function Dicrease Displacement In Y Axis in G00

SBI      PORTB,PB4   //Disable Driver in Y Axis

//if Y axis stop motion before x axis wait in Decrease_x axis loop:
Decrease_x:
          L1: CPI     STEPH,0
              BRNE    L1      // if STEPH!=0 goto L1 else move to L2
          L2: CPI     STEPL,0 // if STEPL!=0 goto L2 else continue to the programm
              BRNE    L2

Escape:       CLI     // Disable Global Interrupts

              RJMP    Start

              ;--------------------Dicrease Displacement In Y Axis in G00 by function call----------------------
              Count_Steps2:
              //Check for displacement in Y axis (STEPY>0)
                              CPI     STEPYH,0b00000000    // compare STEPYH with 0
                              BRNE    half_pulse2     // //if STEPYH ==0 execute function
                              CPI     STEPYL,0b00000000   // compare STEPYL with 0
                              BREQ    exit            // if STEPY==0 return to main programm
              //loop for the first compare match
              half_pulse2:
                              SBIS    TIFR2,OCF2A   // skip if TIFR2 flag == 1 (compare match)
                              RJMP    half_pulse2   // loop if TIFR2 flag == 0
                              SBI     TIFR2,1       // make TIFR2 == 0
              //loop for the second compare match
              one_pulse2:
                              SBIS    TIFR2,OCF2A    // skip if TIFR2 flag == 1 (compare match)
                              RJMP    one_pulse2     // loop if TIFR2 flag == 0
                              SBIW    STEPYL,1       // reduce STEPYL
                              SBI     TIFR2,1        // make TIFR2 == 0
                              BRNE    Count_Steps2   // if STEPY !=0 repeat function else return to main programm
              //Return to main programm
              exit:           RET
              ;-----------------------------------------------------------------------------------------------
```





interrupt

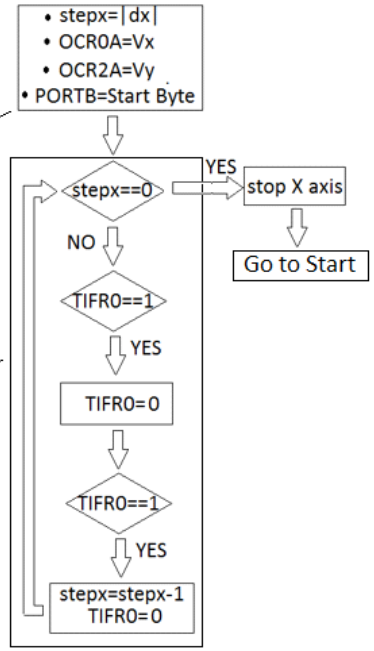



```
;--------Dicrease Displacement In X Axis in G00 using Timer 0 Compare Match Interrupt ----------------------------------
OCR0A_Overflow:
                TST     COUNTER   //use this flag to see if there is the first or second compare match after 1 step reduction
                BRNE    Decrease_Dx  //if COUNTER!=0 one pulse have been generated go to Decrease_Dx
                INC     COUNTER      // if COUNTER==0 half palse have been generated

                RETI              // Return to main programm

Decrease_Dx:    SBIW    STEPL,1 // decrease displacement in x axis 1 step
                BREQ    Dis_Int_T0 //if displacement in x == 0 go to Dis_Int_T0
                LDI     COUNTER,0b00000000  // make flag COUNTER ==0

                RETI   //Return to main programm

 Dis_Int_T0:    SBI     PORTB,PB2   //disable driver x
                LDI     TEMP,0b00000000
                STS     TIMSK0,TEMP //disable interrupt Timer 0 Compare Match
//Return from Interrupt
                RETI
```

A3. G01 Service Routine (Assembly)

```
Dicrease Displacement  in G01
Date:06/05/2021
Written in AtmelStudio
Author:Petros Iakovou
Version: V38
C++ executable: CAD2G V42
;=============================================================================================================
G_01:
        // i can not reduce precsaller of timer0 because i use the displasement in x axis to control movment
                LDI     TEMP,0b000000101  // prescaller timer0 1024
                OUT     TCCR0B,TEMP
        // but i can reduce precsaller of timer0 in combination with prescaller in driver to achive better
        // accurancy as i do not check for displacement in y axis
                LDI     TEMP,0b00000110  // prescaller timer2 128
                STS     TCCR2B,TEMP

                //PEN DOWN
                RCALL    Pen_D

                RCALL    Count_Steps
                RJMP     Start
;=================Dicrease Displacement In X Axis in G01 by function call=====================================
Count_Steps:
//loop for the first compare match
        half_pulse:
                SBIS    TIFR0,OCF0A  // skip if TIFR0 flag == 1 (compare match)
                RJMP    half_pulse   // loop if TIFR0 flag == 0
                SBI     TIFR0,1      // make TIFR0 == 0
//loop for the second compare match
        one_pulse:
                SBIS    TIFR0,OCF0A   // skip if TIFR0 flag == 1 (compare match)
                RJMP    one_pulse     // loop if TIFR0 flag == 0
                SBIW    STEPL,1       // reduce STEPX one step
                SBI     TIFR0,1       // make TIFR0 == 0
                BRNE    Count_Steps   // if displacement in x !=0 repeat function else return to main programm

                RET
;=============================================================================================================
```

Communication to Give Data to Registers

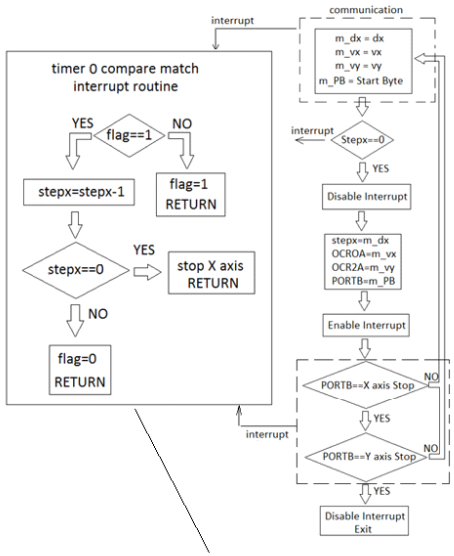


A4. G02 & G03 Service Routine (Assembly)

```
Dicrease Displacement  in G03
Date:06/06/2021
Written in AtmelStudio
Author:Petros Iakovou
Version: V38
C++ executable: CAD2G V42
G_03:
                RCALL    Pen_D   //   enable header
```

Communication
Store Data to memory
not to working registers

```
                MOV     DIR,TEMP        //start or stop motion*/
            loop:
                CPI     STEPL,0     //if displacement in x axis!=0 go to loop
                BRNE    loop

                // Enable Timer 0 Compare Match OCR0A Interrupt
                LDI     TEMP,0b00000010
                STS     TIMSK0,TEMP

                //Disable Global Interrupts
                CLI

                //Give Data to Working Registers
                MOV     STEPL,DXL       // displasement in x axis
                OUT     OCR0A,V_X       // velocity in x axis
                STS     OCR2A,V_Y     // velocity in y axis
                OUT     PORTB,DIR     // start bit

                //Enable Global Interrupts
                SEI

                //if motors in both axis are disable exit the Count_Steps_G03 loop
                //it can work by check only one motor (x or y) but although it is
                //faster i think that is better to check both axis
                SBIC    PORTB,PB4       //if driver Y is enable go to Count_Steps_G03
                SBIS    PORTB,PB2       //if driver X is disable jump next instruction

                RJMP    G_03

                CLI
                RJMP     Start
```




```
;========Dicrease Displacement In X Axis in G00 using Timer 0 Compare Match Interrupt ====================================
OCR0A_Overflow:
                TST     COUNTER   //use this flag to see if there is the first or second compare match after 1 step reduction
                BRNE    Decrease_Dx  //if COUNTER!=0 one pulse have been generated go to Decrease_Dx
                INC     COUNTER     // if COUNTER==0 half palse have been generated

                RETI             // Return to main programm

Decrease_Dx:    SBIW    STEPL,1 // decrease displacement in x axis 1 step
                BREQ    Dis_Int_T0 //if displacement in x == 0 go to Dis_Int_T0
                LDI     COUNTER,0b00000000  // make flag COUNTER ==0

                RETI   //Return to main programm

 Dis_Int_T0:    SBI     PORTB,PB2   //disable driver x
                LDI     TEMP,0b00000000
                STS     TIMSK0,TEMP //disable interrupt Timer 0 Compare Match
//Return from Interrupt
                RETI
```

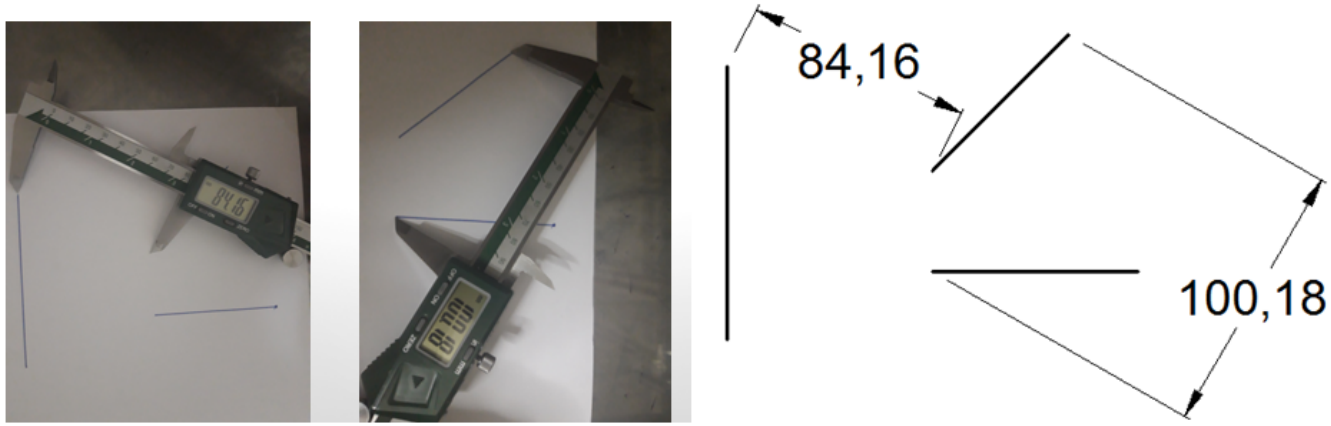
B. Bill of Materials

BILL OF MATERIALS						
a/a	**Code**	**Description**	**NoP**	**Κατεργασία**	**Price**	**TotalPrice**
1	SCS8UU	Linear ball bearing paltform (Small) Φ8	8	GROBOTRONICS	3,80 €	30,40 €
2	1A0LE2R20D084409-A	Driving Shaft Φ8 inox- X and Y Axis	4	GROBOTRONICS	3,00 €	12,00 €
3	1A0LE2D12GT2T20B064-A	Aluminium GT2 Timing Pulley - 6 mm Belt - 20 Tooth - 6 mm Bore	3	GROBOTRONICS	1,60 €	4,80 €
4	1A0LE2SPY01N17SCS8UU7-A	Supportplate - X axis	4	Puching	0,20 €	0,80 €
5	42BYGHW208	Stepper Motor 2.8 kgcm (200 steps/rev)	2	GROBOTRONICS	12,60 €	25,20 €
6	1A0LE2SPX01N17SCS8UU7-A	Supportplate - Y axis	2	Puching	0,20 €	0,40 €
7	1A0LE2D12GT2I20B064-A	Aluminium GT2 Timing Pulley Idler - 20T Smooth - 5 mm Bore	6	GROBOTRONICS	2,90 €	17,40 €
8	1A0LE2R20D064XXX-A	Motion Connector Shaft Φ6- Y Axis	1	Lathe	5,00 €	5,00 €
9	1A0LE2BC4057-A	Aluminum Flex Shaft Coupler - 5mm to 6mm	1	GROBOTRONICS	3,20 €	3,20 €
10	Q020-06-12	Spacer 6X12 (X Axis Motor)	4	Lathe	0,05 €	0,20 €
11	W451V16C5R	Terminal Switch	2	GROBOTRONICS	0,50 €	1,00 €
13	Q131-05-012	Allen Bold With Head -M6X12	8	GROBOTRONICS	0,05 €	0,40 €
14	1A0LE2BC4037-A	SupportCornes - X Axis	4	Puching	0,20 €	0,80 €
15	Q020-06-24	Spacer 6X24 (X Axis Support Plate)	12	Lathe	0,05 €	0,60 €
16	1A0LE2SPY404S037-A	Ball Bearing Support Plate flange	1	3D Printer	0,10 €	0,10 €
17	1A0LE2SPY403S037-A	BallBearingSupportPlate	2	Puching	0,20 €	0,40 €
18	MF126ZZ	Ball Bearing Flanged - (6mm Bore, 12mm OD)	1	GROBOTRONICS	1,40 €	1,40 €
20	Q020-07-05	Spacer 7,3X5	4	Lathe	0,05 €	0,20 €
21	Q010-01-08	TimingBelt - GT2	2	GROBOTRONICS	2,40 €	4,80 €
23	Q141-05-010	Allen Bold With Head -M5X10	4	GROBOTRONICS	0,05 €	0,20 €
24	Q41-05-150	Washer Φ5X150	4	GROBOTRONICS	0,05 €	0,20 €
25	1A0LE2H104504-A	Joint SupportCornersShaft	2	Lathe	0,50 €	1,00 €
26	Q131-04-012	Allen Bold With Head -M4X12	16	GROBOTRONICS	0,05 €	0,80 €
27	1A0LE2BC4117-A	TrackSupport Y Axis	1	3D Printer	0,10 €	0,10 €
28	1A0LE2BC4127-A	TrackSupport X Axis	1	3D Printer	0,10 €	0,10 €
29	1A0LE2BC4157-A	Base for Terminal switch - X axis	1	3D Printer	0,10 €	0,10 €
30	1A0LE2BC4187-A	Base for Terminal switch - X axis	1	3D Printer	0,10 €	0,10 €
31	1A0LE2BC4207-A	Header - PenHolder	1	3D Printer	0,50 €	0,50 €
32	W50-001-021 (DRV 8825)	Shield (ARDUINO UNO)	1	GROBOTRONICS	10,00 €	10,00 €
33	Cables	Cables	1	GROBOTRONICS	5,00 €	5,00 €
35	SG90	Servo Micro 2.2 kg.cm Plastic Gears (Waveshare SG90)	1	GROBOTRONICS	3,6	3,60 €
36	ARDUINO UNO REV 3	ARDUINO UNO	1	GROBOTRONICS	22	22,00 €
37	Q020-10-15	CableDragChain 10x15mm	1	GROBOTRONICS	5	5,00 €
38						0,00 €
39					Total=	157,80 €

Table B - Bill of Materials (Coding development for every Part)

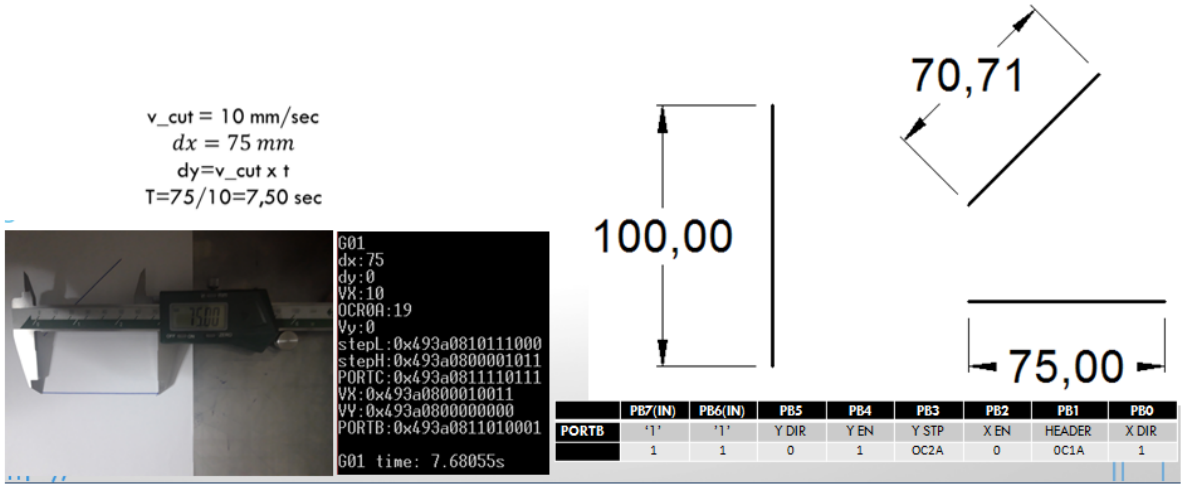
C. Operation Tests

- G00

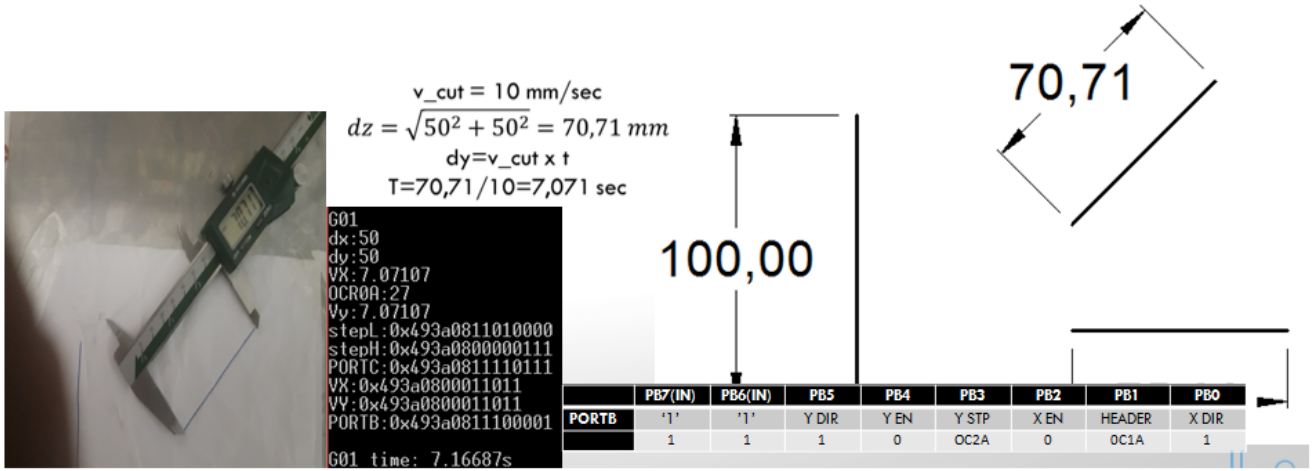


- G01 (vertical line)

- G01 (horizontal line)



- G01 (inclination line)



- G02-G03