POLITECNICO DI MILANO

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
Degree Course of Master Of Science in Mechanical Engineering

Development of a 3 D.o.F. Platform for Additive Manufacturing based on MIM Technique

Thesis Supervisor

Prof. Ing. Hermes Giberti

Thesis by

Kevin Castelli
id: 832716

Academic year 2015-2016

To study and at times practice what one has learned, is that not a pleasure?
Confucius

Acknowledgements

I'd like to thank prof.Giberti for giving me the opportunity to work on a innovative project where I could prove the knowledge and ability learned in a life time of studies. He allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it.
I take the chance to thank also PhD student Luca Sbaglia that was more than helpful to enlighten the project so far and to where it was heading for.
I'd like to express my very profound gratitude to my family for the support in these years. This accomplishment would not have been possible without them.

Many thanks to you all.

Milan, April 2017

Contents

List of Figures

Abstract

In the Mechanical Department of Politecnico di Milano a project has started to develop a new 3D printer based on a MIM, metal injection molding, technology aiming to print metal objects with different mechanical properties and with lower costs than ones printed by usual techniques.

This master thesis continuous on an already started project where an automated machine for 3D printing has been designed and built based on a linear delta robot with 3 D.o.F., degrees of freedom, and a MIM extruder. The aim of the work is to develop a machine capable to provide a complete set of functionalities as usual 3D printers. In order to permit the correct study and development of this new 3D printing technology the machine must be able to print from a G-code, typical command language used for automated machines, allow a generic movement of the system, and allow the control and regulation of specific technological parameters. To reach these goals a study and a development of the mechanic and control system are done. Manufacturing tolerances on mechanical components and imprecision during assembly can lead to numerical mis-estimation of the geometrical parameters used in the kinematic equations of the control system. Thus a calibration method is developed to obtain the actual measures to ensure the correct positioning of the robot. The robot structure needs to be stiffened and an adjustable plate is designed to ensure perpendicularity of the moving plate with the axis of the extruder nozzle. A control scheme with its electrical and electronics components is developed with the aim to power the system, measure correctly some key values of the machine and control it through the use of a Mitsubishi®PLC, programmable logic controller, and a motion control unit. This system is fit inside an electrical cabinet. The control programs are developed using Matlab®and native Mitsubishi®software able to analyze the object to print, either from CAD software (parsing effectively a G-code) or user generated equations, to assign a given motion law and to execute them on the final system. It's presented a *CNC emulator* capable to provide these functionalities taking into account the constraints of the Mitsubishi®control system as for instance the management of the internal memory. A manual control is also developed for a generic use of the machine by an operator.

Key world: parallel kinematics robot, linear delta, adjustable plate, calibrating procedure, CNC emulator, control system

Estratto

L'Additive Manufacturing, AM, é una tecnologia che sta ricevendo molta attenzione dai media e si sta diffondendo nel campo scientifico e industriale. Molte ricerche sono finalizzate alla creazione di componenti stampati a fini pratici e non solo per la realizzazione di prototipi, ancora oggi una delle principali applicazioni di questa tecnologia. La realizzazione di oggetti metallici é solitamente ottenuta per mezzo di laser o di EBM (electron beam melting). Il Dipartimento di Meccanica del Politecnico di Milano ha iniziato da circa due anni un progetto per sviluppare una nuova stampante 3D basata sulla tecnologia MIM (metal injection molding) al fine di stampare oggetti metallici con diverse proprietà meccaniche e a costi ridotti rispetto alle altre metodologie.

Questa tesi continua un progetto già iniziato dove una macchina automatica per stampaggio 3D é stata progettata e realizzata usando un linear delta robot a 3 gdl (gradi di libertà) e un estrusore MIM. Il sistema di estrusione é in grado di depositare un filamento, composto da metallo e un legante, su una piastra mobile mossa dal linear delta. Lo scopo di questo lavoro é di sviluppare un macchina in grado di offrire le medesime funzionalità di una stampante 3D integrando i due sottosistemi inizialmente separati, il linear delta robot e l'estrusore. Essendo la macchina una piattaforma di studio di questa nuova tecnologia AM, é importante fornire il controllo di specifici parametri tecnologici che caratterizzano il processo di stampa, inoltre la macchina deve essere integrata con i software già esistenti utilizzati dalle stampanti 3D, come ad esempio i software di slicing. È inoltre richiesto un sistema di controllo in grado di coordinare sia la piattaforma mobile che l'estrusore al fine di stampare un pezzo a partire da un G-code, uno dei linguaggi di comando più usato per le macchine automatiche, o da una qualsiasi traiettoria generata a scopi di studio assicurando allo stesso tempo il raggiungimento dei parametri precedentemente menzionati per garantire test affidabili sulla tecnologia. Il lavoro é suddiviso nello studio e sviluppo del sistema meccanico e di controllo della macchina. Tolleranze di realizzazione dei componenti meccanici e imprecisioni durante il montaggio possono portare a una differenza tra i valori effettivi di tali componenti e i valori numerici utilizzati dal sistema di controllo nelle equazioni della cinematica inversa. Un metodo di calibrazione é stato sviluppato per ottenere i valori reali dal momento che sono fondamentali per garantire il corretto posizionamento del robot. Questo strumento si basa sul confronto fra la posizione misurata e quella ottenuta tramite le equazioni cinematiche del sistema. I parametri cinematici del sistema vengono fatti variare iterativamente confrontando la posizione simulata del robot

con quella effettivamente misurata. Per far ciò, la struttura del robot deve essere irrigidita e la piastra mobile del linear delta deve essere perpendicolare all'asse dell'ugello dell'estrusore, condizione ottenuta tramite la progettazione di un piatto regolabile. Lo schema di controllo con i suoi componenti elettrici ed elettronici é stato sviluppato allo scopo di alimentare il sistema, misurare correttamente alcune caratteristiche chiave della macchina e controllarle attraverso l'uso di un PLC, programmable logic controller, e una unità di controllo di movimento (Motion) scelti dalla casa Mitsubishi®. L'intero sistema é progettato per essere collocato all'interno di un armadietto elettrico. I programmi di controllo sono stati sviluppati per ottenere una macchina adatta a stampare oggetti dalla forma generica e realizzati attraverso software CAD e tradotti in file STL, il formato solitamente utilizzato dai software di slicing per stampa 3D. Essi sono implementati sia in Matlab®sia nei software nativi di Mitsubishi®in modo da poter stampare un pezzo secondo logiche diverse a seconda delle necessità dei test da realizzare. Si ha quindi la realizzazione di un interprete e di un interpolatore, componenti base di una macchina CNC (computer numerical control), necessari per leggere correttamente l'informazione contenuta in un file G-code ed effettuare la lavorazione del pezzo. Viene presentato un *emulatore CNC* in quanto in grado di fornire queste funzionalità tenendo conto delle limitazioni del sistema di controllo Mitsubishi®, come ad esempio le limitazioni di memoria interna. È stato inoltre sviluppato un controllo manuale per offrire la possibilità di effettuare movimentazioni generiche all'operatore.

Parole chiave: robot a cinematica parallela , linear delta, piatto regolabile, calibrazione, emulatore CNC, sistema di controllo

Chapter 1

INTRODUCTION

In this chapter, the Efesto project, to which this thesis work is related, is presented starting from the description of the Addittive Manufacturing state of art, the description of a generic CNC machine design, the Efesto project itself and its state at the beginning of this work. At the end are listed the main goals of this work.

1.1 Additive Manufacturing

Additive Manufacturing is defined by ASTM F42 (American Society for Testing and Materials) as: "process of joining materials to make objects from three-dimensional (3D) model data, usually layer upon layer, as opposed to subtractive manufacturing methodologies". Although many other definitions exist, the main philosophy of the technology is perfectly embedded.
It was borne in the 1980s, developed by the MIT (Massachusetts Institute of Technology) and sold by 3D Systems that presented the first machine of this kind at the Detroit Autofact show in november 1987 [1].
The success and wide spreading of AM machines is due to affirmation of rapid prototyping during the design process. This kind of machines offers the possibility to sees and touch the developing object as it's being designed. The technology opens up a new field that needs to be charted, since it would be possible to reconstruct damaged goods or directly produce it. For now, it's limited to custom, small size batch (even one single component). If from one side it offers a reduced lead time compared to other prototyping technologies and small engineering time (variation in the CAD file), from the other not all the configuration or object could be realized (although many geometries can not be manufactured with standard technologies).
Now a days, AM machines are widely used also by hobbyist since the diffusion of cheap and/or open source Cartesian plastic FDM machines.
A thorough classification of the different techniques adopting the AM could be harsh to conduct and this is not the intention of this chapter, if interested refer to a text in the bibliography. Here only a brief overview of the AM universe will be presented according to process.

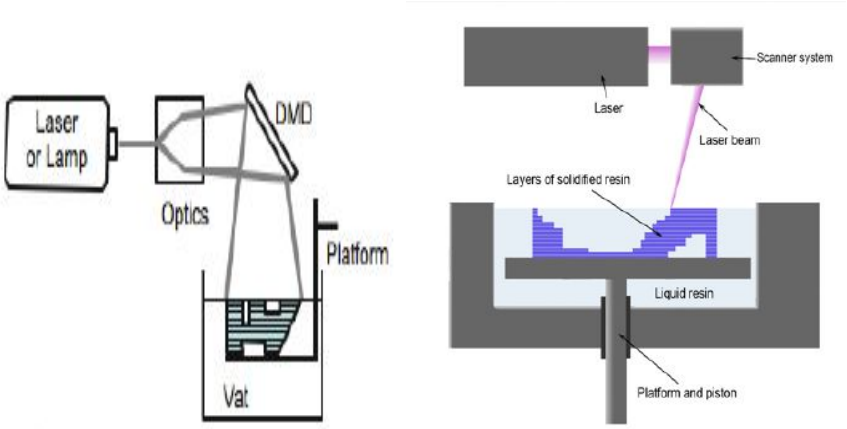


Figure 1.1: Stereolithography with and without DMD

Stereolithography:

It was the first commercialized AM technique using a liquid material (photopolymers) that undergoes a chemical transformation upon interaction with UV light. A platform is placed inside a tank containing the liquid polymer that determines the new layer height. The process begins with the adjusting of the bed in order that a thin film between it and the atmosphere is created. Then a laser system is used to solidify only the interested partition of the film. Completed a layer, the platforms moves to create a new film upon the one just induced. Since standard laser optics interacts with the polymer only in one point at a time making the process quite lasting, DMD (digital micromirror device) has proven to realize a single layer simultaneously (mask projection approach).

Selective Laser Sintering and S.L. Melting:

It's a technique that uses high power laser ($C0_2$ laser) to fuse small particles of plastic, glass or metal. SLM uses even higher power laser resulting in a better final result requiring less post printing operation, but the process is more demanding, errors could leads to residual stresses and strains. The printing procedure is not that much different from stereolithography as can be seen in the Figure 1.2.

Fused deposition modelling (FDM):

FDM was invented and sold by Stratasys. It's an extrusion-based process in which the material contained in a reservoir is forced out through a nozzle when pressure is applied creating the final object. This is the technology behind the majority of the 3d printers available for private users since they do not require laser system or particular operating environments. The material available are mainly polymeric (ABS and PLA).

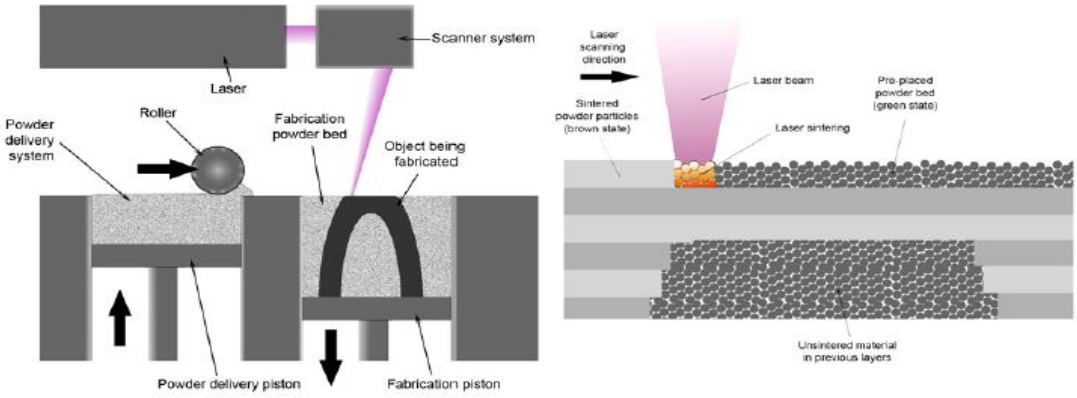


Figure 1.2: Selective Laser Sintering and S.L. Melting

This technique requires the material to reach a temperature a little higher than fusion at the nozzle that solidifies once in contact with air.

AM works with different types of material as it can be deduced from the previous descriptions. Here are the main material characteristics:

polymers: these materials are made up of molecules that repeats the same structural chain. The main ones used are nylon, photopolymers, ABS, PLA. Nylon is used especially with laser based techniques for its melting and joining properties. ABS is the most spread material for FDM. Photopolymers reacts if threated with laser with specific wave lengths.

ceramics: they can be printed by means of binder collecting their particles and, in a next phase, it's removed usually by heat. AM proves to be more affective with these materials compared to standard technologies, due to their brittle and hard nature. Direct printing has been tried, but the procedure is limited by their high melting temperature.

metals: the main metallic materials used in 3D printers are stainless steels, aluminum, Cr-Co and titanium. Particle size around 10-50 μm. They could be directly or indirectly printed. Direct methods are based on melting the particles to obtain the final object by means of laser or EBM (Electron Beam Melting); the final objects are characterized by high mechanical properties. Indirect methods relays on the partial melting of the metal particles or of the binder (polymer), they requires post processing usually to remove the binder itself. It's also possible to avoid melting one of the two parts like in the SLA where the binder is previously bond with the metallic powder.

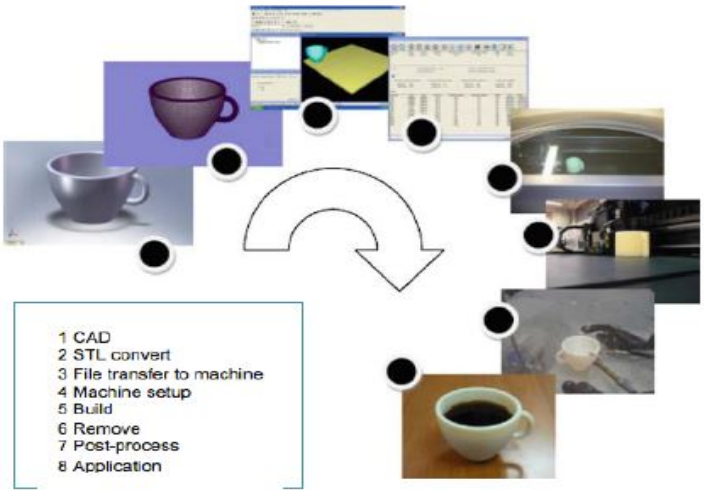


Figure 1.3: AM process

AM process:

The AM process involves several steps (Figure 1.3) [2]:

1. **CAD**:
   The first step is the realization of a software model that fully describes the geometry. Any CAD solid modeling software can be used, but the output must be a 3D solid or a surface representation. Reverse engineering equipment (e.g., laser and optical scanning) can also be used to create such representation.

2. **Conversion to STL**:
   Every slicing software accepts the STL file format, which has become a de facto standard, and nowadays nearly every CAD system can output such file format. It describes the external closed surfaces of the original CAD model and forms the basis for calculation of the slices.

3. **Transfer to AM Machine and STL File Manipulation**:
   The STL file must be processed by a slicing software before being transferred to the AM machine. Here, there may be some general manipulation of the file so that it is the correct size, position, and orientation for building.

4. **Machine Setup**:
   The AM machine must be properly set up prior to the build process. Slicing software requires all these boundaries in order to generated the correct *G-code* that the AM machine will use during build.

5. **Build**:
   Building the part is mainly an automated process and the machine can largely carry on without supervision. Only superficial monitoring of the machine needs

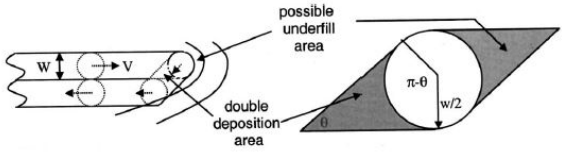


Figure 1.4: Geometrical precision

to take place at this time to ensure no errors have taken place like running out of material, power or software glitches, etc.

6. **Removal**:
   Once the AM machine has completed the build, the parts must be removed. This may require interaction with the machine, which may have safety interlocks to ensure for example that the operating temperatures are sufficiently low or that there are no actively moving parts.

7. **Post-processing**:
   Once removed from the machine, parts may require an amount of additional cleaning up before they are ready for use. Parts may be weak or present supporting features that must be removed. This often requires time and careful, experienced manual intervention.

FDM quality factor parameters:

It's clear that exist several parameters that affect the superficial quality, mechanical resistance and geometrical precision [4]. Defects inside the layers are impossible to remove in post-processing, thus the realization process is very important. Parameters, that condition the superficial roughness, are: layer thickness, material property, temperature, nozzle size and other process parameters. Mechanical resistance is due to the filling pattern, temperature profile and material characteristics. Geometrical precision depends on which case the material is deposited during the TCP trajectory (overfill or underfill (see Figure 1.4)) or expansion or shrinking that the filament might undergo during extrusion and settlement on the plate.

Two filling philosophies exist and they are contour-parallel tool path and direction-parallel (DP) tool path [5]. The first one deposits a series of profiles running parallel to the edge obtained by the slicer. Although gives good results, the algorithm behind it is complicated and layer shape worsen for particular geometries (multi-hollow object). The second is the most used. DP requires the definition of the inclination angle of the reference segment to determine the trajectory made up of parallel segments connected with small curves. This leads to the impossibility to adopt a continuous deposition approach. Another issue is the deformation the materials undergoes due to one direction deposition.

Additional information on AM, if required, will be presented throughout this thesis only for clearness sake.

1.2 CNC machine

For this section, please refer to [6].
Commercial 3D printers usually are CNC machine. Computerized numerical control was developed for controlling the tool trajectory in multi-axis machines especially in "cutting" operation (i.e. milling, turning operation) with great precision. CNC to be exploited requires the generation of CAD model, by means of relative software, that is then passed to a CAM software that generates the tool paths.
CNC is characterized mainly by the NCK (Numerical Control Kernel). This has two main tasks, to read the file containing the trajectory (interpreter) and generate the data of the motion (interpolation).

interpreter: it's the core operation of the system because it has to correctly read and understand the information stored in a file written with a given CNC language (i.e. G-code). Several languages with different characteristic (flavors) exist, although standardization allows a general unification of the language. The interpreter has to be defined according to the CAM generated file, usually taking the majority of the design of the system.

interpolator: it's responsible for the generation of the axis movement from data acquired by the interpreter and its precision reflects on the final object. Only the data sampled interpolator for lines will be here discussed. Defined a starting position (x0,y0) and an ending position (x1,y1), the total path (L) is:

$$L = \sqrt{(x1 - x0)^2 + (y1 - y0)^2}$$

If now a sampling time (T_s) is defined and also a feed override for the velocity, these equations are obtained in [6] :

$$V = Vo \times feedoverride \;\; ; \;\; \triangle L = VT_s \;\; ; \;\; N = int(L/\triangle L)$$

$$\triangle x = \triangle L(x1 - x0)/L \qquad \triangle y = \triangle L(y1 - y0)/L$$

$$x_{i+1} = x_i + \triangle x \qquad y_{i+1} = y_i + \triangle y$$

The values of the incremental displacement are transfered to a FIFO (First In First Out) buffer where they are used to assign the right acceleration-deceleration (AD) profile. If N is not an integer, several ways are available to include the residual (r) displacement such as split it evenly in the first r samples or simply add it to the final movement.
Any interpolation method can be modified to occur after the AD allowing better performances (exact execution of the path) knowing the acceleration and deceleration timing based on the commanded feedrate, residual displacement and allowable AD value and current velocity. This requires more computational cost.

1.3 EFESTO project

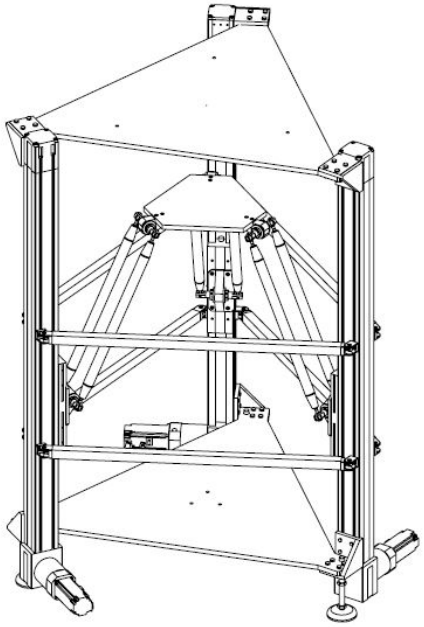


Figure 1.5: first draft of the PKM

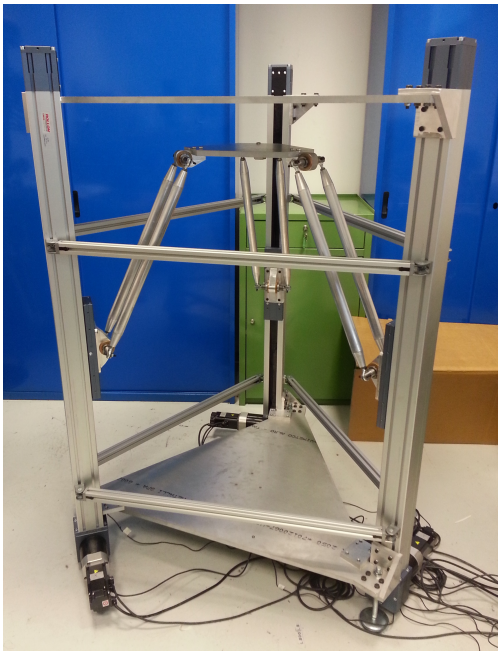


Figure 1.6: initial PKM realization

The EFeSTO[1] (Extrusion of Feedstock for the manufacturing of Sintered Tiny Objects on a parallel kinematics table) project begins in the mechanical department of Politecnico di Milano in September 2014. Its main objective is to design, build and control a 3D printer aimed to test and study a new AM technology based on the already existing Metal Injection Molding (MIM). This printing technique relies on the extrusion of metallic powder combined with a binder, that in post-processing the built object requires to undergo a sintering process to remove the binder itself. This technique is new and strives to create metallic items in opposition to alternative AM technologies based on laser that are more expensive.
The AM technology consist in the deposition of the binary component filament on a moving plate extruded from a fixed position (FDM). The extrusion system weights around 25 kg and it's made up by an extruder, realized by Babyplast® and designed for injection molding (adapted to the project with motor and nozzle), water cooling system, pneumatic circuit and hopper. Due to its bulkiness the extrusion system is the one rigidly connected to the ground while the plate moves, unlike commercial 3D printers with parallel kinematics, otherwise the system would require higher power at the motors and larger dead volume (leading to higher costs) with less precision.
It has been decided to opt for a PKM (Parallel Kinematics Machine) to increase rigidity, precision and robot dynamics [3]; among those robots the combination of two was

[1] Efesto is the mythological god of fire, technology and metallurgy

selected: a Linear Delta (3 translational d.o.f.) and Agile Eye (2 spherical d.o.f). This second PKM will offer the possibility to incline the printing plate avoiding the realization of supporting structures and reduce the step shape profile of traditional slicing. At this stage the second robot has not been realized.
The main task is thus the starting up of a testing lab to improve this technique.

1.4 Starting point of the thesis work

1.4.1 Linear Delta robot [7] [8]

PKMs are robots with two plates (called bases) one moving and one grounded connected by means of links. The relative movement between a base and a link is obtained by joints guaranteeing at least one relative movement. Among the possible PKMs, the one selected was the Linear Delta Robot a variation on the Delta Robot.
A Delta robot has three kinematics chains of the type Revolute-Revolute-Parallelogram-Revolute (where only the first one is active) allowing the moving base three transitional D.o.F.; the Linear version of it sees a prismatic joint in place of the first spherical joint. This is a robot of high dynamic performances and has a vertical disposition of the prismatic joints. This leads to these advantages: compact and ergonomic solution, the only vertical limitation is the travel of the rails and the machine can be symmetric (giving isotropy to the robot).

1.4.2 Motor-transmission system [8]

The system uses a *Mitsubishi® HG-KR43B* electric motor combined with *Bonfiglioli® TR 080 1 10 LOW 50C1 CD14 S5 OR SB KE* reducer moving the *Rollon® ELM 80-SP* guide. See Appendix A.

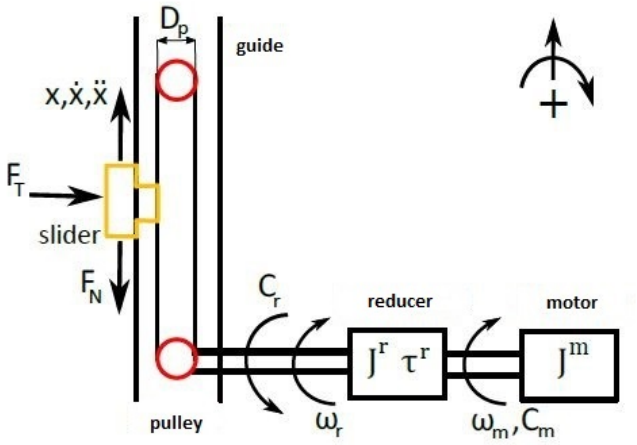


Figure 1.7: motor-transmission system

1.4.3 realization [8]

Material:

The designed component are made of Aluminum alloy 6012, known as Anticorodal. It's light weight, corrosion proof and bears small loads.

σ_R[MPa]	σRp0,2[MPa]	E[GPa]	A% [2]	ρ[g=cm3]	HB [3]	T_f
350	320	69	10	2,75	105	580-650

Joints:

Here are the main characteristics of the Linear Delta joints:
Prismatic (active): *Rollon® ELM 80-SP*
Universal (passive): (see Figure 1.8) bushing *SKF® PBMF 253516 M1G1* - spherical joint end *SKF® SA 8 E*

Figure 1.8: universal joint

Links:

Links are made of two parts. The central one is a hollow tube of 30 mm in diameter, inner diameter of 23 mm, 445 mm long. The end piece is a tapered tube featuring a threaded hole for the spherical joint and a connecting rod for the first piece. The final result can be seen in Figure 1.9.

1.4.4 Extrusion system

Here only the interested aspects of the extruder will be presented, for any other additional information consult the reference thesis paper [9] . MIM is a technique characterized by the extrusion of a metal powder mixed with a binder, usually a polymer. Typically this technique is used for the creation of parts with complex geometry; they require a subsequent step called debinding where the binder is removed leaving a piece with a metal matrix.

[2] cylindrical specimen with diameter 12.5 mm
[3] 10mm sphere loaded for 30 second with 500kg

Figure 1.9: link

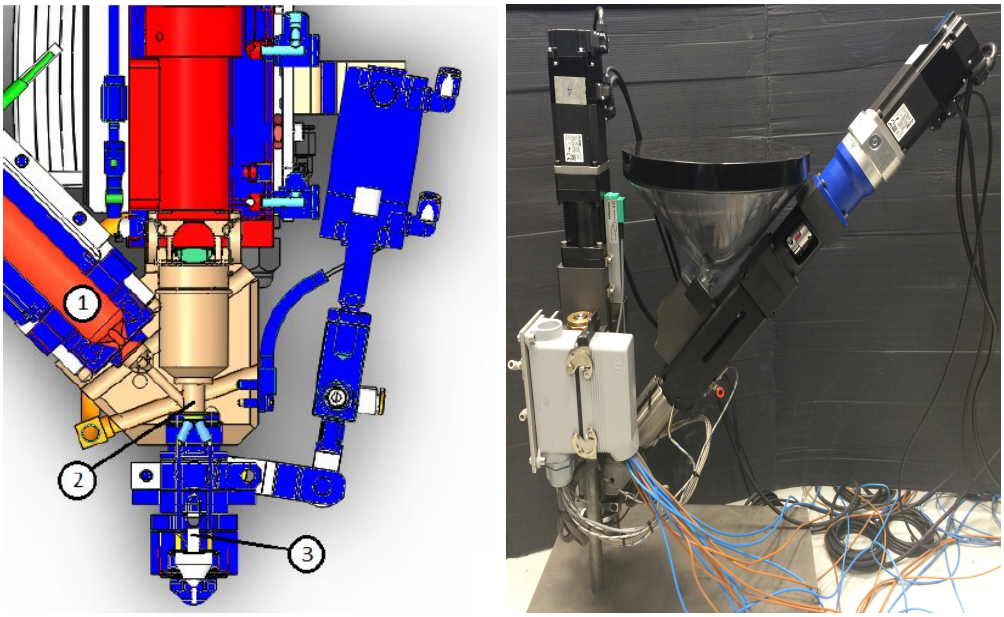


Figure 1.10: extrusion subsystem; (1) plasticizing chamber, (2) extrusion chamber, (3) blocking system

In figure 1.10, the main features of the extrusion system can be seen. The system is made up of a plasticizing and extrusion chambers and a nozzle, all warmed up by resistors. Temperature is achieved by means of PID control logic closing the feedback loop on the temperature by means of thermocouples.
It's advised to make sure that the extrusion punch is at the end, emptying the chamber in order to allow the material to fill it. During the moving procedure from one chamber to the other, the piston is pushed toward the actuator that is off-line by the pressure generated.
Once the heating system is on, it's mandatory that the cooling one is functioning as well to avoid damaging the equipment. The pneumatic piston is used to close the nozzle canal during the filling procedure. The plasticizing task is left to spheres and to the temperature so a good combination of the two should be selected.

1.4.5 Control unit [8][10]

This 3D printer has a control system purchased from Mitsubishi Electric. The main components are:

- PLC: (Programmable Logic Controller), it's a control unit based on a microprocessor used to control the activity of the auxiliaries and coordinate the processes. It's made up of a CPU with memory and an I/O (Input and Output) interfaces allowing it to communicate with other components of the system. PLC cyclically reads the code lines (cycle time of milliseconds) storing the input value (true or false) and updating the outputs. Among the many languages, the Ladder Diagram language is used.
- Motion Control Unit: it's based on a CPU and a memory; here the real time control is carried out. Thus it operates at the lower levels of the control system and it must complete all the operation in the specified time interval.
- Servo drives: they are integrated with close loop control and responsible to the powering of their respective motors (3-phase).

The printer has, at this stage, 5 controlled axis. Two moving the extruder and three actuating the platform. The system uses closed loop control logic. In the extruder the loop is closed on the encoders of the motors, while the ones of the Linear Delta can be also closed on optical sensor on the slider to account for backlash in the transmission chain.
The PLC to motor side connection is bidirectional. Once the trajectory evaluated offline is loaded on the motion control memory, it then uses them to correctly actuate the machine. If the loop is closed on the slider, the reference position is compared to the actual value and the error is used by a PID controller, inside the servo drive. If it's closed on the motor position, the slider position is translated in to an angle by means of the transmission ratio and the diameter of the rail pulley. The current and velocity loop are closed internally by the servo drive to erase the positioning error.

1.4.6 PLC and motion programming [10]

To program the PLC, consult the Mitsubishi® manuals [12][13][14].
The PLC unit can be programmed with the Ladder Diagram (LD). LD is a graphic language derived from the drawing used to control electromechanical relays. It's based on the concept of contacts and coils.
It features two vertical lines, one on the left and one on the right side, the first representing power line while the second the ground. An horizontal line (*rung*) determines the command to run. In fact a rung has on the left a contact (to simplify a switch) and on the right a coil (to simplify a light bulb); if the contact is closed then the current is free to energize the coil. In the LD, this turns the state associated with the coil from 0 to 1.

Figure 1.11: Ladder Diagram example

In the control system of the printer, the contacts are controlled by the HMI (Human machine interface), in this case a touch screen, or by internals relays. Coils are related to the execution of a SFC program inside the Motion. In figure 1.11, an example is shown where M1 is an user input, Y0 an internal state and Y21 is a busy flag to check readiness; if they are close contact, then the right part of the rung is executed (lunches the K1 program and then resets all the state of the contacts).

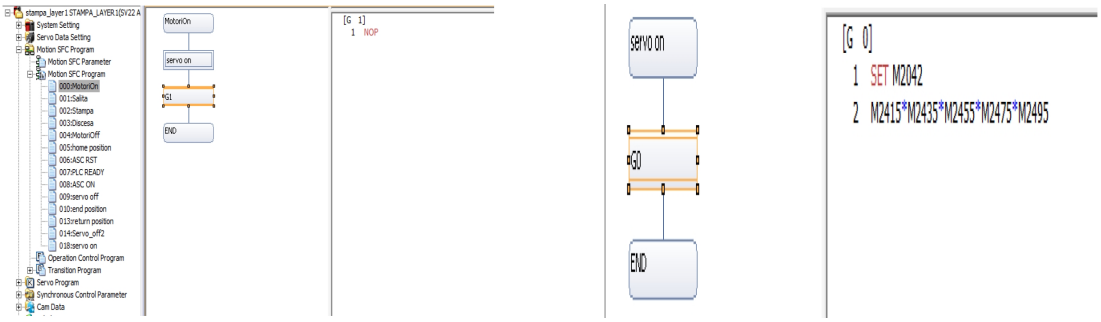


Figure 1.12: Sequential Function Chart (SFC) example

To program the Motion, consult the Mitsubishi® manuals [15][16][17].
The Motion is a real-time controller programmed by a licensed software too. A task is implemented in Sequential Function Chart (SFC) that it's also a graphical language. It's derived from Grafcet[4] used for PLC programming and as a modeling and analytical tool of automated system.
SFC is made up of steps, represent by squares, divided by transitions block where the condition to forward to the next phase is expressed. In Figure 1.12, an example is shown used to turn on the motors. In the first block a subroutine is called that sets the state of the relays associated with the five motors. The transition block is a *NOP*, that simply waits the termination of the previous step.
This controller also allows the usage of the advanced synchronous control. It works on the realization of a series of virtual mechanism and a cam master axis that moves the

[4]functional diagram standardized by UTE (Union Tecnique de l'Electricité) that relies on the concept of State, Transition and Oriented Link

cam slaves associated to an actuated axis.

PLC and Motion share part of their memory to communicate important variables for the control of the system. PLC memory is larger than Motion causing a limitation in the number of points uploaded directly on the Motion, although it's possible to use the memory of the first (that is also expandable) to store data and transfer them little by little to the Motion.

1.4.7 PID tuning [10]

A PID (Proportional plus Integral plus Derivative) control is based on the evaluation of an error function e(t) (or s if in the Laplace domain) by means of direct or indirect measurement of the controlled variable. Once e is known, the control output x(t) is obtained by:

$$x(t) = K_p[e(t) + \frac{1}{T_i}\int_0^t e(\sigma)d\sigma + T_d\frac{de(t)}{dt}]$$

whereK_p ,T_i and T_d are the proportional gain, integral and derivative time constants.
An electric motor requires three concentric loops: the inner is closed on the current (related to the torque T_m), the middle on velocity and the outer on position (they are used to assure execution of the desired trajectory).
The definition of the main parameters of the PID controller can be evaluated directly from the Mitsubishi® software. Here are briefly the auto-tuning procedures:
One-Touch Tuning: assigned an operation to the servo-motor, automatically all the parameters are set. The method only requires to set the response mode (low, basic or high) that is a function of the rigidity of the machine. Used for low oscillation of the moving load.
Auto Tuning: it's obtained from the estimation of the load-motor inertia ratio that can be evaluated automatically (mode 1) by the system from the velocity and current feedback or manually inputted (mode 2). This procedure is repeated every 60 minutes from turning on and the starting estimation are the last one saved in the EEP-ROM (Electrically Erasable Programmable Read-Only Memory) of the servo.
Manual Tuning: last available technique, if both have failed, where the changeable variables, for the position control, are load-motor inertia ratio, motor inertia, position and velocity loop gains and integral action of the latter.
Auto tuning mode 2 is the one that has been used, since mode 1 gave bad results in a trajectory following task. For additional information refer to manuals [18][19].
The PID control of the temperature of the extruder is defined by means of the auto-tuning function, based on the analyzes of the overshoot and undershoot like in the Ziegler-Nichols tuning method.

1.4.8 Specifics [9]

- max. working area: 250x250 [mm];
- max. printing area: 200x200 [mm]
- max. printed weight: 8 [Kg]
- l=0.595 [m] ;Rp=0.198 [m]; s=0.45651 [m] [5]
- C_n^m=1.3 [Nm]; C_{max}^m=4.5 [Nm]; $\omega_{max}^{m,r}$=6000 [rpm]; C_{max}^r=80 [Nm]
- extrusion chamber diameter: 14 [mm]; max plasticizing punch run: 77 [mm].
- nozzle resistor power: 150[W]; central chamber resistor power: 500[W] and plasticizing chamber resistor power: 100[W]

1.5 Thesis objectives

The AM technology proposed in the Efesto project is new, thus it requires a reliable and efficient 3D printing machine to test it. As has been seen in this chapter, a 3D printer is like CNC machine that adds material instead of removing it. Commercial softwares for 3D printing generate a command list in the G-code language from 3D CAD models. In the early stages, testing may require deposition trajectory difficult to obtain with commercial slicers. The final result, as has been seen also in the AM state of the art, depends on different parameters that need to be controlled. Thus the main objectives of this master thesis are:

- Printing from a G-code generated by a slicing software
- Printing from generic custom made trajectories
- Control of the parameters of the machine:
  - velocity of the moving plate
  - extruder feed rate
  - printing temperature

[5] initial values, *l, Rp* and *s* might be changed

Chapter 2

MECHANICAL SYSTEM DEVELOPMENT

2.1 Objective and Overview

One of the main task of the Efesto Project is the testing of a new AM technology. A reliable system is needed in order to carry out those tests thoroughly and precisely. The linear delta has to guarantee accuracy and precision to the deposition plate of the 3D printer. In this chapter the development of a calibration tool and the mechanical improvements needed to make it applicable are presented. This tool is needed in order to achieve the precision required. It is described a generic calibration process by pointing out how it is mandatory on this machine as well as any other robot(2.2). Calibrating tools, as the one here presented, only consider variation in the geometrical parameters, thus it is necessary to eliminate any kind of positioning error due by mechanical defects. The structural changes, that were made to cut down excessive play and to house the extruder while increasing the rigidity of the overall system (the aspects related to the repeatability are addressed), therefore will be discussed (2.3). This is followed by the design of an adjustable plate to assure acceptable normality between printing plate and extruder nozzle axis (2.4). Once contained the phenomena not included in the model, a calibration process is explained that will help to find the most suitable geometrical data to control the Linear Delta Robot (2.5). This requires the study of new equations for the direct and inverse kinematics with geometrical variation within the three links. Then x-z behavior is presented as a function on the geometrical parameters and travel values to give an overview to roughly estimate the kinematic parameters that affect the model. The actual calibrating algorithm based on numerical minimization of the errors is then described.

2.2 Calibration process

The calibration process is defined as the estimation of the errors of a robot model [7]. If a model (inverse and direct kinematics) is created to control a robot, it can be generalized as (its inverse also stands), where Λ are the geometrical terms, while Q the displacements in the joint-space:

$$S = F(\Lambda, Q)$$

If the actual system does not correspond to the model, when a position is aimed, this will happen:

$$S_r = F(\Lambda + \triangle\Lambda, Q_i) \simeq F_i + \frac{\partial F}{\partial \Omega}\Delta\Omega$$

where S_r is the actual position reached by the robot, while Q_i are the joints coordinate evaluated by means of the model. Variations from the theoretical case could also be introduced by the impossibility to actually reach the Q_i evaluated.
The calibration procedure has two main phases to ensure the reduction of such dependencies: measurement of the positioning error in the working space and development of a mathematical technique to overcome those errors.
Two approaches can be found in literature: parametric and non-parametric calibration. The first aims to use kinematic equations as well as measurements to be able to always foretell the behavior of the robot. The steps are: structure analysis, measurement of the errors in few configuration, data analysis and evaluation of the uncertain or unknown parameters. The second one, uses the knowledge of the errors in few points to generate a statistical distribution of the errors. The steps are: measurement, statistical analysis and/or interpolation of positioning error in all the working space, error compensation. The first approach requires the definition of a model capable to distinguish the possible sources of error (usually only geometrical without any consideration on phenomenon that are difficult to describe such as elasticity, play, variable load, etc.). The second approach uses the nominal model, but requires several points to generate good results. The compensation, once again from literature, can be carried out on or off-line, where the on-line case represents the one on the robotic system (after the inverse kinematic). If the compensation is carried out in the joint-space, this can be obtained:

$$Q_c = Q_i - J^{-1} J_\lambda \Delta\Lambda$$

knowing the structural errors ($\Delta\Lambda$), the theoretical inverse kinematics, Jacobian matrix (J) and the Jacobian matrix related to the structural parameters (J_λ). Obtained on the need of a linearized model to describe imperfect robots. The off-line case compensates the errors in the working-space. Using the same notation as before this is obtained:

$$S_c = S_i - J_\lambda \Delta\Lambda$$

that states that, if the positioning error is known, then, the coordinate can be changed in order to make the real system reaching the position intended.

The calibrating procedure could also be improved not only taking into consideration the position reached, but also the measures of the actuated and non-actuated (if it's possible to place redundant sensors) joints.

2.3 Structure modifications

Here are presented additional changes that were made to the structure in order to increase the rigidity of the machine and reduce plays in the kinematic chain that would have affected the correct positioning during operation (due to intentional manual contact with the moving plate to adjust the procedure during the first layers, i.e. cleaning residuals, although should be avoided for safety reason or even accidental contact with the outer frame or vibration from the surrounding environment exciting the first eigenvalues) or repeatability of the built object (play may cause small variation in the inclination of the moving base leading to different projection of the filament on it). A brief inquiry on the realized kinematic chain and how it could affect the control of the robot is firstly discussed.

justification of the calibrating tool:

From the analyses of the assemble report and of the technical drawing (only the main features are illustrated in figure 2.1), it can be seen how the model can significantly deviate from the real system. Each link, despite the numerical value, is made of five elements assembled together. Each components should have been realized with precision down to the $10^{-1}mm$, the real mismatch is expected to have been introduced in the assembling operation. The tapered components are glued to the main tube, that may cause a gap between the two pieces, and the spherical end joint is screwed on the tapered block. If each link has not been assembled with a more than reliable template, the error could be expected in the order of the mm. This variation is more than likely to occur if the closure of the four-bar mechanism is seen; in fact the joint obtained combining the spherical end and the rod by means of a bolt leaves enough room to accommodate different length of the links. Other backlashes throughout the machines allows the system to settle into a new configuration.
This is expected to reflect on the control of the machine since the model is no more a description of the real system.

Backlash reduction:

The Spherical joint *SA 8 E* presents an intrinsic backlash allowing the inner ring to move axially with respect to the outer frame, thus causing an overall oscillation of the moving platform.
To reduce the impact on the machine, cup springs were placed between the joint and its housing causing the two subsystem composing the *SA 8 E* to touch.

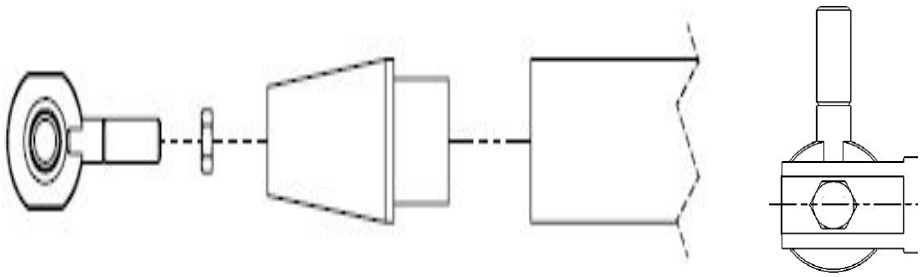


Figure 2.1: main source of variation theoretical-real system

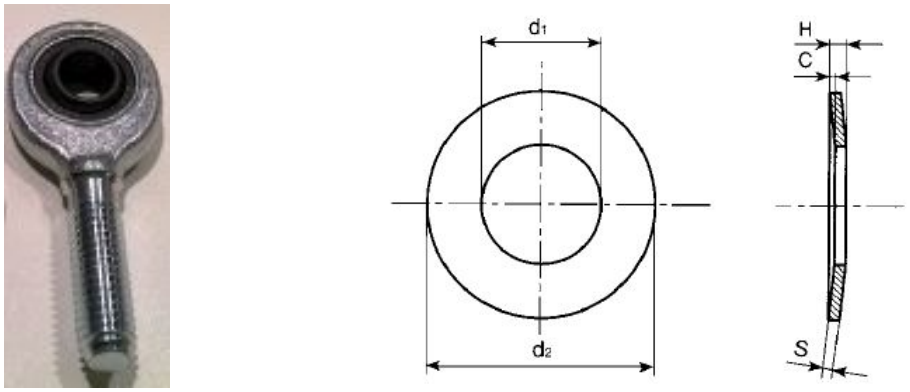


Figure 2.2: Spherical joint and cup spring

Used cup spring specification:

$$d2 = 25\ mm \quad d1 = 12.2\ mm \quad H = 1.6\ mm$$

Extruder housing and outer frame:

The new frame designed[1] to hold the extruder, while increasing the overall rigidity of the machine, can be seen in figure 2.3. Let's start the description beginning from the top part, that is also the most interesting one since it's the feature that allow the transformation of the Linear Delta into a 3D printer.
The extruder has two vertical rods enclosed within a "C" shaped support with one of the jutting plates being welded at the bottom, while the other is tighten after the positioning of the extruder. These two "C" shaped support are welded on an hexagonal squared beam made structure that is fixed on the higher strip after three plate have been placed inside the gap that is generated. This allows the distribution of the load through the strip itself down to vertical guide.
The middle part sees a crossed horizontal band made of five strips joined together and

[1]The design was developed to increase rigidity after some experimental analysis that showed a first frequency at very low frequency ($< 20Hz$). Simulation carried out in previous thesis work showed that this configuration with crossed horizontal strips could significantly increase the first eigenvalue of the system

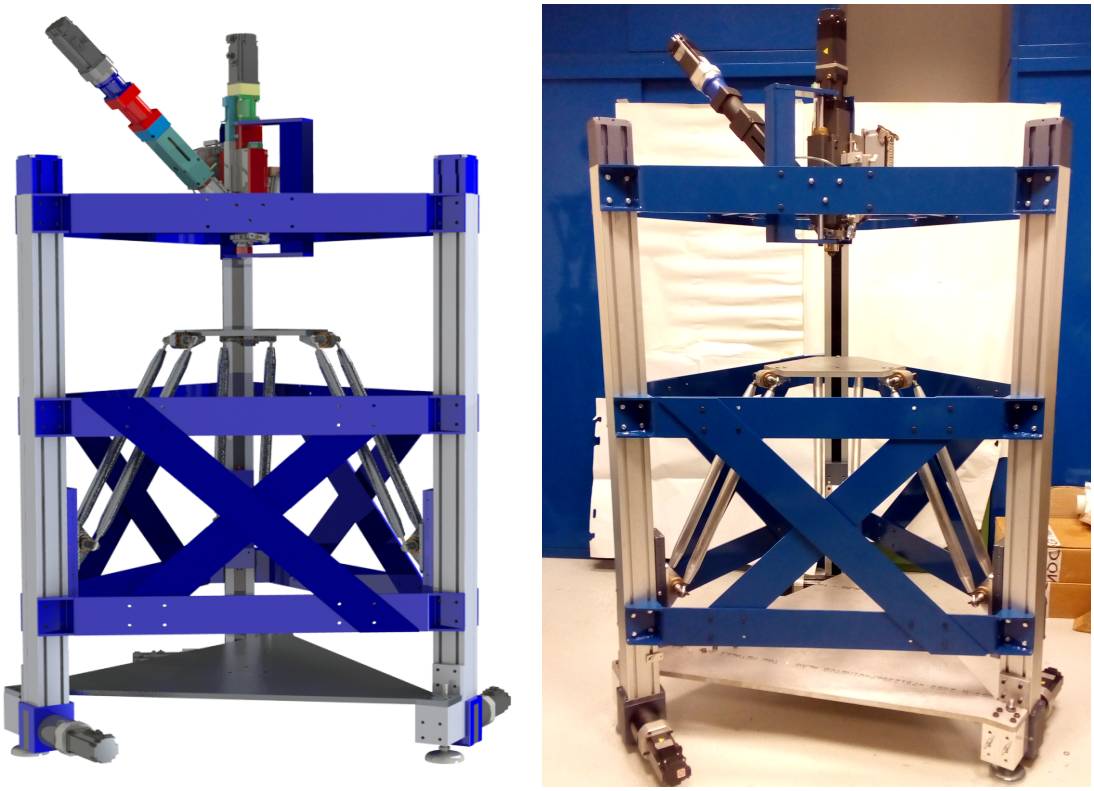


Figure 2.3: final realization of the framework

tighten to the guide.
The lower support was also modified; an almost cubic block is assembled to the vertical rail through a plate. An "L" shaped element is fixed to this component that will allow the printer to be anchored to the ground; also a self leveling base is screwed on the block to adjust the inclination of the entire machine.
Further work on the selection of the screws and bolts for the assembly of the new robot structure can be found in appendix A.7. The sized used are the same that were already assessed for the previous structure.
Here is presented a guideline for the assembling procedure to follow. Start putting together the lower support parts and placing it on the guides by means of a template to assure good positioning. Place the lower plate on a wooden block or anything that will keep it high enough to position under the aluminum block just fixed upon the guide. Screw them on. It's then time to fix the first strip to avoid failure in the sub-assemble. Mount on the rest of the robot before completing the horizontal band with the other strips. Place the last band at the top. The definitive tightening operation must be carried out to assure perpendicularity, where requested in the drawings, between the parts. Place the hexagonal element on the last strip without the top plate. Screw on the extruder initially only on the lower part; then close the "C" shaped holding block with the top piece.

2.4 Adjustable plate

In order to precisely determine the system behavior, it's required to be able to separate variation on the z-axis due to unleveled plate or due to a not precise estimation of the kinematics parameters. Thus an adjustable plate, allowing easy and reliable compensation, seems in order. This step and this equipment is necessary only in case of clearly unbalanced moving base (by means of a level or laser system sampling different spots on it) or uncertainty in the normality between plate and extruder.
Two solutions are presented that can be chosen according to the user preferences or following the comments during the exposition.

2.4.1 Solution A

Solution A requires three, equally spanned from the center, screws placed 120° from one another. The existing plate needs to be drilled and threaded, allowing the screw to move vertically. At the end of each screw, a domed nut is placed in order to guarantee a smooth contact point. The new plate is blocked it the final position by means of two countersunk flat head screws.

Calibration by three screws:

In Figure 2.4 the regulation unit can be seen. For a first draft M6 screws are taken under consideration, since several other part requires this size. The fundamental variables, in order to adjust the level, are described in the following table.
These equations are carried out supposing that d_z, distances needed to solve the problem, are known. h_0 is used to force a required gap between the plates. These equations are obtained considering the center of the new plate fixed along the z-axis and center of all the rotation of the plate itself can undergo. If a different point is considered as reference adjust the value of the parameters accordingly.

Parameters	[mm]		Parameters	[mm]	
$L_{measure}$	75	screw-center gap	h	13	Domed cap nuts height
h0	15	plates gap	d	9.5	Domed cap nuts diameter
p	1	pitch			

$$r_t = d/2; \quad h_t = h - r_t; \quad Z = h0 + d_z;$$

$$alpha = \arctan \frac{d_z}{L_{measure}} \qquad OH = \frac{Z}{\cos(alpha)}$$

$$ls = \frac{Z - r_t}{\cos(alpha) - h_t} \qquad turns = \frac{l_s}{p}$$

where: OH can be seen from the picture and it's the distance between center point of the bolt and the intersect point of the middle line with the lower side of the new plate, ls is the required gap between the domed nut and the old plate.

Tightening by means of two bolt:

One single centered bolt can be used to fasten the structure, but it could lead to the rotation of the new plate resulting in the ruining of the printed object. Thus two seems the least number of screws. One of the main issue related to this subsystem is the necessity to tighten the screw to at least one inclined plate. In order to do so, several techniques can be named:

- washer-spring-washer unit placed between the old plate and the nut or the head of the screw
- binary system made of convex-concave washer (spherical washer) allowing a 3° rotation
- sacrificial washer or plate

One further concern is in which direction to place the bolts. One reasonable solution will be to create two countersunk holes in the new plate, in order to slightly hide them; this solution could prove itself efficiently if a heated bed is placed on top of it to increase adherence during the printing of the first layers.

2.4.2 Solution B

The compensation and the tightening procedure are realized by means of three bolts. This solution can be seen in Figure 2.5 where the head of the screw juts out from the new plate that is interposed between two spherical washer (convex plus concave washer) allowing the free rotation of the new plate. The final tightening of the aforementioned component can be achieved by a nut (only when desired) or a spring (could lead to anticipated blockage). An additional nut can be placed to block the screw position.
A first draft of the system can be seen in Figure 2.6; using some data the maximum allowable rotation until contact (use the lowest one) can be evaluated.

$$h = 10\ mm \quad r_h = 4\ mm \quad r_s = 3\ mm \quad e = 136.25\ mm$$

Where h is the approximated height of the hole (plate plus two spherical screw), r_s is the screw radius, r_h is the hole radius and e is the distance between the center point of the plate and the bolt position.

$$\theta_0 = \arctan\frac{h}{2(e-r_h)} \qquad \theta_1 = \arctan\frac{h}{2(e+r_h)}$$

$$d_0 = \sqrt{(h/2)^2 + (e-r_h)^2} \qquad d_1 = \sqrt{(h/2)^2 + (e+r_h)^2}$$

$$\alpha_{max0} = \theta_0 - \arccos\left(\frac{r_h - r_s + d_0\cos(\theta_0)}{d_0}\right) \quad \alpha_{max1} = \theta_1 - \arccos\left(\frac{-(r_h - r_s - d_1\cos(\theta_1)}{d_1}\right) \tag{2.1}$$

Where d_0 and θ_0 are the distance and the angle between the center point and the inner vertex of the hole (index 1 for the outer) that the center sees, α_i are inner and outer

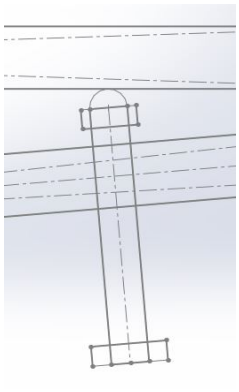


Figure 2.4: solution A

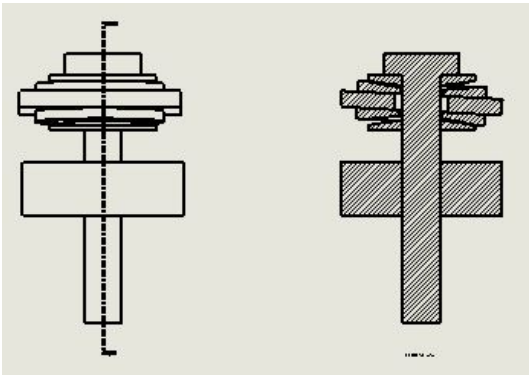


Figure 2.5: solution B

maximum angle that the plate can rotate to (the point of rotation needs to remain fixed in space).

$$\alpha_{max1} = 5.1\,\text{deg}$$

For non centered rotation, but between two bolts placed at a distant of $e = 136.25$ mm from the center, the new rotation distance e and the maximum rotary motion become:

$$e = 236 \qquad \alpha_{max1} = 4.1728\,\text{deg}$$

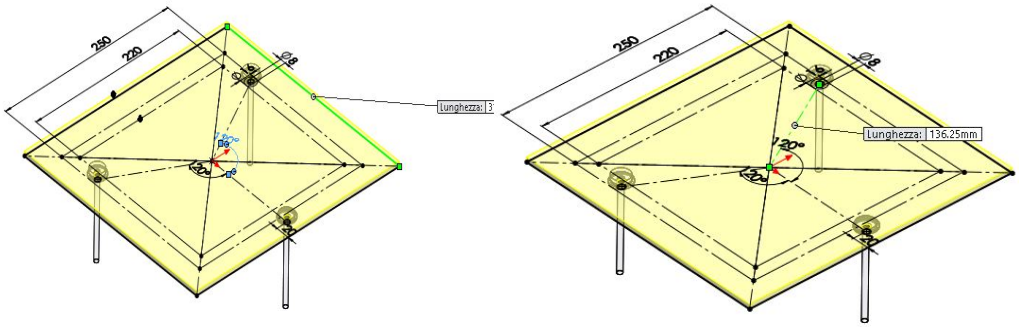


Figure 2.6: first draft to allow the inscription of a 22x22 cm square to accommodate at least a heated bed of printing area of 20x20 cm

2.4.3 Used solution

It has been decided to proceed with the solution B, using M4 screws and a 3mm aluminum plate. Two possible blockage mechanism have been initially proposed and they need to be sorted out.
By using a spring, the system sees two vertical forces, one directed upward due to

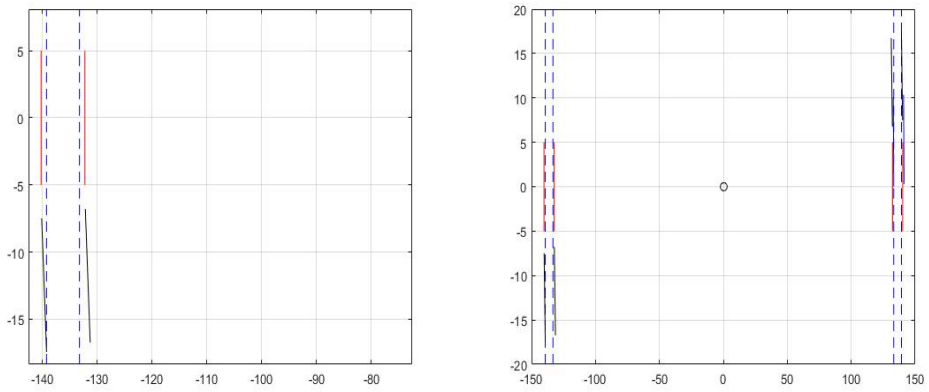


Figure 2.7: graphical solution to the maximum rotation problem for the case with e=136.25 mm

the compression of the spring and one downward due to the weight of the plate. An equilibrium point can be found by using Hook's law, where P_i is a third of the overall plate, k is the spring constant, l_0 is the free length:

$$x_i = l_0 - \frac{P_i}{k}$$

The system will be also subjected to the weight of the printed material, that will lower the equilibrium point. One way to solve this would be preloading the spring (tightening the screw) in a way that, in the working range, it would be seeing the maximum printable load. So substituting the previous weight with the one of the plate itself added by the one of maximum expected objected (also corrected by a 1.2 factor to assure a minimum resultant force directed upward clamping the plate to the screw head) and by imposing a lowering from the equilibrium position of one centimeter (corresponding to maximum correcting displacement), while keeping the length variation under a given percentage to avoid any damage to the spring itself, this equation can be found:

$$\beta l_0 < \frac{1.2P}{k} + h_{max}$$

where β is the percentage representing the maximum deflection acceptable. Solutions available require wider inter-plate gap (see figure 2.8) and an overestimation of the forces acting on the system leading to an increase of the upward resultant force limiting the work done by the spherical washer. Thus the one using a nut as closing and regulating mechanism is used.

In order to place a 22x22 cm square inside the position of the bolts, keeping in mind the equation shown in the outline, this new configuration can be found where the holes are placed at 14 cm distance from the center (see Appendix A).

Solving equation (2.1) for this configuration α_{max1} becomes equal to 3.19° for the center referenced case and to 2.67 ° for the inter-bolts case, considering the new plate of 10 mm thickness instead of 3mm to account to possible realization and modeling error.

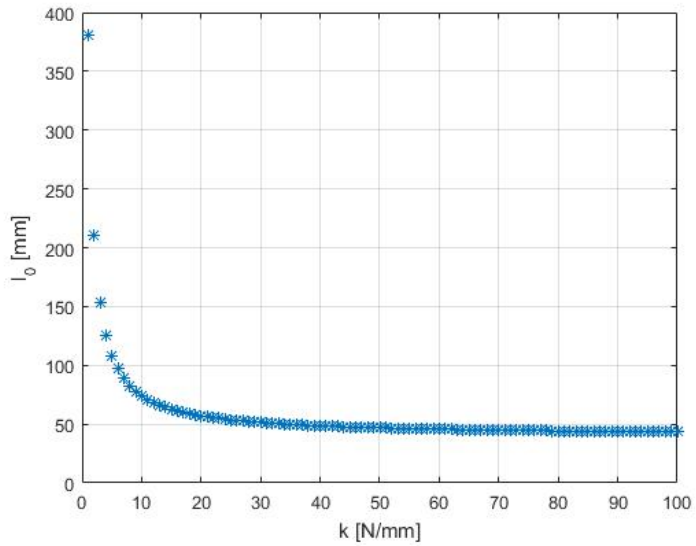


Figure 2.8: variation of the l_0 as a function of k for a 3mm plate with same geometrical and material property of the one already existing, $\beta = 0.3$ and $h_{max} = 10mm$ and a weight of 7 kg acting on a single screw

Bolt sizing and assessment:

The screws used need to be at least 50 mm long since it has to account for the 15 mm aluminum plates (already existing base), 10-14 mm free space for the regulation (the extra four can be used for a second M4 nut or one single M4 locknut), 4 mm for the M4 nut (actually 3.2mm), 0.8 mm for M4 washer, 4.15 mm for the spherical washer, 3mm for the plate, again 4.15mm plus 0.8 mm plus 4mm to close the bolt. It ads up to 49.9 mm.
As for a first verification, let's consider a maximum load of 20 kg [2] acting on the plate and that one single screw is bearing it; the force will only act axially due to the spherical washer. Under the assumption that a threads of the screw-nut coupling are able to withstand the load until it leads to the failure of the bolt itself. Thus:

$$\sigma = \frac{F}{A_{res}} \tag{2.2}$$

where P=mg=196N approximated to 200N; A_{res}=8.78mm^2 for an M4 (it's the tensile-stress area of the screw, although the screw is under compression). A safety coefficient can be found substituting (2.2) in the next equation:

$$\sigma = \frac{\sigma_y}{\eta}$$

[2]corresponding more or less to a 20cm cube made of aluminum alloy 6012, since no information on the printing material density are available at this stage, used also because gives an overestimation of the load compared to the nominal bearing load

where σ_y is the admissible stress, that in this case is considered equal to 160 MPa (lower class 4.6 according to *CNR-UNI 10011*).

$$\eta = \frac{\sigma_y}{\sigma} = \frac{160}{22.8} = 7$$

η is high enough to assure the screw to bear the compressive load.
M4 screws have a 0.7 mm pitch (p), corresponding to around 20 threads in the Aluminum plate (15mm thickness divided by the pitch and roughing accounting for countersink formation during the threading procedure), under the assumption that the load is uniformly distributed along the thickness, each thread will see a load of 10N (although according to the literature [11] the load will be withstood mainly by the first six threads). If the load is supposed to be applied only on the maximum section of the thread[3] and that the pitch is the base of the triangle (the one attached to the plate) then the resisting area can be approximated with [4]:

$$A_{res} = \pi p D = 8.79 mm^2$$

The shear stress for Aluminum 6000 series is around 172 MPa[5].
The safety coefficient, as before can be found:

$$\tau = \frac{P}{A_{res}} = \frac{\tau_{lim}}{\eta} \qquad \eta = \frac{172 \times 8.79}{10} = 151$$

each thread is able to bear the load. If the first one, that bears almost the 40% of the load [11], is tested then the load seen is 80N, the safety coefficient is divided by 8 leading to 18.

Assembling and regulating procedure:

The actual solution differs from the initial presented, the head of the screw is placed below the moving plate and the two spherical washers are closed between two washer-nut system. This configuration allows to adjust the height directly moving the first nut from the 3mm plate tightening with the last nut at the end of the tuning procedure or directly adjusting the gap from the head of the screw. This second solution can prove itself more user friendly since, after losing the last nut, that once tighten is in a desired position (the screw juts out not extensively), allows to correct the height without difficulties (placing correctly the spanner without a clear vision and applying a torque between two narrow metallic components).
Here is presented the assembly required. Screw the M4 completely on the 15mm Aluminum plate. Then place the nut in order to leave around 10mm gap between it and the plate (or leaving enough thread from the nut and the end of the screw in order

[3] it could have been approximated with the force acting on the medium diameter leading to an additional bending stress term.

[4] the actual area for a spiral would have been pL, where $L = 2\pi\sqrt{R^2 + (\frac{p}{2\pi})^2}$, but the additional term under square is neglected since is few order smaller than the other.

[5] average value of shear strength for this series

to assure the desired jut out). A second M4 nut or one single locknut can be used to assure a complete lock (the screw and the nut are required in the second configuration to act as one single system). Place a M4 washer (if required) and a spherical washer. Once all three screws are in position, place the 3mm Aluminum plate. Close the adjustable plate placing a spherical followed by an M4 washer and a nut. The final result can be seen in figure 2.9.
The adjustment should be performed by means of laser system (or any device able to read the correct z-axis position of an object) sampling at least three points on the new plate placed parallel to the old one. The measuring tool should be placed parallel to the nozzle, even better if with an arm of fixed offset able to be easily rotated. Then the correcting heights should be evaluated either way off-line using the equation previously discussed or experimentally while receiving feedback from the measuring device. Moving the machine at this stage is not advised since errors in the model used to control it might affect the final position reached (the in-plane movement with fixed z-axis value can not be assured).

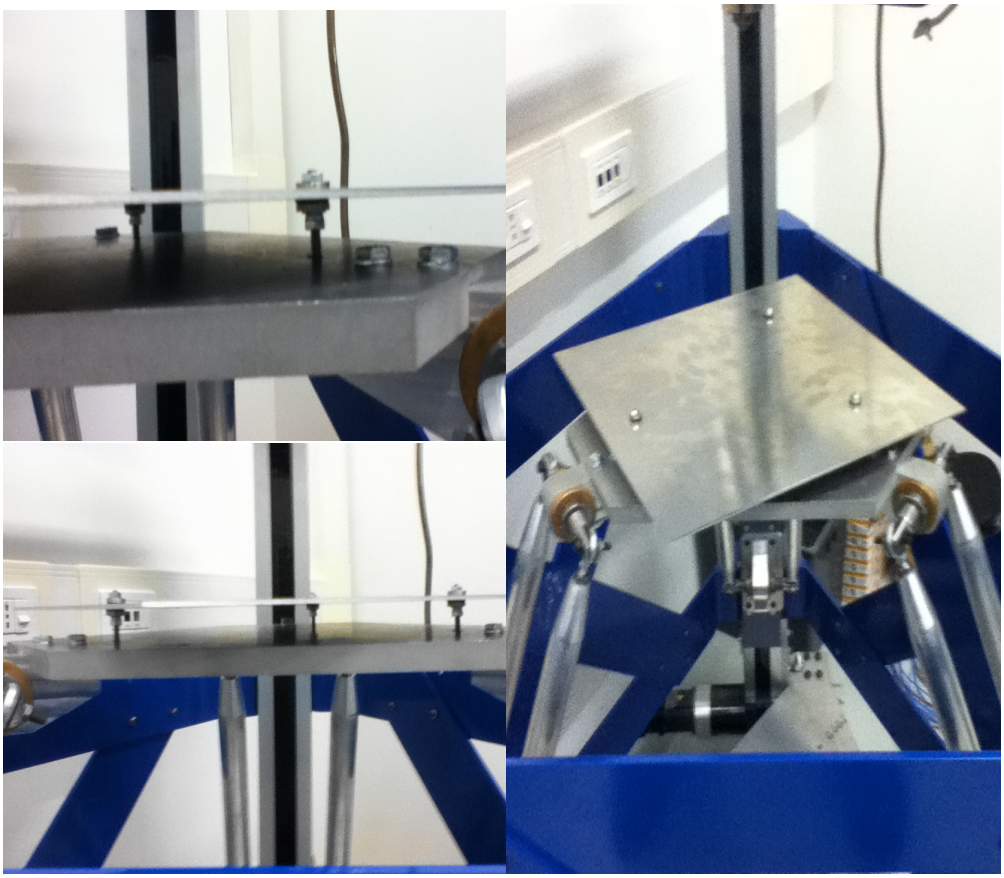


Figure 2.9: adjustable plate realization

2.5 Linear Delta calibration

The calibration process here presented uses a parametric approach and the measurements to estimate the best fitting kinematic parameters. The compensation will be therefore done on these variables and, if deemed still necessary, using one of the two compensation philosophy. Thus the nominal and a more generic definition of the kinematic equations will be presented as well as a brief study on how these parameters affect the system behavior, in order to refine the model taking into account only the actual variables of the system.

2.5.1 Linear Delta Kinematics

The inverse kinematics can be solve straight forwardly, in fact defined the TCP (Tool Center Point), p, b_i and s_i vectors are known. They respectively are the TCP position, spherical joint position referred to the TCP and initial position of the slider. Given these three, the vector d connecting the i link to i slider is found. To allow such TCP position, a q_i displacement of the slider should exist able to guarantee the connection between it and the plate and this should occur for all of the three rail simultaneously. Here are the equations:

$$\underline{d}_i = \underline{p} + \underline{b}_i - \underline{s}_i$$

$$\underline{l}_i = \underline{d}_i - \underline{q}_i \underline{u}_i$$

$$\underline{q}_i = \underline{d}_i^T \underline{u}_i - \sqrt{(\underline{d}_i^T (\underline{u}_i \underline{u}_i^T - [I]) \underline{d}_i + \underline{l}_i^2)} \tag{2.3}$$

Equation (2.3) has no solution when the term under square root is negative, showing the fail of the guaranteeing condition.

Direct kinematics can be solved for a 120° disposition of the linear guides and keeping in mind that R_p and s are the modules of b_i and s_i. For these machines, the solution of this problem can be quiet challenging compared to the one of serial robots where for any given displacement in the joint space a position is reached in the work space. Here a solution in closed form can be found.

$$\underline{d}_i^T \underline{d}_i = \underline{p}_i^T \underline{p}_i + \underline{b}_i^T \underline{b}_i + \underline{s}_i^T \underline{s}_i + 2\underline{p}_i^T \underline{b}_i - 2\underline{p}_i^T \underline{s}_i - 2\underline{b}_i^T \underline{s}_i \tag{2.4}$$

$$l_i^2 = \underline{d}_i^T \underline{d}_i - 2\underline{d}_i^T \hat{u}_i q_i + q_i^2 \tag{2.5}$$

1. substitute (2.4) into (2.5).
2. develop all the scalar products of the equation obtained (obtaining equation A, for clearness sake, as a function of i).
3. $B = 2A(i = 1) - A(i = 2) - A(i = 3)$
4. $C = A(i = 3) - A(i = 2)$

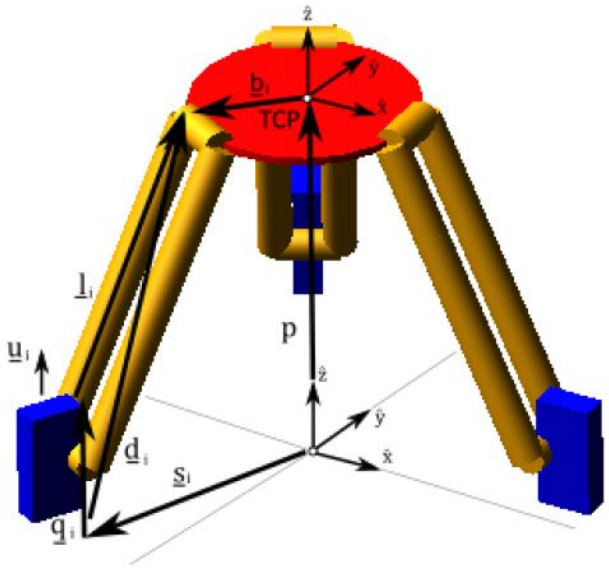


Figure 2.10: Linear Delta model

5. B and C define a linear relationship among p_x, p_y and p_z; substituting them in A(i=1), this can be obtained:

$$k_1 p_z^2 + k_2 p_z + k_3 = 0 \tag{2.6}$$

where:

$$k_1 = \frac{(2q_1 - q_2 - q_3)^2 + 3(q_2 - q_3)^2}{9(R_p - s)^2} + 1$$

$$k_2 = \frac{3(q_2 - q_3)(q_3^2 - q_2^2) - (2q_1 - q_2 - q_3)(2q_1^2 - q_2^2 - q_3^2)}{9(R_p - s)^2} + \frac{2(2q_1 - q_2 - q_3)}{3} - 2q_1$$

$$k_3 = \frac{3(q_2^2 - q_3^2)^2 + (2q_1^2 - q_2^2 - q_3^2)^2}{36(R_p - s)^2} - \frac{(2q_1^2 - q_2^2 - q_3^2)}{3} - l^2 + q_1^2 + (R_b - s)^2$$

6. from (2.6) p_z can be evaluated, substituting it in B and C, p_x and p_y are found.

Variation within the three links case:

In this section, the new kinematics equations are derived to deal with unexpected behavior of the machine that strictly relays on this, such as assembly errors and realization imperfection.

Inverse Kinematic:
As concerns the inverse problem, no significant variation tells the two cases apart, only the diversification of the l_i squared.

$$d_i = p + (Rp - s)u_i$$

$$q_i = d_i^T u_i - \sqrt{(d_i^T (u_i u_i^T - [I]) d_i + l_i^2)}$$

If the case with R_p and s varying is required, simply change in d_i the two parameters.

Direct Kinematic:
The direct problem is solved using the same aforementioned procedure carrying on the indexes of the length of each link (using different alphabetic representation).

$$A = p_x^2 + p_y^2 + p_z^2$$

$$B = Rp - s$$

$$A + 2B\cos(\frac{2(i-1)\pi}{3})p_x + 2B\sin(\frac{2(i-1)\pi}{3})p_y - 2q_i p_z + B^2 + q_i^2 - l_i^2 = 0 \tag{2.7}$$

$$p_x = \frac{2Cp_z + D}{6B} \tag{2.8}$$

$$p_y = \frac{2Ep_z + F}{2\sqrt{3}B} \tag{2.9}$$

Equation (2.7) is obtained substituting (2.4) inside (2.5) as a function of the i link. Equation (2.8) and (2.9) are the linear functions of p_z. Here are the definition of the alphabetic symbols.

$$C = 2q_1 - q_2 - q_3$$

$$D = -2q_1^2 + q_2^2 + q_3^2 + 2l_1^2 - l_2^2 - l_3^2$$

$$E = q_2 - q_3$$

$$F = -(q_2^2 - q_3^2) - (l_3^2 - l_2^2)$$

p_z is obtained solving the quadratic equation (2.10).

$$K_1 p_z^2 + K_2 p_z + K_3 = 0 \tag{2.10}$$

$$K_1 = \frac{\frac{C^2}{3} + E^2}{3B^2} + 1 \tag{2.11}$$

$$K_2 = \frac{CD + 3EF}{9B^2} + \frac{2C}{3} - 2q_1 \tag{2.12}$$

$$K_3 = \frac{D^2}{36B^2} + \frac{F^2}{12B^2} + \frac{D}{3} + B^2 + q_1^2 - l_1^2 \tag{2.13}$$

If also R_p and s would have been let change, then p_x would have been a function of both p_y and p_z and vice versa. If the equation presented before do not match the behavior experienced then the resolution of this case should be carried out.
In oder to do so the three equations obtained varying i, from 1 to 3, in (2.7) are put in matrix form.

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}^T \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \begin{bmatrix} 2B_1 & 0 & -2q_1 \\ -B_2 & \sqrt{3}B_2 & -2q_2 \\ -B_3 & -\sqrt{3}B_3 & -2q_3 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} l_1^2 - B_1^2 - q_1^2 \\ l_2^2 - B_2^2 - q_2^2 \\ l_3^2 - B_3^2 - q_3^2 \end{bmatrix}$$

To solve such second order system a symbolic calculation in Matlab was run. The term a,b,c are the B_1, B_2, B_3; q, t, r are the i^{th} (from 1 to 3) of the term: $l_i^2 - B_i^2 - q_i^2$.

$syms\ a\ b\ c\ q1\ q2\ q3\ q\ t\ r\ x\ y\ w\ S$
$A = ones(3);\ C = [q; t; r];$
$B = [2 * a, 0, -2 * q1; -b, b * sqrt(3), -2 * q2; -c, -c * sqrt(3), -2 * q3];$
$xt = [x, y, w]; xn = [x; y; w];$
$S = solve(xt * A * xn + B * xn == C)$
$X(a, b, c, q, t, r, q1, q2, q3) = S.x;\ Y(a, b, c, q, t, r, q1, q2, q3) = S.y;$
$Z(a, b, c, q, t, r, q1, q2, q3) = S.w;$

$double(X(a, b, c, q, t, r, q1, q2, q3))$

The solutions are not presented due to their extended length. One of the additional reason is that, if this is the case, the solution, if required, can not be implemented in the Mitsubishi softwares to check the position of the moving base after several operating cycles avoiding the accumulation of errors. The *double* function is used to convert from the symbolic formulation back to the numeric form.

Jacobian matrix definition:

It's interesting to also see how this deviation from optimality changes the velocity behavior, thus the initial estimation of the resulting velocities required at the motors. Deriving in time the equation (2.5) and with some rearrangements and therms recognition, this equation can be found:

$$l_i^T(\dot{d}_i - \hat{u_i}\dot{q_i}) = 0$$

If the l_i term is expressed as $l_i\hat{n_i}$ and the equation divided by the module l_i, this is obtained:

$$\hat{n_i}^T\hat{u_i}\dot{q_i} - \hat{n_i}^T\dot{p} = 0$$

Considering the three links simultaneously putting them in matrix form and rearranging the terms to explicit the Jacobian matrix, finally it's obtained:

$$[J]^{-1} = [J_q]^{-1}[J_{gs}]^{-1} = diag(1/n_{i,z})\underline{n}^T$$

$$\dot{q} = [J]^{-1}\dot{p}$$

where for the theoretical case the only dependency to any geometrical parameter is:

$$\underline{n} = p[1, 1, 1] - [0; 0; 1]q^T + (R_p - s)\begin{bmatrix} 1 & -0.5 & -0.5 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \\ 0 & 0 & 0 \end{bmatrix}$$

The only case, where variation in the expected parameters leads to unpredicted behavior, is when the $R_p - s$ terms changes. If a diversification is required then the difference of a given link multiplies the respective column ($[R_p - s]^T$).
Variation in these two parameters is expected to be smaller compared to the one of the links that were manually assembled by previous thesis-writers.

In general, all this consideration are valid if the orientation between two adjacent links is of 120°. The discussion from here after is carried out considering that this angle does not change, but if, for same reason, this assumption is overruled (due to the believed imprecision in the realization of the supporting structure), the matrix containing the orientations terms needs to be corrected with the right directors cosines.

2.5.2 X-Z behavior

Here are presented the behavior in the vertical plane (it would have been analogous talking about the y-z plane) of the robot as a function of different parameter that will help refining the model of the system accounting for only those uncertain parameters. If the dimension of geometrical parameters is not specified [m] is intended.

Function of the motion law:

For this section, as well for all the others concerning the behavior of the robot, the results are obtained imposing a given motion law and solving point by point the direct kinematics equations. The parameter c used in the figures is the normalized inversion point of the acceleration diagram.
Figure 2.11 and 2.12 show the increase of height for different point lying on the x-axis. They are obtained using a trapezoidal velocity motion law. This parabolic trend depends on the travel value rather start and end point.
Different motion law gives similar trend where the higher point is reached mid curve, with the same value. It's the only mutual point among all the motion laws (including the linear interpolation between start and end point) (Figure 2.13).
This section can prove to be quite useful during the G-code adaptation or the points assignment. In fact allowing the machine to reach far points in the space, from the starting one, could lead to significant variation in the z-axis leading to bad result or even failure of the machine or printed object (if during first layer, the nozzle will smash against the moving base, damaging the mechanics, or if while executing a mid layer, the nozzle will pierce or impact against the printing).

Function of the kinematics parameters:

Figure 2.14 shows the behavior for three, but equal, links length:

$$p_0 = (X_0, Y_0) = (0, 0)mm \quad p_1 = (X_1, Y_1) = (10, 0)mm$$

$$alpha_i = linear\_inverse\_kinematics1(p_i, L, s, Rb)$$

$$\triangle q = alpha_1 - alpha_0$$

$$p_{1.new} = linear\_direct\_kinematics1(\triangle q, L1, s, Rb)$$

As suspected no significant variation (up to $10^{-1}mm$) is found. If the three lengths are different from one another (Figure 2.15) this difference increases according to the

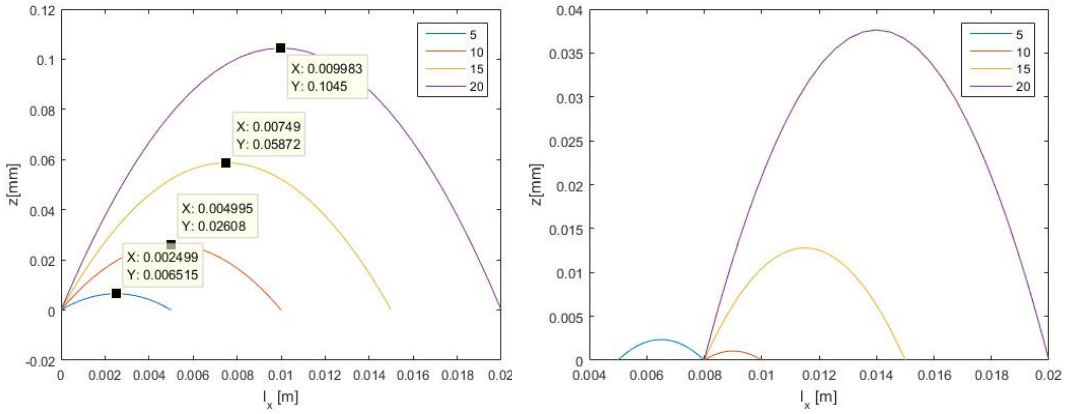


Figure 2.11: (Left)X-Z behavior: dz as a function of l_x given a trapezoidal velocity profile (c=1/3); l=0.595; Rb=0.198; s=0.45651;

Figure 2.12: (Right)X-Z behavior: dz as a function of l_x given a trapezoidal velocity profile(c=1/3) starting from X=8mm; l=0.595; Rb=0.198; s=0.45651;

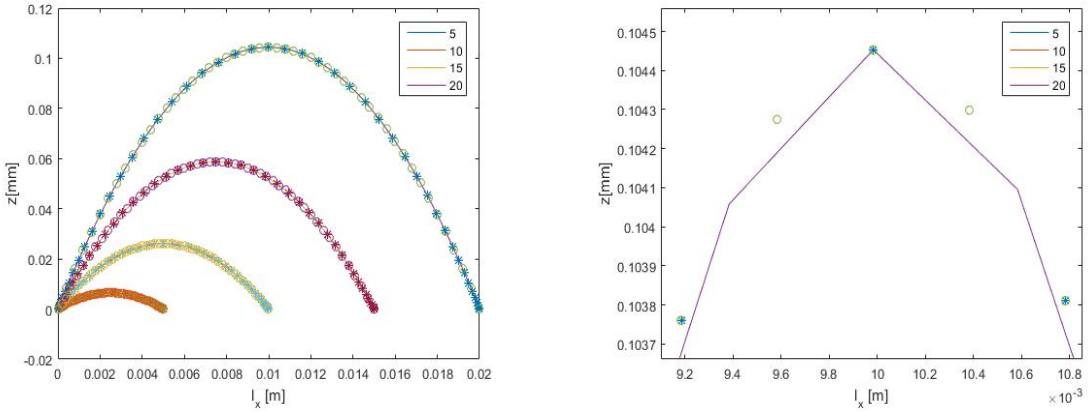
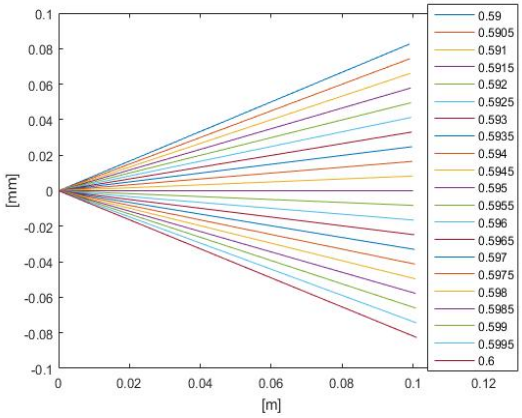


Figure 2.13: X-Z behavior: dz as a function of different motion law. Line is associated with a trapezoidal velocity profile (c=1/3), circle with a constant velocity and asterisk with a cycloidal motion law (c=1/3)

mismatch (the drawing is obtained using different values contained in the matrix M[6] where the columns contain the different cases). The closer to the actual value the lower dz gets. Significant is also a slight variation of the x position causing the robot to miss the target aimed. Now the numerical solution with R_b and s that vary from one chain to the other is considered. It needs to be checked first. In figure 2.16 the result can be seen for same parameters as in the case Figure 2.11 for a 10 mm travel value, the x-z behavior is practically the same while, the x-y has a very small variation in the order

[6] M= $l1$ = 0.5800 0.5844 0.5889 0.5933 0.5978 0.6022 0.6067 0.6111 0.6156 0.6200
$l2$ = 0.5700 0.5744 0.5789 0.5833 0.5878 0.5922 0.5967 0.6011 0.6056 0.6100
$l3$ = 0.5850 0.5861 0.5872 0.5883 0.5894 0.5906 0.5917 0.5928 0.5939 0.5950



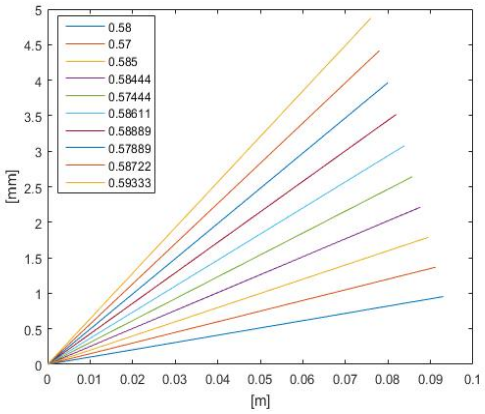
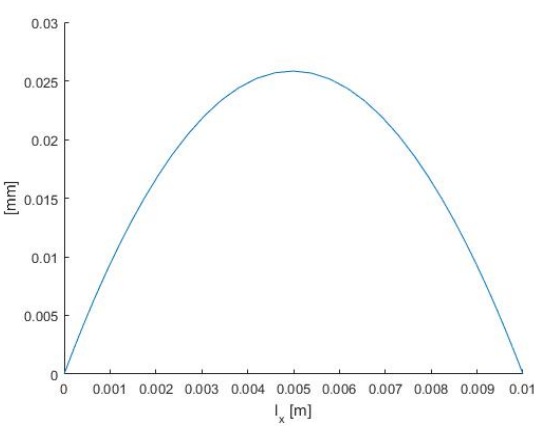


Figure 2.14: (Left) X-Z behavior: dz as a function of different,but equal, real link length given a displacement (X=100mm) evaluated with initial geometric setting

Figure 2.15: (Right)X-Z behavior: dz as a function of random different real link length given a displacement (X=100mm) evaluated with initial geometric setting



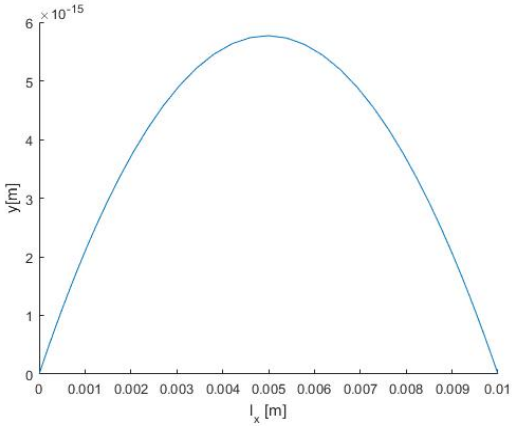


Figure 2.16: X-Z behavior: test of the numerically solved direct kinematic problem as in Figure 2.11

of the 10^{-15} m, that can be neglected. It's now possible to go ahead and check how the robot moves if such parameters are not the same, since the two solutions gave almost the same result[7]. Figure 2.17, 2.18, 2.19 and 2.20[8] show a significant residual error if s_i are slightly different from one another compared to R_b case.

[7]true for z=0 (initial and final position), otherwise the solutions are significantly different. Since the position are fed as relative position to the first one, the movement can be split as a strictly vertical and one in plane movement leading to such approximation

[8]the x-y-z position of the figures has been rigidly moved to start from (0,0,0) position to clearly see the variation in the considered case

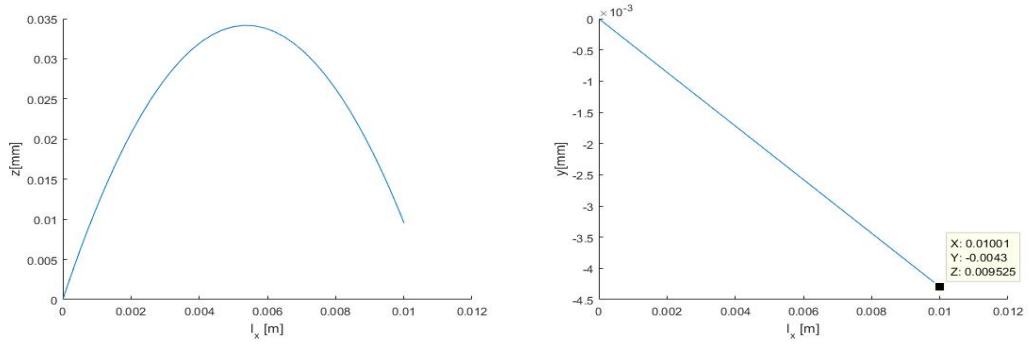


Figure 2.17: X-Z behavior: trajectory followed for s=[0.4561 0.4565 0.4570] other parameters are left as previously; the motion law is trapezoidal velocity profile (c=1/3)

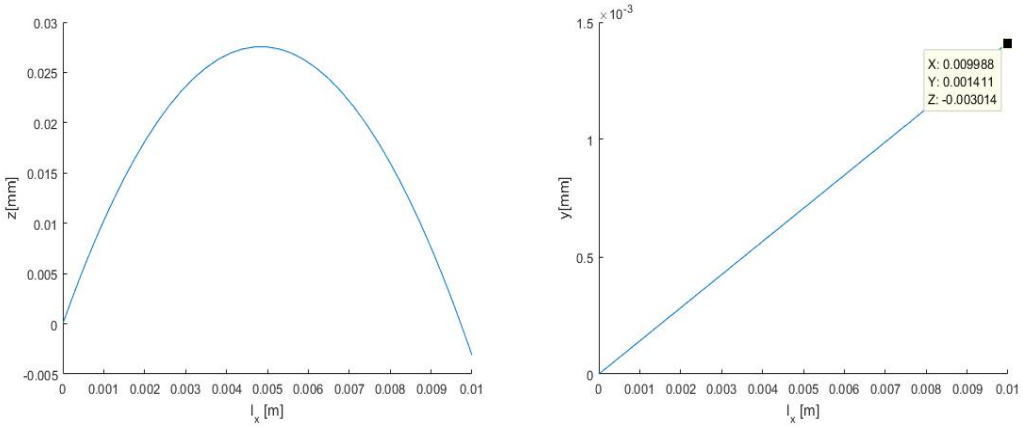


Figure 2.18: X-Z behavior: trajectory followed for R_b=[0.1498 0.1500 0.1501] other parameters are left as standard; the motion law is trapezoidal velocity profile (c=1/3)

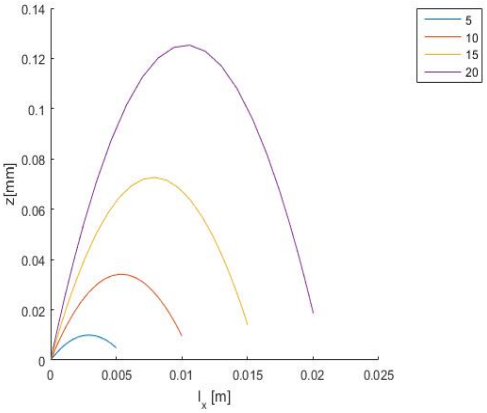


Figure 2.19: X-Z behavior: trajectory followed for s=[0.4561 0.4565 0.4570] other parameters are left as standard; the motion law is trapezoidal velocity profile (c=1/3) and different travel values

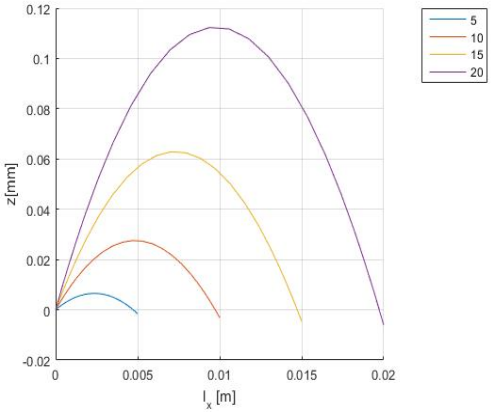


Figure 2.20: X-Z behavior: trajectory followed for R_b=[0.1498 0.1500 0.1501] other parameters are left as standard; the motion law is trapezoidal velocity profile (c=1/3) and different travel values

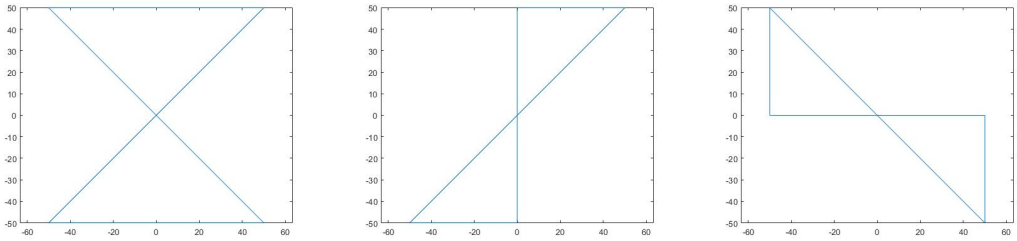


Figure 2.21: three different pseudo eight path examples

2.5.3 Calibrating tool

Once the plate has been leveled properly, a new test needs to be run to conclusively determine the causes of errors in the model.
In the real system, the three rail displacements are fed feed-forwardly with no knowledge of where the end effector will be. The algorithm allows us to simulate this behavior because at every iteration a new pseudo real system is created until it matches the one searched. The break condition is the reaching of the minimum error tolerance between measure and simulated data($\|E\|_\infty < tol$). Since a redundancy of points are expected to be available, the tolerance might be applied to the norm of the vector of errors ($\|E\|_2 < tol$ as well the infinity norm).

Here are exposed the main points to follow to address the misbehaviors due to an erroneous estimation of the Linear Delta parameters:

1. execute tuning drawing (ex. 1 turn square or pseudo "8" path or simple line)
2. point by point inquiry or measure x_{max}, y_{max} and z_{max} (for pseudo "8" also $_{min}$)
3. execute the respective Matlab® script
4. rerun the calibrating drawing with new geometrical parameters
5. check results.

This algorithm is based on the assumption that initial values could prove themselves effective. If this is not the case, an expedient is in order to determine the correct values. According to the results obtained, several options arise. If the references and actual values differ, a calibration procedure is in order, otherwise the machine is already set to operate. Established that additional work is needed, two cases need to be differentiated: x_{max} and y_{max} are or not equal to each other. If their are equal than run the tuning script only changing R_b and s parameters that will cause a dilatation or contraption of the sketch; if this proves, after a new test, ineffective try also changing the length of the link (but keeping them equal; no new kinematics equations are required). If x_{max} and y_{max} differ one another (with different discards), then the tuning script requires to iteratively change every combination of length links until the most suitable solution is found.

Repeat these steps until a desired result is reached. If no significant improvement is achieved then it may also be tried the case with different l, R_b and s, but, due to its nature, it's computationally expensive leading to very high waiting times compared to the analytical solutions. The scripts for these cases are no presented due to their specific nature; they clearly depend on the data type and shape the instrument will output.
A first measure set is required to increase the accuracy of the estimation of the kinematic parameters. Once solved this issue, a second set can be used to estimate the residual error, but this time used them to correct the target position exploiting either the on or off-line compensation (the choice should be made considering experimental test, since the correction depend on the nominal definition of the kinematic problem). The error distribution can be in first attempt approximated as a second order equation in function of x-y displacements (same order of the kinematic equations), while with no z-axis value dependency: $e(x,y)_i = k_1 x_i^2 + k_2 xy + k_3 y_i^2 + k_4 x_i + k_5 y_i + k_6$. Supposing to have obtained more than six measurements (one every dL for any pattern can be obtained), then the equation becomes: $\underline{E}(x,y,z) = A(p_i)\underline{K}$, where p_i is the vector of the in x-y plane coordinates of the target position. Only a least-square solution can be found using the left-pseudo inverse of A: $\underline{K} = (A^T A)^{-1} A^T \underline{E}$. Thus known the target position, an off-line compensation can be set finding the error value and subtracting it to the aimed configuration. The procedure here described has to be adjusted to better fit the considered system, therefore the dependency of the error of also the z direction, the selection of a suitable polynomial order approximation and the need to use a recursive evaluation of the error, since it's approximated as constant in the neighborhood of the target position while it should be evaluated with the one that is commissioned knowing where it will end up .

Measuring:

Run a tuning program such as a one turn square or a pseudo eight path with known geometry. The points laying on the trajectory can be measured by means of a laser tracking device[9] placed on the extruder support or accordingly to the specific device used. These family of instruments are based on the positioning of a sensing/reflecting target placed on the machine and the measuring of angles and distance required to hit the sensor (either external or internal to the tracker).
The space tracking instrument could be used few times during the main calibration process to increase the performance of the machine or, if the cost related is financially acceptable, permanently placed on the machine to execute the compensation automatically every starting up or every unspecified number of hours. This could compensate

[9]For low budget, hobbyist application of this calibrating tool, place a piece of paper on the robot and a drawing mechanism on the extruder (a pen with a pen-holder seems a reasonable solution for early analysis). Use the sketch on the sheet to find out the x_{max}, y_{max}. If the drawing mechanism during the test is incapable to perform its task due to detach or forced to move upward, please make sure to also measure z_{max}. If this is the case adjust the script to the projection on the plate.

for variation in the system due to wear or other phenomena happening during several working hours. This instrument could be used also to calibrate the machine every switching on to find the z-axis home position of the 3D printer (the nozzle-plate contact level), by placing a sensing element also near the extruder, with all the problematics that this arises (temperature related).

Script comment:

For the actual code consult the appendix B[10].
Import the *.csv* file containing the data fed to the Mitsubishi® software. Then the specific information related with the Motion software is discarded and only the important data is carried on. After *LAYER=1;* the information is restored in such a way that the kinematics functions can be run (for the possible cases please refer to the programming chapter since it's related to the data format chosen). Then the reference and obtained values are stated; a for cycle is used to end up back again with the end effector position record. Results are plotted to confirm the import procedure.
This part of the code can be used to easily change Rp or $\mathbf{s}$ (even $\underline{l}$) and forwardly see how the result adjust. If a proper set of value is required to be obtained automatically, than run the second part.
The second part of the script starts with the setting up of all the data that define the recursive procedure that determines the geometrical data. *low* and *up* store the coefficient to be multiplied by the reference lengths, while *dt* the increment between to consecutive measures points. The user can comment this part and manually introduced a proper inquiry array. *tol* and *tolz* contains the tolerances that breaks the for-cycles; the higher these values the lower the precision of the result. These parameters need to be set in order to allow the iterative process to converge to at least one solution (in fact higher increment or a too close range combined with restrictive tolerances could lead to a negative outcome).
This version of the code is based on the variation of $\underline{l}$, Rp and s, if reputed unnecessary remove them from the code (it will also reduce the computational cost, and this step could delayed to a next tuning test). The vector $\underline{l}$ is obtained by fixing the first term, varying the second and for each of the middle term set a new third. Then the kinematic problem is solved and discrepancies between real and obtained case are evaluated. Only the in-range values are stored (change the data file name in consecutive run). The one part of the code that studies the errors of the all in-range configuration is omitted due to it's immediate realization.
Figure 2.22 and 2.23 show the final result of a simulation considering a maximum x-y-z displacement as if it would have been measured; it's interesting to notice the strong relationship with the z-axis behavior (in the figures is in [m]). Change in Rp or s value also modify the vertical displacement as can be seen in figure 2.24 and 2.25.

[10] A variation of the script (not presented in the appendix) is based on the trace left on the plate and not the relative x,y,z position, that can be required if dealing with trace based inquiry.

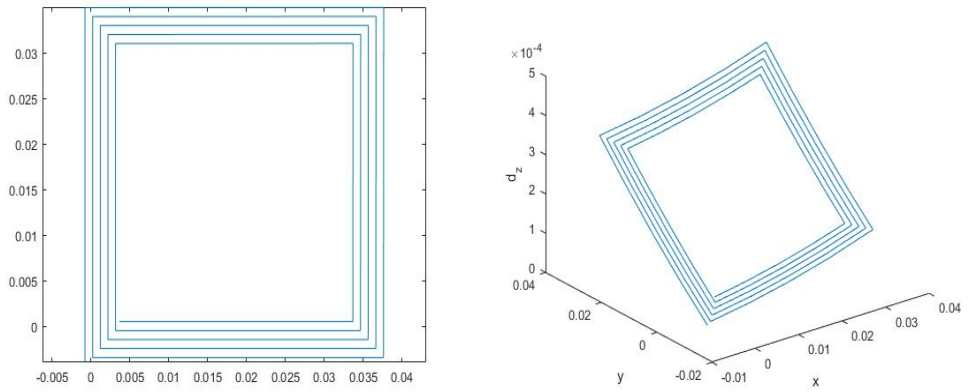


Figure 2.22: final result of tuning process simulation in the x-y plane and in space

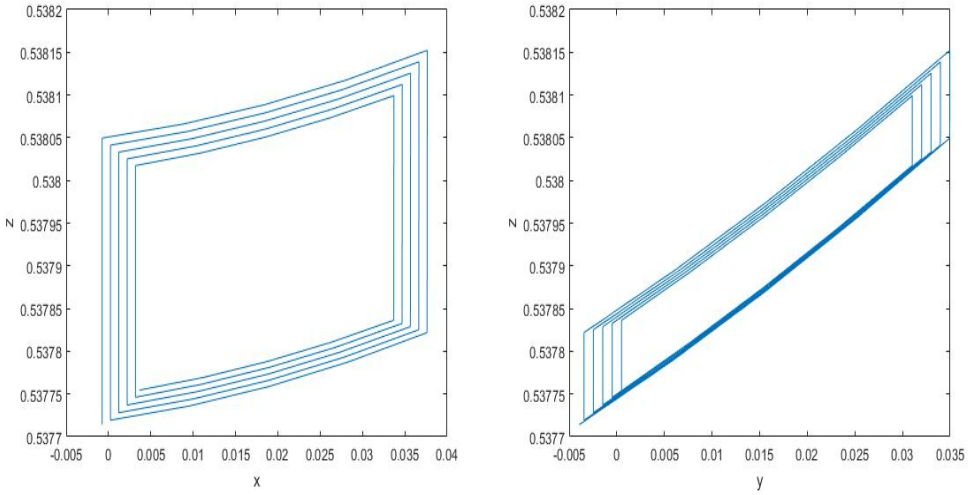


Figure 2.23: final result of tuning process simulation in the x-z and x-y plane

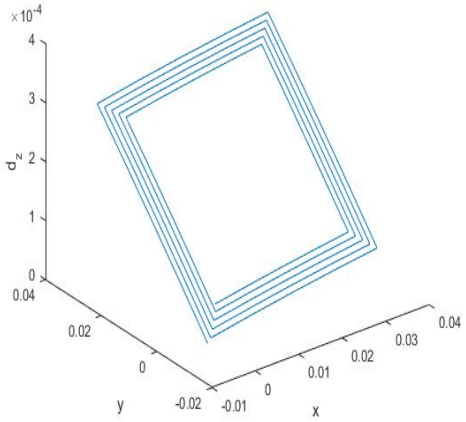


Figure 2.24: solution with initial Rp

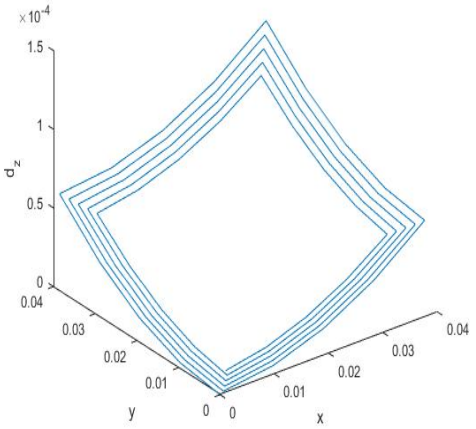


Figure 2.25: case with initial length, but different Rp

Chapter 3

CONTROL SYSTEM (HARDWARE)

3.1 Objective and Overview

The main function of the control system is to guarantee the correct operation of the entire machine. In order to control specific technological parameters and permit the mastering of the printing process the design of an integrated control system is developed taking into account the two subsystems, linear delta and extruder. The control system developed provide the possibility to follow a trajectory with a given velocity profile, to steadily reach the temperature in the different chambers and at the nozzle, to correctly cool down the overall extrusion system, to position the material as it should be inside the extrusion chamber and to extrude with a given feed rate. To complete these tasks the control unit needs to be connected to motors, resistors, a pump, an electro-valve, sensors and the external world (power, water and air grid).
In this chapter the hardware component of the control system is described and an electric layout is proposed to fit a 900 x 800 mm cabinet. After a concise presentation of the component and their function (3.2), the electrical hook up is shown (3.3). Then is the turn of the initial layout of the cabinet(3.4). The final solution is the one obtained from an external company that actually carried out the realization of the cabinet from the delivered information (3.5). The chapter is ended with the description of an heated bed that will be placed later on the machine(3.6). As can be seen in figure 3.1, the starting configuration was made by two separate subsystems with their similarities and differences that required to be integrated and adjust to fit together. All of these components are connected both to the control system and the power grid (same of any household appliance). The first fitting operation, as well as the one commissioned, are designed to ensure easy access to the main components and to allow an easy upgrading of the feature and maintenance. The heated bed is required to ensure adherence of the first layer to the moving plate, otherwise detachment has to expected (other solutions requires usage of more material that needs to be manually separated).

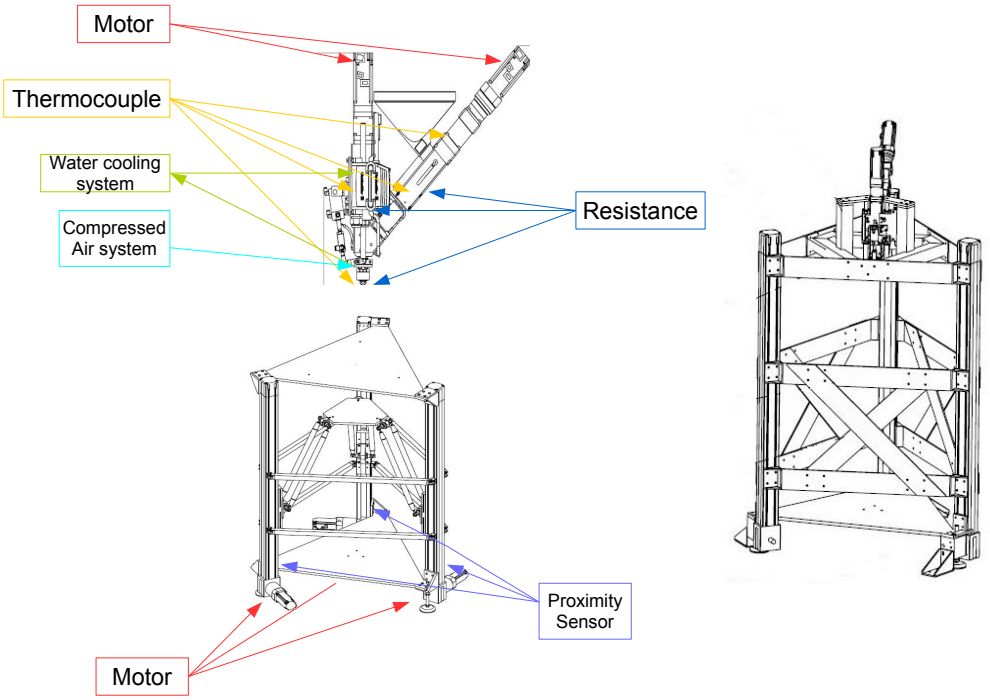


Figure 3.1: electric component outline of the two initial subsystem

3.2 Components and their function

The Motion Controller System is constituted by the rack of the PLC, by the motor control modules and by relays for the activation of cooling and warming system of the extruder.

component	reference	number	component	reference	number
PLC	Q03UDVCPU	1	6A Switch	-	5
Motion Control Unit	Q172DSCPU	1	Contactor	-	5
Servo drive	MR-J4-40B-RJ	5	Transformer	5V	1
HMI	GS2107-WTBD	1	Transformer	24V	2
Security relay	XPSAC5121	1	Relay RA	-	5
Emergency Pushbutton	-	1	Relay RB	-	5
Current Pushbutton	-	2	Varistor	-	5
25A Switch	-	1	prox. sensor	-	3
temp. relay	-	3	pump relay	-	1
air relay	-	1	thermocouple	-	4

For the cables please refer to the Mitsubishi manuals.

PLC *Programmable Logic Controller*, it's an high level controller for industry, capable of executing a code and dealing with digital and analog signals in input or output. It's made up by a CPU and I/O modules.

Motion Control Unit it's a control unit based on a microprocessor and a memory; it controls in real time all the axis of the system. It operates at a low level without exceeding imposed temporal limits.

Servo Drive The servo drive gives power to the motor as a function of the reference signal coming from the motion. Current, position and velocity loop are closed within this unit.

HMI *Human Machine Interface*, it's the interface between the machine ad the operator, thus allowing the selection of available operation saved in the machine memory. One touch screen is connected to the PLC.

Relays They are electrically actuated switches.

Pushbuttons They open or short the circuit allowing current flow leading to temporary or emergency stop.

Transformers Their main function is to feed to the various component with the required electric energy, converting the alternate current from the grid to continuous constant current.
In this project 3 transformers are required: one 5V for the PLC e Motion Control and two 24V suppliers one for input-output circuit of the servo drive and one for the electromagnetic break of the motor.

Magneto-thermal Switch Their main function is to guarantee the current flow with a threshold value to protect the downstream circuits.
One 25A switch is placed at the beginning of the plant and one 6A switch before every servo-drive module.

Contactor Similar in function to the relays, but with higher current resistance.
Only one contactor is placed at beginning of every servo-drive module.

Varistor Mainly used to prevent power surges in the electromagnetic breaks circuit.

Thermocouples A temperature measuring device in which two wires of dissimilar metals are joined together. The system features four of them on the overall extruder.

Proximity sensor A sensing device that operates without making contact with the measured object. The Linear Delta has one for each guide for the homing procedure.

Here are presented the electrical requirements for each component, please consult the relative data sheets.

Component	tension [V] AC	I [A]
servo drive	230	1.5
I/O servo drive	24	0.2
magnetic break circuit	24	-
proximity sensor	24	-

In Appendix A (A.6) the assembly specifics for the main control unit are quoted from the respective wiring manuals.

3.3 Electrical layout

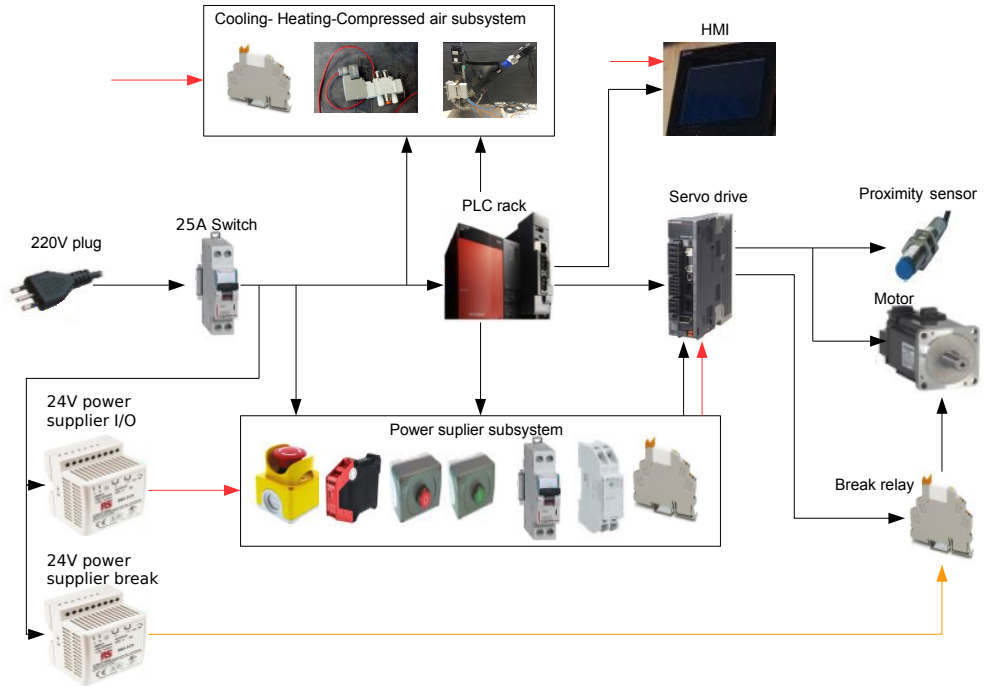


Figure 3.2: simplified electric layout

This section refers to the schemes that can be found in *Appendix A.5* of this thesis paper, while a simplified version can be seen in figure 3.2. In *scheme 2*, the main layout is shown. The main plug is connected to 220V single-phase current, since the servo drive can internally convert it to a tree-phase current (to power the motor). A 25A magneto-thermal switch is placed to separate the system from the power grid.
The main line is then split a first time to power the PLC rack, a second time to create a current circuit for the servo drive (it will be later clearly explained), a third by two power supply unit at 24V (one for the I/O and one for the electromagnetic breaks).
The rack unit can be seen in *scheme 3* and it's made of the 5V power unit supplier, the

PLC, Motion, the QX80 digital input, QY80 digital output, Q64AD analog to digital converter and the temperature control module. The PLC module is connected to the HMI unit by means of an Ethernet cable that is powered by the I/O 24 V power-supply (*scheme 6*). The motion unit is connected to the servo drives by means of SSCNET III/H (optical communication protocol) cables and is also connected to the 24V current by means of the safety relays. Once power is disconnected by pressing the emergency switch, this relay is not powered, thus the motion receives the information to promptly arrest the system (this is done to avoid that the control unit keeps trying to continue the work once the current has already been cut off from the motor). The digital input (and output) module as well as the analogue one can be seen in *scheme 7*. The digital output module is connected to the 24V current and controls the relays associated with cooling system (pump) and the compressor (*scheme 6*). Finally the temperature control module is connected to three relays controlling the passage of 220V current through three resistance placed on the extruder to heat it up (*scheme 6*). It's also connected to four thermocouples placed on the extruder. The 24-pins connector hook up table can be found in *Appendix A.5* that shows the port to port connection to be established in order to guarantee a correspondence with the controlling programs already set up.
A servo drive is powered through a 6A magneto-thermal switch that is directly connected to the internal control unit of the drive, while a second branch is connected to the motor circuit by means of a contactor relay (*scheme 5*). This component is powered by 220V current incoming from the ON pushbutton; two switches are connected to the motor circuit (by pushing the ON button the motors are powered) and one is connected in series creating a holding circuit for the ON pushbutton (the on state is kept until OFF button, or other emergency flags, disconnect the power to the coil of the contactor). In *scheme 2* the security circuit can be seen (in the simplified version is just stated *power supplier subsystem*). The safety relay is powered by 24V current that can be disconnected by the emergency pushbutton, as stated previously. This relay also controls a switch that cuts the powering of the system described in this paragraph so far. The current to the motors can also be disconnected by pressing the OFF pushbutton or in case of internal alarm in at least one of the servo drives. In fact for each and every one of them an emergency relay (RA) is connected to the pin terminals (connected to the MR-J4 through the CN3 port) that regulates the passage of current. The fact that they are connected in series assures a stop if anyone of them is faulty. In *scheme 4*, these components can be seen. The servo drive is connected to a break relay (BR) that controls the activation of the electromagnetic breaks in case of an alarm. The breaks are powered by a 24V current that for safety reason is required to be delivered separately from the I/O grid.
A proximity sensor is connected to the CN3 pin terminal box and to the 24V current in order to determine the home position of each linear guide.

3.4 First layout concept

Before going into detail of the first draft, the geometrical specifications of the different components required to be placed inside the cabinet are shown. They were used for the conceptual optimization of the fitting job and to assure feasibility with the mandatory requirements for safety according to the Mitsubishi® manuals. They are listed in the following table.

name	B[mm]	H[mm]	W[mm]	name	B[mm]	H[mm]	W[mm]
rear plate cabinet	750	800	10	Contactor	35	85	80
24V transformer	80	100	70	relays	6.10	92	60
25A switch	20	85	80	CN3 "box"	60	80	55
6A switch	20	85	20	MR-J4	40	180	180
Security relay	25	100	120	rack	250	100	10

The basic idea to fit all the equipments in the cabinet, in an efficient and smart way, was to separate the high tension branch from the low tension branch. The 220 V line is kept on the left while the 24 V one one the right. The second point was to separate each motor axis vertically so that future addition will be effortlessly (they would require the introduction from the right, for instance, of the new components on the respective DIN guide).
The first draft can be see in Figure3.3 where the PLC rack is placed at the top but is actual position will be determined by availability of the required cable length (one connecting the touch screen and the one to the first servo drive); relays, pushbutton and contactor just below; terminal box on the second DIN guide and on the button the motor drivers. Relays associated with the heating and cooling system are not represented, but they will be placed in the lower section.
The pressured air placed to the outer right in a separate space to avoid contact with electric components for safety reason (air will spread the flames in case of a fire). Equation used for the first assessment are (3.1) and (3.2). In figure 3.4 can be seen a slight variation on this concept where transformers are placed where needed.
It could also be possible to use equipotent terminal arrays to easily access to a given potential. Here follows some initial geometrical consideration:
H gap determination:

$$Htop = Hbutton \qquad h_{gap} = H_{alpha}/3$$

$$H_{alpha} = H - Hrack - Htbox - Hrelays - Hdriver - Htop - Hbutton \tag{3.1}$$

B gap determination:

$$n_{axis} = 7 \qquad B_{rel.block} = 6.10 \times 2 + 35 + 20 = 67.2$$

$$B_{element\_max} = max([B_{tbox}, B_{driver}, B_{rel.block}])$$

$$B_{alpha} = B - B_{element\_max} * n_{axis} - B_{left} - B_{right} \tag{3.2}$$

$$b_{gapmin} = B_{alpha}/n_{axis}$$

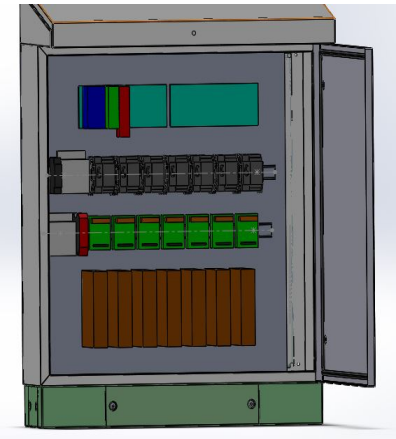


Figure 3.3: first draft

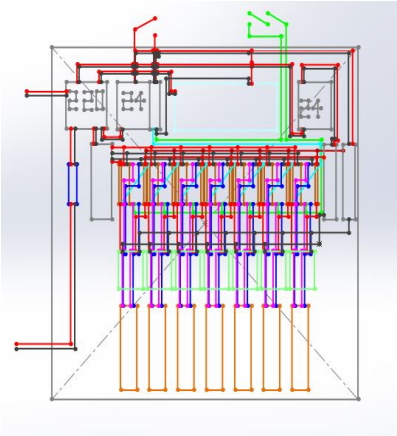


Figure 3.4: first draft

3.5 Cabinet realization

Due to safety requirements, it has been decided, later in the design phase, to assign the task of the realization of the electrical cabinet to an external company. Here are the requirements enforced to the final result:

- HMI screen, emergency stop, 7 female USB connector (one for the PLC, HMI and 5 for the servo drives) and a laptop stand are required to be placed on the control panel with a wall-plug to power the laptop.

- In the lower part of the cabinet, terminals strips need to be placed in order to later allow the connection of: thermocouples, thermo resistances, pump power, electromagnetic breaks, proximity sensor with their power supplies, all the usable connections of the QX80 module and Q64AD module (*scheme 5* and *scheme 7*) and 24 terminals (2 for each available QY80 module port) for future connection.

- easy and safe air connection to the electro-valve in the lower part of the cabinet.

- leave access for the encoder and motor powering cables (for each controlled axis) that will be connected later on to the machine. The same has been done for the heated bed components.

The final realization can be seen in figure 3.5 that reflects the specification imposed, following the electrical layout scheme available in the appendix, and with additional safety features (mainly a cooling fan to avoid over eating of the cabinet). The presence of the terminal strips enforced a different configuration of the components, leading to the servo drivers placed next to the PLC rack. The cables to the extruder are placed inside a corrugated tube to protect them.

Figure 3.5: final result of the electric cabinet

3.6 Heated bed

Commercial 3D printers also offer the possibility to place on the moving plate a heated bed to increase adherence. Efesto also offers such possibility.
The adjustable plate was designed to accommodate a 22x22 cm plate. At this stage of the machine, it's only a possibility that has been considered. In this early stages, the outline of the system is made of: the bed itself capable of heating up to 90°C, a controller board and a power unit. It constitute a completely separate control system that will be placed outside the cabinet for now, but room inside the cabinet has been left for it.
The bed is a printed circuit on a fiberglass wafer that heats up once current is applied by Joule's principle. Temperature is read by an NTC (Negative Temperature Coefficient) thermistor (resistance that vary according to temperature) of 100 kΩ soldered on it.
The control unit is the one used for open source 3D printers based on Arduino Mega, to allow, in the testing phase, enough literature to not represent an obstacle. It's sold with already loaded the latest firmware. This is useful because it's compatible with the printer client (control panel) Repetier-Host® (developed by Hot World Media) allowing both the regulation of the temperature of the bed, but it also allows the

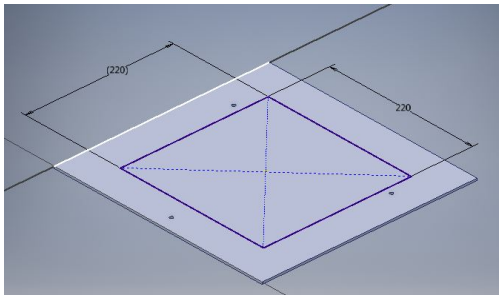


Figure 3.6: heated bed and its placing spot

slicing procedure. This printer client can use both Slic3r®and Cura® to perform the task.
The power supplier is required to deliver at least 15V and 10A.

assembly guideline:

At least four holes are necessary to place the bed on the adjustable plate. An additional window might be created to allow enough room from the NTC and connecting cables to the control unit, otherwise it could be distanced from the metallic platform by means of a nut or any appropriate gap creating element avoiding any accidental short circuit. Using pin terminals to connect the cables will increase versatility of the machine, in fact the heated bed would be easy to remove. Such feature could increase productivity if two bed are available. The cables run along a link, fixed by strips for electrical application, passing by the centers of rotation of the kinematic chains elements. This allows to block the length required from the prismatic joint up to the heated bed. The same can not be said for the segment associated to the guide. A longer cable, equal to a little bit more of maximum vertical displacement of a guide, is required. This cable is soldered to a female terminal compatible with the control unit.

Chapter 4

CONTROL SYSTEM (PROGRAMMING)

4.1 Objective and Overview

In this chapter, the attention is directed towards the programming part of the control system with the main goal being able to print any object desired starting from a G-code generated by a slicing software or starting from printing trajectories created by the user. After a brief introduction (4.2) on the main synchronization processes between the Linear Delta and the extruder and on standard motion law, focusing on the ones mainly used, the attention is directed towards the *G-code* reading script wrote in Matlab® capable of importing data of this format ad modify them accordingly (4.3). The information obtained needs to be properly fed to the Mitsubishi® softwares after being processed (4.4). Two possible solutions are available to the user: synchronous control or a new method that is more user-friendly and more close to the control of commercial 3D printers that is therefore presented (4.5). After this topic is fully exhausted, manual control is introduced in order to properly set the machine to print (4.6). This chapter opens also at the possibility to directly feed the *G-code* to the Mitsubishi controller with less invasive off-line operation.
During the description of these points, the analysis and comments of the Matlab® script will be illustrated to help any future user to fully understand and operate the machine. Only the necessary information concerning the *G-code* or PLC/motion language will be discussed (4.7).
The alternative control program derives also from need to reduce the number of points required to describe a trajectory of the Linear Delta. This does not solve completely the issue, but allows to print more layers with the same amount of points. The realization of a communication bus between the PLC and a computer will definitively eliminate the problematic.

4.1.1 Control outline

In Figure 4.1, the outline of the control system is shown where S are the data in the working space, while Q the same but in the joint space, q the one after motion law assignment and q^* the information actually feed to the motors.

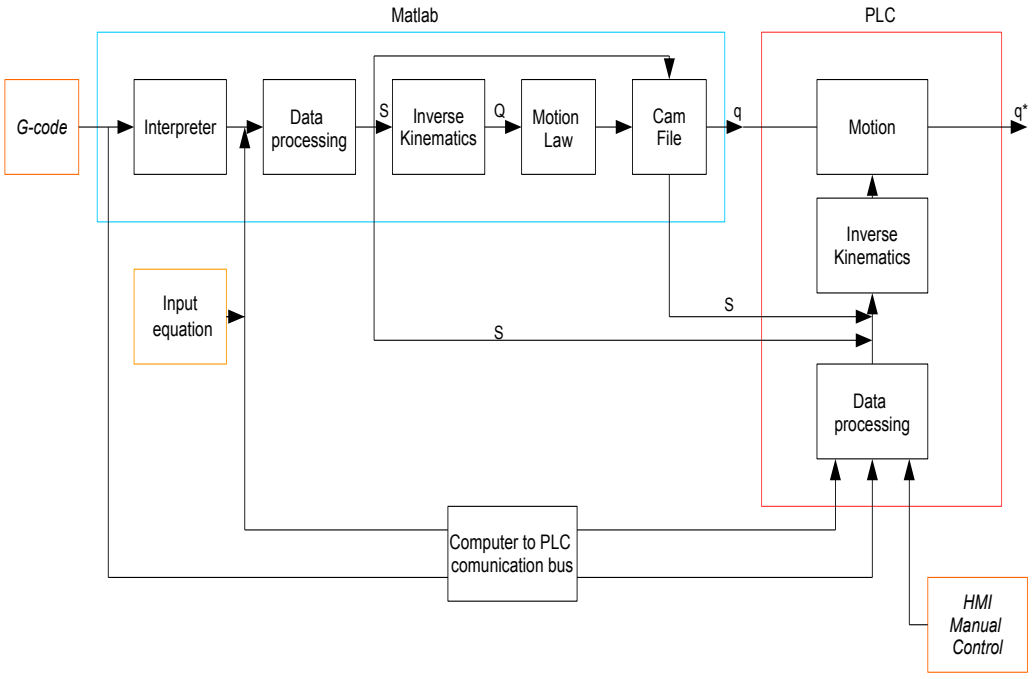


Figure 4.1: control outline

Three input typologies of data are considered with the first being the *G-code* obtained by means of any slicer software for 3D printers, the second the user defined equation describing the trajectory and the last the manual control from the HMI. Two main environments differentiate the in-machine operation (*PLC*) from the off-machine ones (*Matlab*) with their respective steps.
From the different inputs, several paths can be taken leading to the realization of the desired object. The plethora of paths are the consequence of the need in the early stage to improve the first programs as well as to test them while searching to exploit best all the possibility offered from the industrial control system.
Starting from the *G-code*: the information stored is read and processed by an interpreter (*G-code reader*) developed in the Matlab environment. The points are then processed (*Data processing*) to obtain a good discretization in the working space as well as time required for a given movement. The inverse kinematics is solved to pass from working to joint space after which any given motion law can be assigned (or not). The final information is saved in cam data files to be manually imported on the Motion

system that runs the "CNC emulator" program or using the synchronous control (in this thesis will not be discussed).
If user defined equations are used then the chain is practically the same, but without the interpreter since they should be directly defined in the working space of the machine and not in CNC language.
The user may wish to control the system from the HMI, in our case a touch screen, so an interface is in order to accommodate this need. The information is processed as it will be in the main *G-code* path and also the kinematic problem is solved, but this time it should be carried out inside the Motion before executing the exact point.
The design of these two steps inside the industrial control is the main reason behind the several combination (only the main ones are shown in the figure). Upon the development of a reliable communication bus between a computer and the PLC, the *G-code* could be imported using an interpreter before or even inside the control unit. The same could be done for the input equation. Inter-processing data can be drawn and feed to the PLC for testing and comparison.
This topic will be presented following the first path, followed by the manual control one as well as the on-line one (with this term the fact than no, or at least a few, operation are carried outside the industrial control is intended). It will be concluded with definition of the human-machine interface programs. The algorithms behind the user equation adopted in this thesis work will be discussed in chapter 5.

4.2 Preliminary information

4.2.1 Synchronization

It's clear how important is to adjust the velocity of the material flow with the one of the robot and vice versa in order to guarantee the exact volume and shape of material deposited. Using the principle of conservation of mass in the form of volumetric flow a tool to relate the two system can be obtained [1].

$$V_1 A_1 = V_2 A_2$$

This general formulation is then adjust to our case in (4.1) (4.2).

$$V_{extruder.out} A_{extruder.out} = V_{end.effector} A_{end.effector} \tag{4.1}$$

$$V_{extruder.out} A_{extruder.out} = V_{extruder.in} A_{extruder.in} \tag{4.2}$$

The definition of the layer height (d_z) determines the thickness of the trace (distance between two parallel lines of the square test) to allow a 100% infill, supposing that the section of the trace is rectangular, due to the selection of a $d_z \neq d_{nozzle}$, and the

[1] Bernoulli's equation can be used, but requires the knowledge of more therms like density of the material and static pressure, data unavailable at this stage.

velocities are the same:

$$r_{gap} = \frac{\pi d_{nozzle}^2}{8 d_z} \quad (2)$$

If instead the trace is kept fixed to the nozzle diameter and the velocity are left to change, then:

$$V_{end.effector} = \frac{\pi d_{nozzle}}{4 d_z} V_{extruder.out}$$

As for the Linear Delta, a calibrating phase is in order to adjust experimentally the model with the system, allowing the optimization of the machine as a 3D printer.

4.2.2 Motion laws

In this section, only the main equations are quoted, use references for a detailed discussion [20]. C_a and C_v are respectively the acceleration and velocity coefficients.
Motion laws could also not be used during the preparation of the data, as it will be explained later on.

constant velocity:

This motion law is one of the simplest to implement, but is almost impossible to obtain since it requires instantaneous peak acceleration:

$$C_a = \infty \quad C_v = 1$$

$$a = 0 \ ^3 \ ; \ s = C_v \ ; \ d = C_v x$$

trapezoidal velocity:

This motion law is one of the most used in mechanical application, although same variation of it is preferred (to avoid discontinuity in the jerk). It's the one that the motion control uses as default to change position thus it's required to be known to correctly predict the behavior of the machine.

$$C_a = \frac{1}{\xi_v(1-\xi_v)}; \quad C_v = \frac{\xi_v}{\xi_v(1-\xi_v)}$$

$x \leq \xi_v$	$\xi_v < x < 1 - \xi_v$	$x \geq 1 - \xi_v$
$a = C_a$	$a = 0$	$a = -C_a$
$s = C_a x$	$s = C_a \xi_v$	$s = C_a(1-x)$
$d = 0.5 C_a x^2$	$d = sx - 0.5 s \xi_v$	$d = 1 - 0.5 C_a - 0.5 C_a x^2 + C_a x$

[2] if d_z is close to 80 % of the nozzle diameter (approximation of $\pi/4$) then $r_{gap} = r_{nozzle}$. If $d_z = d_{nozzle}$, it'll be better to consider a circular section of the depositated material.

[3] for start and end point use equal to C_a

4.3 G-code reader

In this section the *G-code* interpreter will be presented starting from a few words describing the language itself; it will then move on to the algorithms, with their relative comments, implemented in Matlab that actually reads and adjust the file.

4.3.1 G-code introduction

A brief presentation of this language seems in order [21][22]. *G programming language* is one of the most spread Numerical Control languages. Here follows an example (obtained with CURA ®):

$;FLAVOR : UltiGCode$
$;TIME : 813$
$;MATERIAL : 1225$
$;MATERIAL2 : 0$
$;Layer\ count : 48$
$;LAYER : 0$
$M107$
$G0\ F9000\ X104.844\ Y103.451\ Z0.300$
$;TYPE : SKIRT$
$G1\ F1200\ X105.845\ Y102.439\ E0.17081$

In the first five rows information on the file generated is stored (they do not affect the run on the code since their are commented (*;*)); then command are stated each line. Here is a table with the most common commands for a 3D printer machine:

variable	**description**	**example**
Mxxx	action code	M107: Fan off
		M112: emergency stop
		M109:set extruder temperature and wait
Gxxx	motion command	G0: rapid linear movement
		G1: linear movement
X	absolute x position	
Y	absolute y position	
Z	absolute z position	
E	incremental extrusion position	
F	velocity	[mm/min]

A G0 command is usually without an E variable to avoid a loss in quality of the print. This variable also depended on the slicer used so they should be investigated according to the software used. There are several slicer available, most of them also are open-source, like CURA® (by Ultimaker®) or Slic3r®. They are simple to use,

they need to be set for the machine considered (usually a Cartesian Robot) imposing (as our robot is concerned) these main parameters:

nozzle diameter	0.9mm
layer height	0.45-0.9[4] mm
fill density	100%

The fill density needs to be 100% to avoid bad pieces after the final procedure that each piece will undergo. Setting this parameter automatically void the possibility to choose different fill pattern. Lower fill density values can and are expected to be used during testing to evaluate the best fill density to avoid significant problematic due to evaporating bubbles during sublimation of the binder (space left empty on purpose can allow the gas to remain in such rooms avoiding internal strain of the metal object).
Layer height requires to be lower then the nozzle diameter to assure good adherence with the plate or the previous layers. When choosing, be careful to take into consideration the behavior of the machine after the calibration process. If z-axis variation occurs due to an uneven plate adjust also the first layer height (100% or more of the following layers) to assure good leveling, otherwise it will be impossible to correct the final object geometry.
Due to the extruder configuration, a skirt is required to properly set the material flow, in any case make sure, previous to the program start, to position the printing system ready for a good stream. A brim and/or a raft can be placed at the bottom to increase adherence with the plate (if no heated bed is used) and avoid detachment of the printing object from the plate during the building procedure.
No parameter regarding the temperature of the bed or the extruder need to be set in these slicing softwares, but the printing velocity need to be chosen avoiding maximum capability. For the initial tests do no exceed movement velocity of 2 cm/s. The F value is the one required to move the TCP in the x-y-z space from one point to the other; from it and the extrusion E, it's possible to reconstruct the velocity at the nozzle.

4.3.2 Repetier-Host® *G-code*

It was suggested the possibility to use Repetier-Host® as a slicer environment, since is already used as controlling interface of the heated bed. Cura® Engine (*ver.15.02.1*) is the one used to save the *G-code* with the specifics detailed in the following section. Unfortunately the one saved with the printer client (if Slic3r® is selected) has some different features, for instance there's no indication of the layer transition, the extrusion filament keeps changing reference value and the disposition of the commands are slightly different. Some modifications are required to adjust it to the script already implemented.

[4]the software itself advise to use no more than 80% of the nozzle diameter

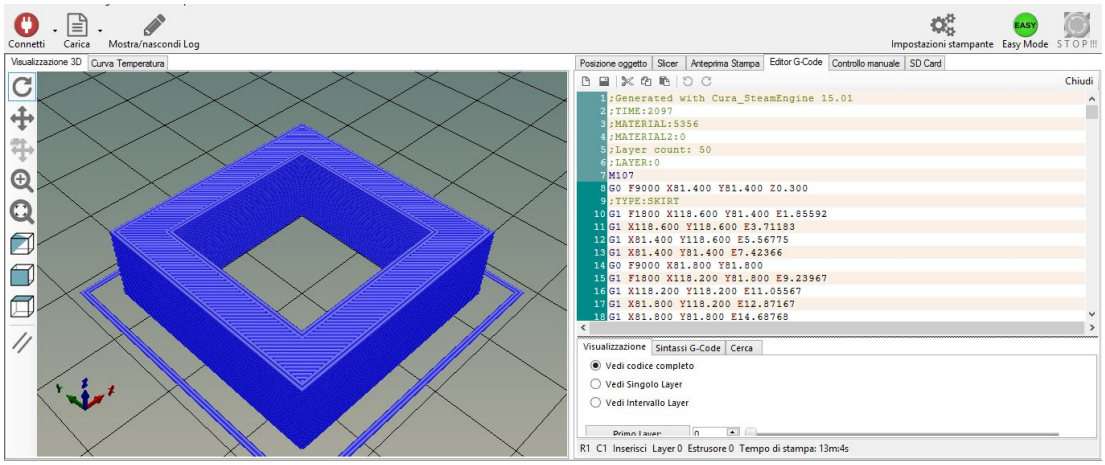


Figure 4.2: G-code (Cura® Engine) example generated by Repetier-Host®

Repetier-Host®: Cura® Engine:

In configuration, set *G-code* start:
;TIME:1
;MATERIAL:0
;MATERIAL2:0
It's a fake command used to adjust it with the one naturally created by the slicer (allows the creation of three nonempty lines that are created usually with stand-alone software).
In *extrusion* also make sure that the retraction of the filament is not selected. Refer to the next sections for additional changes to impose to the text.

Repetier-Host®: Slic3r®:

In configuration, select a *G-code* flavor by MakerBot.
In the Printer settings: add *;LAYER:[layer_num]* in Custom *G-code* before layer change *G-code* and *END* in end G-code (Figure 4.3). In Extruder 1, set to 0 the retraction length. Open the text and write the first lines to resemble the following example:
; generated by Slic3r 1.2.9 on 2017-02-22 at 23:32:09
; external perimeters extrusion width = 1.00mm
; perimeters extrusion width = 1.70mm
; infill extrusion width = 3.97mm
;Layer_count: 49 [5]
Of all the lines, the one that really matter is the fifth that should always be the number of layers. In extrusion also make sure that the retraction of the filament is not selected. The final result should look like the one in figure 4.4. Refer to the next sections for additional changes to impose to the text.

[5]Use the right number of layer count

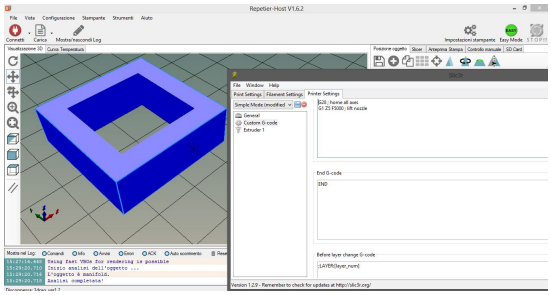


Figure 4.3: slic3r G-code variation

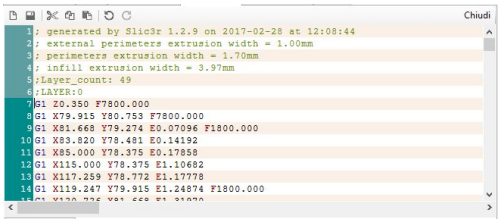


Figure 4.4: slic3r G-code first lines example

4.3.3 Algorithm variables

Before entering more into detail of the actual code, the main variables are presented to allow a better comprehension of the topic.
Upper level:

variable name	function
fileID	file Gcode to open
status	sub-cycle temporary break
$flag_{base}$	initial layer[6]
LAYER	vector of layers interested in
$layer_{count}$	number of layers

Lower level:

variable name	function
T_x	X coordinate
T_y	Y coordinate
T_z	Z coordinate
$T_{ext.on}$	extrusion state[7]

4.3.4 Algorithm outline

1. main execution where run data are stored and can be changed
2. upper level: $LAYER = [1, 2, .., nr_{layer}]$
3. upper level: for-cycle for every LAYER aforementioned
4. for-cycle: lower level function for the i^{th} layer
5. lower level: it reads every line separating G0 to G1 to others commands and for each of them distinguishes X,Y,Z and extrusion position (only on/off, no rise) and velocities until it reaches a new layer or an empty line.

[6] the MatLab ® script works only with layer figuring una tantum
[7] 0: off state, 1: on state (extrusion)

6. for-cycle:

$$x_{min} = \min(T_x) \quad y_{min} = \min(T_y)$$
$$d_x = \frac{\max(T_x) - \min(T_x)}{2} \quad d_y = \frac{\max(T_y) - \min(T_y)}{2}$$

7. for-cycle: all variables are saved in cell arrays

8. upper level: variables named as in the for-cycle are vector here. (this step is for centering the object to the plate)

$$X_{min} = \min(x_{min}) + \max(d_x) \quad Y_{min} = \min(y_{min}) + \max(d_y)$$

9. upper level: new for-cycle to refit all the points to the new configuration

10. for cycle: X,Y are cell arrays, function of the i^{th} layer

$$T_x = X - X_{min} \quad T_y = Y - Y_{min}$$

In the next subsection the algorithm will be explained using the code in the appendix as an explanatory tool.

4.3.5 Matlab® scripts

The G-code file was obtained using Cura® Engine (*ver.15.02.1*). Please refer to appendix B for the actual code:

Main:

In the first section almost all the variables that the user can exploit are declared in order to allow an easy change from run to run. Geometrical parameters of the robot are the first ones, followed by the maximum acceleration and deceleration of the linear guides (useful to adjust velocity if it's required to assure the following of a given motion law in a given time). *flag_plot* can be used to plot after the reading process the data, especially if modification are made in the code; *nr_layer* allows the selection of a partition of the layers available if aimed analysis are required. *dL* stores the maximum rise in the working space allowed to avoid any unwanted movement as presented in the mechanical chapter. The maximum number of point for file can be set to split the overall layer in a new bunch (such procedure is naturally done by the saving function for the velocity-displacement format as it will be shown in the following section). Finally the saving title can be inputed as well as the partition, if interested in, of the layers that are required to be saved.
The script then launches the reading (actual *interpreter*), intermediate point generator and delta time generation (*data processing*), if required, layer reducer and motion law assignment (they need to be manually placed), closed by animation and saving.

Reader (Upper level):

This function is the main one to recall in case of *G-code* acquisition. After some variables declaration, *flag_base* is set equal to zero. Such value can and need to be changed according to the file imported; the majority will present no support material (it's also possible to add it during these adjusting phases), that will feature as negative layers. In that case it's required to set the value as the layer stated in the first rows of the original file. If in future they will be a basic feature, it's possible to modify the "*G_code_layer1*" function to also give back the first layer value. Since it was mentioned, this sub-call routine as the solely responsibility to read the final number of layers; the user could also choose in the pop-up menu window to use only a set of layers, he or she declared in the main file. A menu based interface allows the user to adjust correctly the script for a given G-code flavor.
The upper level function then recalls the lower level one (read the dedicated subparagraph) inside a for-cycle that is forced to start from the top once the *status* turns to 1 in the sub-function. Before completing a loop, it analyzes the data to be able to properly reposition them in our printing area and to save them in a proper form, called structured (a matrix that is not bounded to have necessarily same number of columns, since the count of points per layer could and do vary from one another). Once all the layers are known, the code starts to rescale the information to the machine center (like a rigid movement). If future system variation do require any other particular change in position (such as offsets) or orientation of the x-y axises modify these lines[8].
Without going much into detail there is also a function that plots the final data stored. This was mainly done as visual aid to see the final printed object, as for correction purposes. It could also prove effective when using the layers reduction script (*layer_reducer*), that divides all of them into bunches with same number, by the principle that significant variation in the geometry leads to equal changes in the points counts. Since this is a weak statement, but easy to run and numerically not challenging operation, needs to be visually checked (Figure 4.5).

Velocity acquisition from *G-code*:

In the final part of the reading script, velocity obtained from the *G-code* are shaped accordingly. The layer script (lower level), as it will be seen, creates a 0 value every time that F command is not declared, knowing that F stands as long as it's not changed, the following lines adjust the F vector to contain the right velocity for each movement (speed, in the work space, required to reach point i to $i+1$). The travel value for the machine and the extruder are known so it's possible to obtain the velocity that the filament of a commercial 3D printer requires to be forward of. Accordingly to the flavor of the G-code, the value saved in E is different. The one obtained in Cura Engine resemble a volume, defined like this $E = t \times h \times dl$ where dl is the variation of the

[8] 120°rotation is easily obtained changing the numbering of the linear rails (changing the input order)

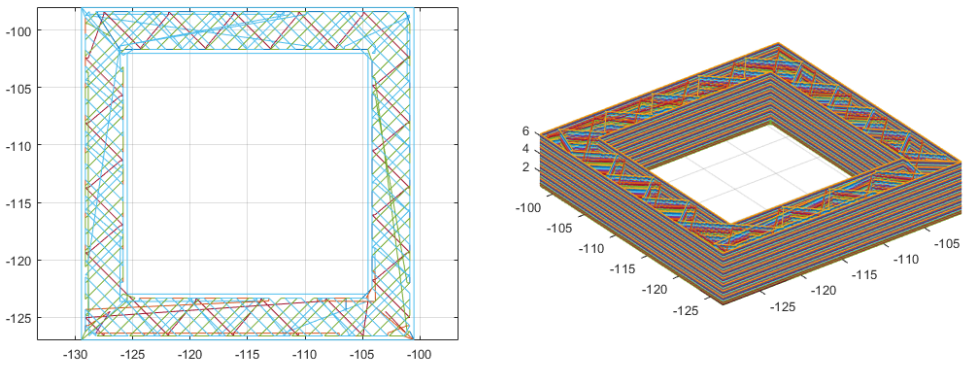


Figure 4.5: example of drawing in Matlab of the *G-code*

movement of the robot, h is the layer hight and t is the trace left on the plate/object. Thus it's a loose statement that in F_{ext} is contained a velocity of the filament, but allows to clarify the idea behind. More generally $E = A_{printed}dl$, therefore $A_{printed} = E/dl$ that can easily be substituted into (4.1)[9]. The *G-code* obtained with Repetier-Host uses already the extrusion displacement, anyhow make sure to check the saving property of your file to properly adjust the reading function to the specific case.

Layer (Lower level):

This subroutine is the actual *G-code* interpreter. In the main script, the user has the possibility to choose the file he or she wants to open. After some technical operation, a cycle reads line by line until a stop condition is met. In a simple way, it reads the first few letters and accordingly assign X, Y, Z, F and E data to a vector. The E case is very simple, since no increment value is required for our printing system, it just distinguish the extrusion case to the not case, although it's used to obtain velocity of the material at the nozzle.
All of this could have also implemented in a more efficient way, instead of restarting the line analysis for each case. The modularity offers to any user the capability to copy, paste and adjust to future necessity or flavors.
It then continues by checking if no break condition has been reached. The first one is the empty line, that actually requires a few words. Some slicer leave a space in one of the first rows that needs to be eliminated, fortunately this type of file can be easily opened with any text editor and modified. Other condition are related to an M action code, that depends on the slicer used (on the firmware used by the 3D printer that the slicer has been designed for) and thus will require special attention (they were designed

[9]Such formulation was not included inside the present scripts commented due to the lack of necessity in the early stages, but was introduced as a guide line for future usage. In general this depend on the filament diameter: Cura by default uses 1.75mm while in Slic3r the diameter needs to be manually indicated. This allows to directly describe our system in the CAM software; using directly E and F to move the extruder.

to work with a CURA® elaborated text).
The Slic3r case did required a few changes in the data acquiring method due to the tendency to break z-axis movement from the other commands and for the breaking condition (as can be seen in the appendix).

G-code layers agglomeration:

Due to the nature of the controller (without the communication system), it may be required to split or combine several layers into one. The cam data has a limited unit storage memory, that can hold up to 16 cam profiles of 8000 points (128000 total points). If using the new version of the controller, the number per file is limited to 2000 (actually 2048) points, but the overall memory can be filled with more than 16 data file (62 cams file[10]). This leads to the necessity to break (done by the saving program) or join layers. Here is presented the algorithm that joins them: each layer is analyzed one after the other and the single length is stored in a variable; until it reaches the maximum value, it saves the information in a temporary variable. Once met the limit the variables are saved in their respective structured array.
The user can select the number of points that define the new block combining a lot of layer to one or few and let the saving option split them up accordingly.

G-code layers reduction:

If 100% infill is used, all the layer will be printed using 45deg lines (or with different angles if stated otherwise). It will seem unnecessary to use a lot of data to simply repeat information already stored. The layer deposition using a line infill sees the alternation of 45 and 135 deg lines creating a grid every two layers. Therefore the knowledge of two layers and for how long they repeat themselves is enough to print the object. In order to tell them apart a script is easy to construct that analyze the number of points per layer and if the values discard from two consecutive layers more than one (or more if it's deemed necessary) then it hypothesizes that a variation in the geometry has occurred. This procedure should be used if the communication bus has not already been established and high number of layers are to be printed. Also check feasibility of such approximation with specific used object.

G-code saving option:

The Matlab® main script for the *G-code* interpreter allows the user to decide which saving format to use: four cams containing x, y, z, extrusion velocity and moving base cumulative velocity on the cam-master axis using the *on-line* (see next subsection) algorithm, or the velocity-displacement approach (*off-line*) as it will be presented later on, with the fourth cam velocity-velocity of the extruder. The third option, on-line, is to feed the x, y, z, extrusion and velocity, as in the case of the *G-code*, and let the

[10] the estimation needs to be done on memory consumption

motion move the machine as it will be presented[11].
The actual saving function for cam-file format splits the data in as many files as necessary to assure the possibility for the motion to read and execute all the information stored.
In order to prepare the system for what will come later on, a general saving function (*save2txt_gcode*) creating back the *G-code* is implemented. This could be used as stand alone to create the final interpolated data vector or, with simple adjust, correct it to save in whatever format to match the feeding data program of the on-line control. Such function does not sees any limitation on number of points for saved file, but it offers through a pop-up menu to chose if all the layers wished to be saved or a smaller partition previously indicated in the input part of the main script. The *save2txt_gcode* requires to be expanded, if future paths will take the machine there, to also features the point by point velocity both for the extruder and the moving plate.

```
;txt file similar to G-code
;used layers: from 2 to 3
433
G0 X0 Y0 Z0.7
G0 X6.825 Y-6.825 Z0.7
G0 X13.65 Y-13.65 Z0.7
G1 X13.65 Y-4.55 Z0.7
```

Figure 4.6: result of the saving function to create back a G-code file (but as a .txt); the second row states the layers saved in that file; the third is the overall number of points containing actual data

4.3.6 G-code reader consideration

To assure a smooth reading process, here are the points to make sure to follow:

- *G-code* should not have any empty line until the end.
- the fifth line needs to contain the total number of layers.
- the Cura engine generated file should end with the final layer to break the reading.
- it's advised to use the velocity ratios (or values close by) obtained by the slicer to have good layer deposition. If they need to be changed, rerun the slicing software rather than modify them on existing trajectories.

The code here presented adds a starting and ending point in the center of the machine and interlayer movement is done between them. If required the code can be modify to

[11] differences between the first and last case are related to the motion itself (as seen in the control layout): if data is stored in a cam data file or directly communicated in a memory block. Due to a lack of the communication parameters up to this day only a general .txt file can be presented as example of this third point.

completely reflect the imported file as does the on-line case. Such decision was made to allow a easier testing platform in the early stages since it's possible to stop after one layer or keeping repeating the same two layers to reduce the number of points needed for a single built.

4.4 Data processing

Here the adaptations that the information requires to undergo before being uploaded to the machine are discussed briefly. The first step is to check if the travel value is feasible or if it has to be chopped into small segments. The second task left to carry out is to determine the time required to go from point *i* to *i+1*. This requires to choose one of the two motion profile (or other, but they need to be implemented). Once solved, the actual motion law has to be assigned (also leaving the Motion to use its default profile it's a choice). For any further information, please consult any book on this topic[20]. These points are valid for *G-code* and user input equation.

4.4.1 Intermediate point generation

As stated in the mechanical behavior, a parabolic-like shape trend characterizes a movement between two points given any standard motion law. Slicers, like the ones mentioned before, do not take into care this issue (since they usually are combined with Cartesian 3D printers); so a few line are in order to properly analyze the data and adjust it to our case. Here is the algorithm with the main equations:

1. for each layer:

$$flag_i = [0, i, 0] \quad i = X, Y, ext.on$$

$$d_{x,y} = diff(flag_{x,y});$$

$$l_{seg} = \sqrt{d_x^2 + d_y^2}$$

2. for each layer:

$$if: \ l_{seg}(j) > dL \quad j = 1, 2, .., n_{l_{seg}(1)}$$

$$n_{points} = ceil(l_{seg}(j)/dL) + 1$$

$$p_{flag.i} = linspace(flag_i(j), flag_i(j+1), flag(j)) \quad i = X, Y$$

$$if: \ flag_{ext.on}(i+1) == 0 \ or \ 1$$

$$p_{ext.on} = zeros/ones(1, n_{points})$$

3. put all together in XX_{final}, YY_{final}, EE_{final}

The same is also done for the extruder and the velocities.
The code determines the distance between two consecutive points, then uses this information to evaluate if it's required to add intermediate points. Once established the

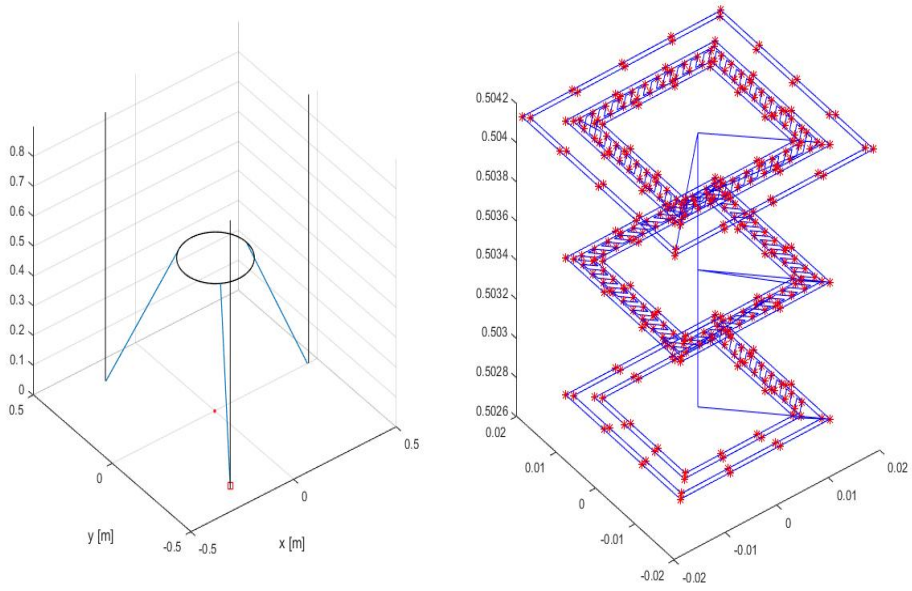


Figure 4.7: example: result of the reading process: 30x30mm external-20x20mm; sliced with Cura; 100% infill; obtained by feeding to the direct kinematics equations (to simulate machine behavior) the absolute position of the three linear guides

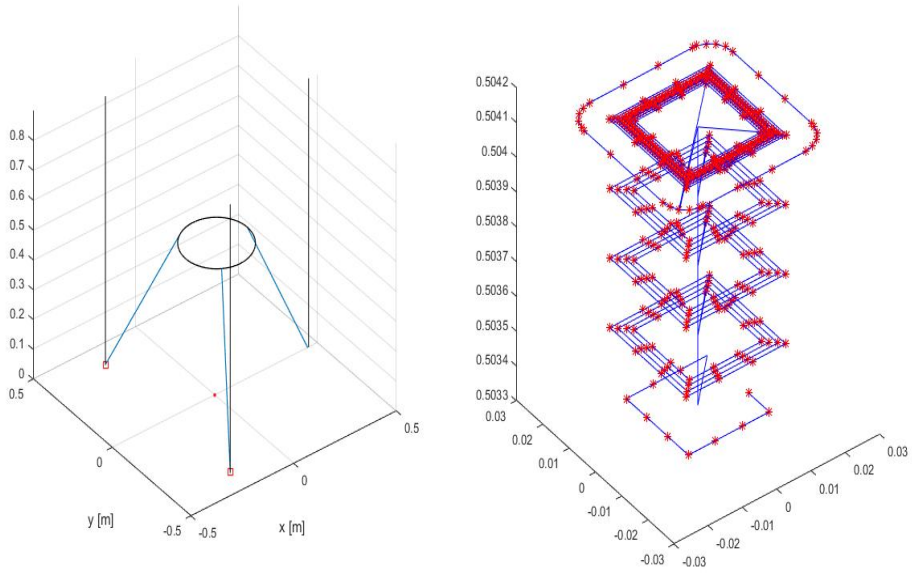


Figure 4.8: example: result of the reading process: 30x30mm external-20x20mm; sliced with Slicer; 40% infill;obtained by feeding to the direct kinematics equations (to simulate machine behavior) the absolute position of the three linear guides

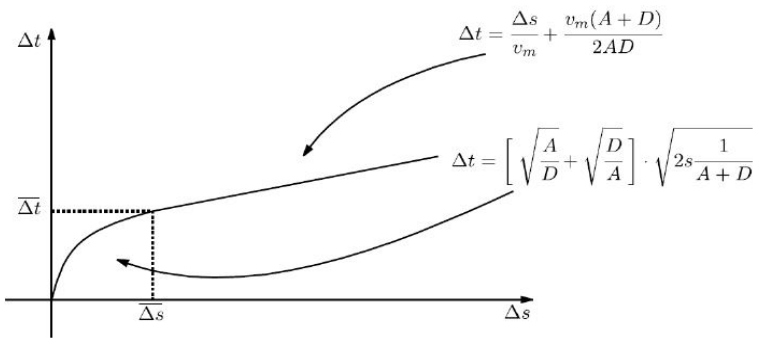


Figure 4.9: trapezoidal velocity profile. in our case:A=1200000 $[mm/min^2]$ and D=2A

case, it divides evenly the space in as many segment as calculated. The on/off state of the extruder is simply propagated to the additional points.
Due to the start and ending in the (0,0) coordinate the z-axis movement is achieved externally from the main data (for input user equation) or extending the last velocity definition to this movement. The selection of this interpolator instead of other available was to ensure in the early stage of the machine the control of the z-axis, avoiding any accidental failure of the system due to bugged codes.

4.4.2 △t determination

The variation of time is used to pass from velocities in the working space to the ones in the joint space without the Jacobian matrix, that if implemented in the on-line control would require more time to solve the equation and memory consumption.

1. for each layer a for-cycle (j from 1 to $length(XX_{layer})$)

$$p = [XX_{layer}(j); YY_{layer}(j); ZZ_{layer}(j)]$$

$$alpha_i = linear\_inverse\_kinematics^{12}(p, l, s, Rp) \quad i = 1, 2, 3$$

2.a Constant velocity:

$$d_x = diff(XX_{final}\{layer\}) \;\; same\ for\ d_y$$
$$L = \sqrt{dx^2 + dy^2} \;\; ; \;\; \triangle t\{layer\} = L/v_{ext}$$

2.b trapezoidal velocity profile:
look at figure 4.9 for the formulation. Use v_m not as the actual maximum of the machine but a lower value, according to the work-space requirements. If using *G-code* file, use F as v_m, making sure to check the final result from unexpected variation.

[12]or other definition of the kinematics. linear_inverse_kinematics1 is used for different length of the links; linear_inverse_kinematics2 is the one that uses the numerically solved equations

4.4.3 Motion law assignment

$$d_i = diff(alpha_i) \quad i = 1, 2, 3$$

$$alpha_{temp.i} = alpha_i(j) + d_i(j)d1 \quad (j\ from\ 1\ to\ length(d_i))$$

The final results are listed in $alpha_{motlaw.i.final}\{LAYER\}$, for any layer. *d1* stores the normalized motion law. This step can be skipped if the default motion law of Mitsubishi is used. If the synchronous control is not used the Motion will either way approximate in this way two consecutive points.

4.5 CNC emulator

This section deals with a different way to control the machine; the name is just a representation of the control philosophy of these kind of systems. It derives from the need to reduce the count of points and to overcome issue that may rise with synchronous control, such as mis-filled information, out of phase cam or accidental changes in the setting. This approach is not bulletproof, but could be easily adjust so that only cam file importation is required by the user without no change in the motion controller whatsoever (if no communication bus has been created yet).
The method is based on the exportation to the control system of the displacements of the linear rails or with their differences. The choice is done by the user if to reduce the burden or not on the Motion, both cases will be presented (they only differ from three additional memory block storing the previous positions). In the alternative case, four data are written in three cams (3 displacements on the y-axis and on the same x-axis the travel value of the cam-master); it would appear that to complete our task a fourth cam data file is required. The control is actually made to believe to posses standard cam format by simply writing on the x-axis the cumulative sum of the velocity of the longer displacement. Full motion laws or only one point every $\triangle$L can be used or even letting the motion controller discretize the travel in opportune intervals (see on-line control) storing directly the work-space coordinates instead of the joint-space ones.
The focus now moves on the control system where all of this simple operations need to be carried out to print. A list of memory blocks used are listed below with further comment on these. Two types of printing methods were required in the early stages: continuous and discontinuous deposition. The first suits the need to test deposition with a good stream without any break that could prove challenging while setting the machine, although a basic test of pause is done from one layer to the other. Since not all the cases can be well described with a continuous line, a discontinuous version of it was written. This well adjust to any other case (both *G-code* and user equation). This last variation requires a fourth cam containing the extruder breaks (as do the synchronous control).
The development of this algorithm is mainly due to the presence of two functions of the Motion software: one is a cam read function that easily allow to extrapolate information from effortlessly imported data files and an incremental control function allowing

to move three axis simultaneously.
One of the main limitation of this approach, if used off-line, is that intrinsically the cam read function is bounded up to 2048 points per cam file (using point data format), requiring the user to input more files that in the previous case. One main other limitation is the initial version of the emulator is the approximation of each segment with a trapezoidal profile causing continuous cycles of acceleration and deceleration. If this approach is proven to be affective this could be tried to be solved, realizing a control system that is even closely to a CNC.
One last issue concerns the usage of motion laws different from the trapezoidal one, that is the golden standard of the control system. With this method, they will be approximated leading to results different than expected (in time and therefore in deposition). To overcome it, the focus is moved on the average velocity to match the one required to complete the task. A brief study on effect of deposition diameter due to mismatch in the velocity profile will be illustrated.
In the Matlab scripts the variables are already prepared for the Mitsubishi control system, by describing the guides displacements in $10^{-1}\mu m$ while velocities in $10^{-2}\ mm/min$.

4.5.1 Velocity-Displacement Cam file

They are presented considering a constant velocity motion law. The main variation from any standard cam storage file is on the cam time/angle axis (here called DX_{camV}). The Matlab script realized allows to prepare and save these three format for this particular solution.

1. **full motion law**:

$$P_{max} = \max(|\ diff(P_{flag}, [\ ], 2)\ |) \times 10^3$$

$$v_i = \frac{P_{max}}{t_i} \quad ; \quad DX_{camV} = cumsum(v_i)$$

2. **full motion law (differences)**: after the previous steps, *P_flag* is substituted by:

$$P_{flag2} = [zeros(3,1), diff(P_{flag})];^{13}$$

3. **only one every** $\triangle L$: (without motion law assignment)

$$P_{i.flag3} = alpha_i\{LAYER\} - alpha_i\{LAYER\}(1);$$

Then same as point 1, but with $P_{i.flag3}$.

Where *P_flag* is $alpha_{motlaw.i.final}\{LAYER\} - alpha_{motlaw.i.final}\{LAYER\}(1)$.
The single values will be broken up by the Motion unit.

[13] this does not represent a Matlab instruction, only used the notation to sum up information

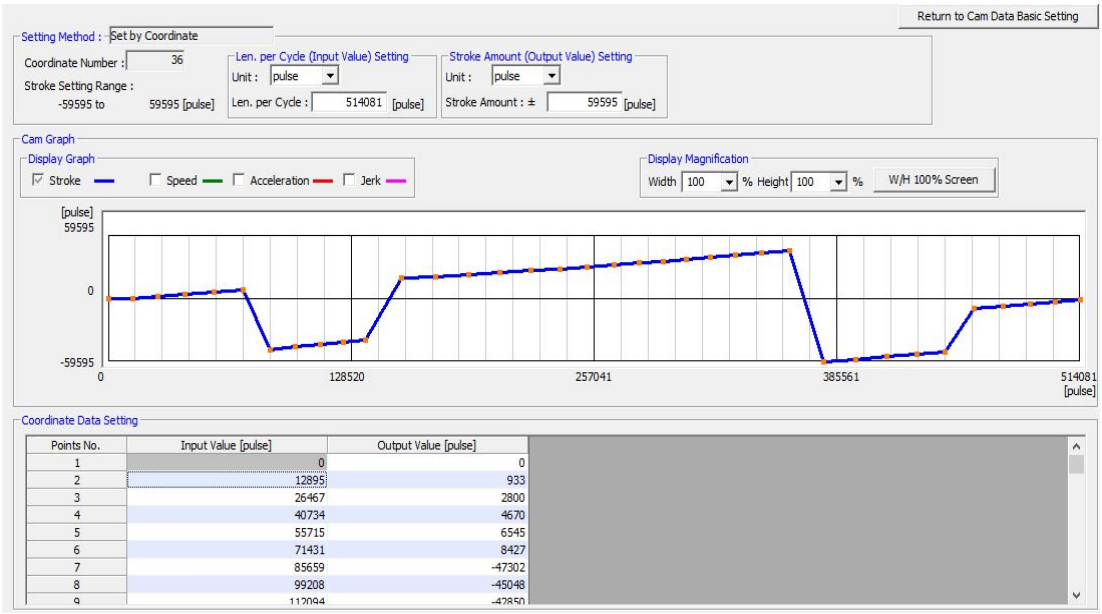


Figure 4.10: example of the velocity-displacement cam data format imported inside the Motion software

4.5.2 Memory allocation

These are the memory of the Motion used for the continuous deposition, additional term related on the fourth cam can be found in the algorithm paragraph. The selection of # or D at this stage (testing) was entirely an arbitrary choice.

#0L	number of cams transition		#104L	cumulative velocity at (i)
#2L	cams transition variable (j)		#106L	j^{th} cam 1 increment
#4L	N.points of j^{th}cams		#202L	j^{th} cam 2 N.points
#6L	N of layers[14] variable		#204L	cumulative velocity at (i)
#10L	iteration variable (i)[15]		#206L	j^{th} cam 2 increment
#12L	N of layers count		#302L	j^{th} cam 3 N.points
#20L	j^{th} cam 1		#304L	cumulative velocity at (i)
#22L	j^{th} cam 2		#306L	j^{th} cam 3 increment
#24L	j^{th} cam 3		D556L	velocity at (i)
#102L	j^{th} cam 1 N.points		D558L	cumulative velocity at (i-1)

4.5.3 Algorithm

The continuous and its opposite are discussed simultaneously pointing out their differences in the operation sequence and memory allocation. Here the two cases are summed up for a rapid consultation and overview of the coding:

[14] used for manual control of layer deposition. Set #12L equal to one when using data from *G-code*
[15] within j^{th}cams

Continuous deposition:

Printing:

1. set number of layers
2. while-cycle until all layers are printed. At every cycle execute simultaneously algorithm "Point by point" and "Extrude". Before completing the cycle it prepares the moving base for the next layer.

Point by point:

1. set first three cams and number of sets of cams to read
2. reset required variables to 0
3. while-cycle until all cams to read have been considered: first determines number of points for that given butch then it creates an inner while-cycle until all the data stored are read. It evaluates the current velocity from the x-axis value and the previous saved:
$$D556L = \#104L - D558L$$
Position and velocity are fed to the move function.

extrude: moves the piston according to feed value

Discontinuous deposition:

Point by point:

1. set first four cams (new one #400L-#406L) and number of sets of cams to read
2. reset required variables to 0

3.a while-cycle until all cams to read have been considered: first determines number of points for that given butch then creates an inner while-cycle until all the data stored are read. Velocity are found: $D556L = \#104L - D558L$.
if $\#406L == K0$, it sets M3260 to stop motor 4 (extrusion) and vice versa.
Position and velocity are fed.

3.b as 3.a, but the fourth cam has extrusion velocity on the y-axis instead of a simple 1 for on-state.

Memory #6L is set equal to zero in order to start from the beginning the execution of the layers. In memory #12L the number of layers needs to be inserted; for manual operation this need to be stated, while when using *G-code* this requirements is overruled by the number of cams. A pseudo while-cycle is then started until #12L value is reached; the first operation is to rehabilitate the extruder motor that in between layers is manually stopped (see Figure 5.5). Later on two subroutines are called simultaneously, one that controls the Linear Delta and one the extruder. Once the points available for

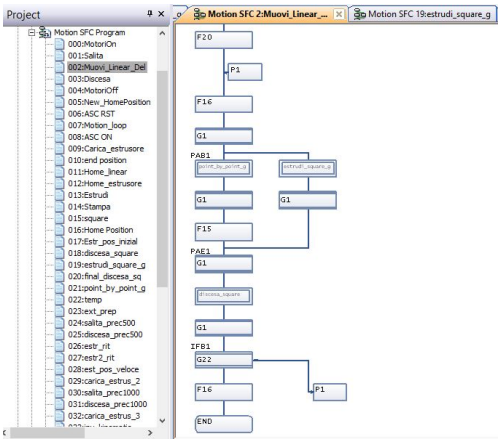


Figure 4.11: main program

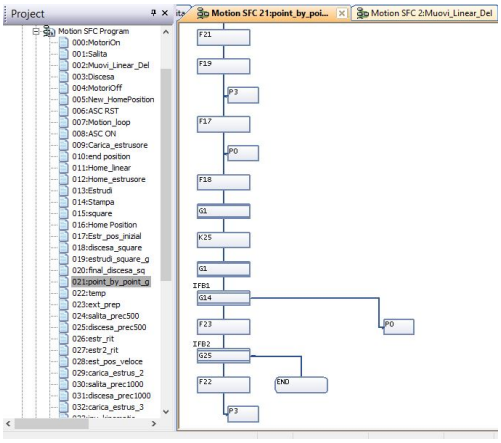


Figure 4.12: point by point

a singe layer run out, the extruder is stopped and the two sub-function are terminated. Before moving on, the layer counter is increased by one. The machine lowers accordingly to the input data, preparing for the new operation. Once the while-cycle is completed the extruder is set back to working condition.
Let's now go back and deepen the subroutines topic. The one that controls the Linear Delta starts by setting #2L equal to one in order to read the first useful point, #0L to the number of cams file defining the layer and by setting #20, #22, #24 (and #26 for the discontinuous case) equal to one to three (four). It then imposes D558L equal to zero (x-axis of the cam file at previous step). A new inner pseudo while-cycle is created until all the cams have been considered, that first reads one of them to get the number of point, for that batch. A second inner loop is responsible for the reading of each point and their execution. Here is the code for the discontinuous case, that simply differs by an *if* statement:

$CAMRD\#20, \#10, \#10 + K1, \#100$
$CAMRD\#22, \#10, \#10 + K1, \#200$
$CAMRD\#24, \#10, \#10 + K1, \#300$
$CAMRD\#26, \#10, \#10 + K1, \#400$
$\#10L = \#10L + K1$
$D556L = \#104L - D558L$
$D558L = \#104L$
$IF\#406L == K0$
$SETM3260$ $//stop\ ext$
$ELSE$
$RSTM3260$
$IEND$

The information is then fed to the motors and the variables are updated to complete the loops. The second subroutine just moves the extruder with constant velocity or the one stored in the fourth cam.

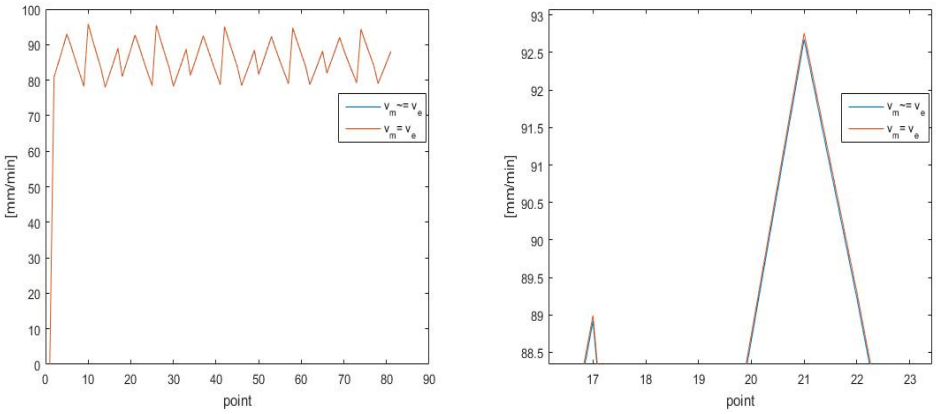


Figure 4.13: $v_{ext} = v_{mean}$ vs $v_{ext} \neq v_{mean}$ for 4x4 square with 3.1x3.1 [cm] inner hole, geometrical initial setting and $v_{ext} = 2.5\ mm/s$

4.5.4 Additional consideration

Trapezoidal velocity profile:

The Motion will move from two consecutive points by means of a trapezoidal velocity profile. Thus to assure at least same mean velocity in the interval as the one intended for that segment (first approximation of the printing process is done with constant velocity) a few passages are required:
$v_{ext} \neq v_{mean}$ (constant velocity):

$$P_{max} = \max(\triangle alpha_i) \quad v_i = \frac{P_{max}}{t_i}$$

$v_{ext} = v_{mean}$:

$$t_i = \frac{v_i}{a_m} + \frac{v_i}{d_m} + \frac{\triangle s}{v_i} \tag{4.3}$$

$$ds_i = \triangle S_i - 0.5 a_m (\frac{v}{a_m})^2 - 0.5 d_m (\frac{v}{d_m})^2 \tag{4.4}$$

$$\triangle = (\frac{2}{3} a_m t_i)^2 - \frac{4}{3} P_{max} a_m$$

$$v_i = \frac{2}{3} a_m t_i - \sqrt{\triangle} \quad \triangle \geqslant 0$$

$$v_i = \sqrt{\frac{4}{3} a_m P_{max}} \quad \triangle < 0$$

Where $\triangle$ determines if after the ramp up a constant velocity segment is required or not before deceleration. See Figure 4.13.

Deposition check:

This brief inquiry is carried out for the full motion law case and with equal length of the links. It's supposed that the extruded material is able to flow though the nozzle without friction. Once evaluated the velocity at the end effector, after applying the motion law, by means of the Jacobian matrix, it's possible to reconstruct the in-plane resultant of the x-y-z velocity vector .Then the diameter of the deposited filament is calculate.

$$d_{dep} = \sqrt{\frac{v_{ext}(2r)^2}{v_{plate}}}$$

This approach could lead to numerical error (Figure 4.15), one major fault is due to v_{plate} that may be equal to zero leading the solution to infinity, this is the reason way many firmware uses the inverse for the computation instead. Here are the results for ideal match in velocity on both sides, but with a trapezoidal motion law ($\xi_v = \frac{1}{3}$) on the rail and a constant velocity on the extruder. It also give a representation of the error incurred when approximating a constant velocity profile (i.e continuous deposition) with a trapezoidal one without any correction (Figure 4.14).

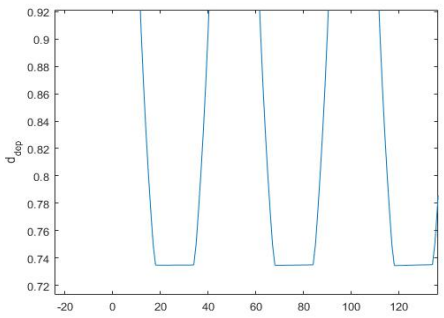


Figure 4.14: variation of the diameter deposited: motion law mismatch. $d_{nozzle} = 0.9mm$

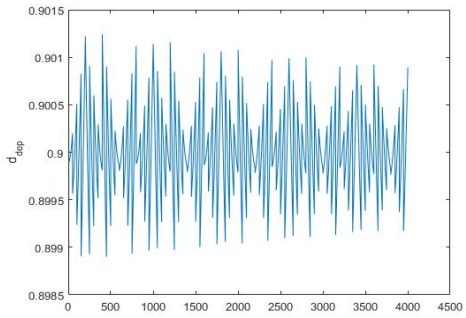


Figure 4.15: variation of the diameter deposited: motion law match. $d_{nozzle} = 0.9mm$

4.6 Manual control

Up to now, the controller was used to merely execute the commands obtained by means of other software. It'll be helpful to have any away to input and move the machine directly from the HMI. The motion controller requires to solve the inverse kinematics for any input, that is obtained by calculating the matrix product ahead.
Here is presented the program in the alpha-testing phase. The input process begins by pressing the start button on the HMI that evaluates the inverse problem from the start position and save the value in the memory block where the previous state is saved. The program is based on an in-plane movement so there's no z-axis dependency. To change the X,Y value several increment are available that are made accessible to the

motion by pressing them on the HMI. Until a new value is inserted the system will use that value to determine the final position. Different from other philosophy where by pressing the incremental direction ($\pm$ *x or y*) the machine moves, here it allows to set the final value and then operate the machine. This could prove useful in testing the robot; by combining the increment values and directions the user, helped by the display, can input the target position and by pressing the "move" button control the robot.
The programs behind these concepts are quite straight forward. The incremental values only set a memory cell equal to a particular number, that gets rewritten when a new button is pressed. The incremental directions only add or subtract that value in an other memory cell where the target position is stored. The move command uses the new value to solve the kinematics and feeds the value subtracted by the previous position to the motor (data might be modified and analyzed by the *data processing* block). Before completing the task, it substitutes the old position with the one reached.
In order to display the value on the HMI the value is converted in $10^{-2}mm$, allowing the selection of values of the order of the tenths of millimeters. In the Motion, the incremental value needs to wait a given time before completing the task using *TIME K2000* (2sec) in a wait block. This is done to avoid an over selection of that variable since, during the pushing of the button, the PLC and the Motion have time to complete several cycles. During the target selection, the input values are bounded to $\pm 10cm$.

4.6.1 Memory allocation

Here are presented the memories used by the Motion unit. The differentiation was done with no particular reason, if not to separate different objectives.
memory:

#500F	l_1	#536F	-	#566F	increment
#504F	l_2	#540F	q2	#570L	q_1 old
#508F	l_3	#544F	-	#572L	q_2 old
#512F	$R_p - s$	#548F	q3	#574L	q_3 old
#516F	p_x	#552F	-	#580F	temporary px
#520F	p_y	#556F	-	#584F	temporary py
#524F	p_z	#560L	$q_1(10^{-1}\mu m)$	#590F	input px
#528F	-	#562L	$q_2(10^{-1}\mu m)$	#594F	input py
#532F	q1	#564L	$q_3(10^{-1}\mu m)$		

D memory:

D500L	x to HMI	D504L	$\triangle q_1$	D508L	$\triangle q_3$
D502L	y to HMI6	D506L	$\triangle q_2$		

The PLC sees the D memories to HMI in D2024L and D2026L.

4.6.2 Inverse kinematics for the Motion software

The equations are obtained by developing the standard formulation of the inverse kinematics in order to reduce the evaluating time and memory consumption. The case for

different length of the links is the one implemented, if also R_b and s needs to vary from link to link modify the code accordingly.
The function requires the position in meters and saves the guide displacement from the home position in 10^{-7}m.

$$d_i = p + (R_p - s)\bar{u}_i = [a_i; b_i; c_i]$$

$$q_i = c_i - \sqrt{l_i^2 - a_i^2 - b_i^2}$$

example (i.e. q_2) :
$\#552F = (\#520F + \#512F * float(K0, 86603)) * (\#520F + \#512 * Ffloat(K0, 86603))$
$\#536F = \#504 * F\#504F - (\#516F - \#512Ffloat(K0, 5)) * ...$
$...(\#516F - \#512 * Ffloat(K0, 5)) - \#552F$
$\#540F = \#524F - SQRT(\#536F)$
The data format (#xxxF), as can be seen, is different from the one of the "CNC emulator". Here the presence of the square root function force us to use a 64-bit floating point type of data, instead of the 32-bit integer that works more than fine when dealing with linear guide positions.

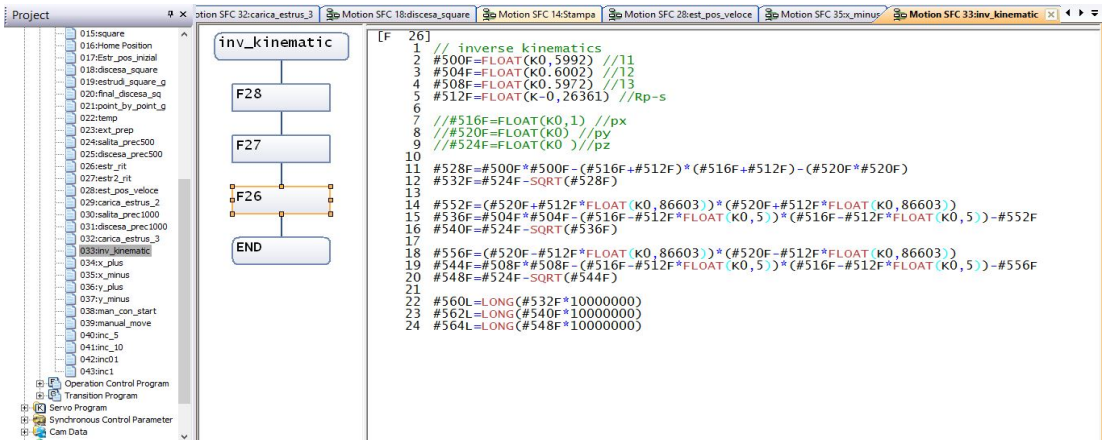


Figure 4.16: inverse kinematics for the motion software

4.6.3 Direct kinematics for the Motion software

Direct kinematics is not required for the manual control, but, since in this section the opposite case has been presented, it's included in this subsection. It was developed keeping in mind future run of the robot where users might be interested to control the machine also knowing the error due to approximation or simply infer the position of the end effector from the measurements of the encoders. This could prove effective for the assessment of the starting position, but also to check every fixed amount of points if any approximation in the control chain has deviated the behavior from the one intended of a given small tolerance. This procedure should be carried out with particular attention since this function itself could introduce approximating errors.

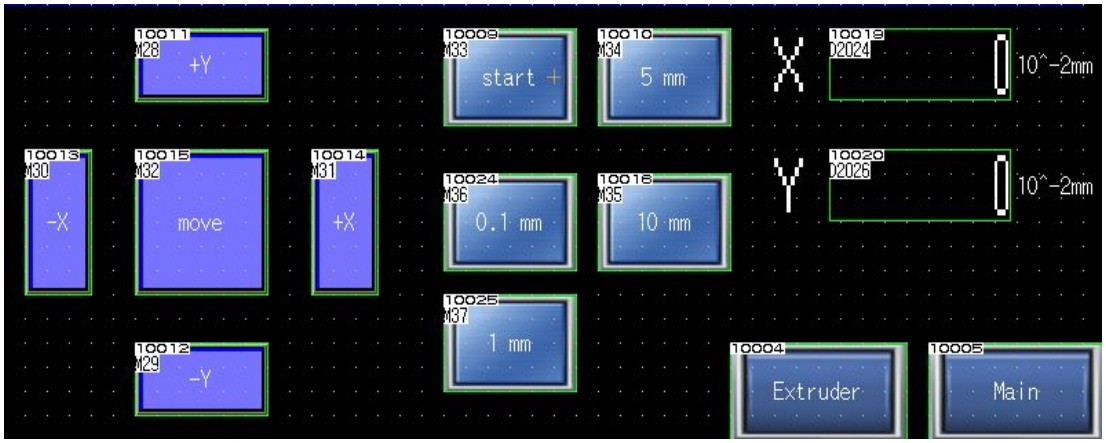


Figure 4.17: HMI layout for the manual control

#600F	C	#604F	D	#608F	E	#612L	F
#616F	K_1	#620F	K_2	#624F	K_3	#688F	$q_1(m)$
#692F	$q_2(m)$	#696F	$q_3(m)$				

The nomenclature in the table refer to relative paragraph in the Mechanical System chapter.
Omitting the input data:

$$\#600F = 2 * \#688F - \#692F - \#696F \quad //C$$

$$\#604F = -2 * \#688F * \#688F + \#692F * \#692F + \#696F * \#696F + 2 * \#500F * \#500F - \#504F * \#504F - \#508F * \#508F \quad //D$$

$$\#608F = \#692F - \#696F \quad //E$$

$$\#612F = -\#692F * \#692F + \#696F * \#696F + \#504F * \#504F - \#508F * \#508F \quad //F$$

$$\#616F = ((\#600F * \#600F/3) + \#608F * \#608F)/(3 * \#512F * \#512F) + K1$$

$$\#620F = (\#600F * \#604F + 3 * \#608F * \#612F)/(9 * \#512F * \#512F) + 2 * \#600F/3 - 2 * \#688F$$

$$\#624F = \#604F * \#604F/(36 * \#512F * \#512F) + \#612F * \#612F/(12 * \#512F * \#512F) + \#604F/3 + \#512F * \#512F + \#688F * \#688F - \#500F * \#500F$$

$$\#628F = \#620F/(2 * \#616F)$$

$$\#632F = \#624F/\#616F$$

$$\#640F = -\#628F + SQRT(\#628F * \#628F - \#632F) \quad //Pz$$

$$\#644F = (2 * \#600F * \#640F + \#604F)/(6 * \#512F) \quad //Px$$

$$\#648F = (2 * \#608F * \#640F + \#612F)/(2 * SQRT(K3) * \#512F) \quad //Py$$

4.6.4 Data processing and on-line control

The possibility to solve the kinematics equations inside the motion controller unit opens a new field of inquiry. If up to now, all the solutions were introduced as a mean to reduce point number or simplify the procedure for an not expert user, here all of these points could be minimized or even wiped out. If a communication bus between the PLC and a computer can be established, then it would be possible to transfer data like position and velocity directly from a *G-code* file (or by means of an interpreter). If this

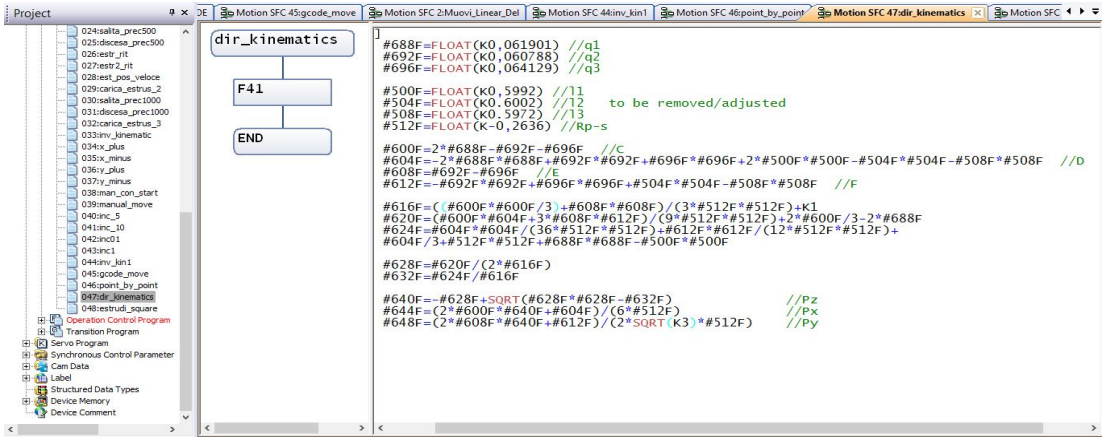


Figure 4.18: implementation of the direct kinematics in the motion software

is the case, a new program running in the motion is required. This could be also used in combination with the method previously introduced (uploading of cams), but letting the Motion split up the travel value and solve the inverse kinematics. As can be seen in the *control outline*, the information requires to be analyzed before being executed. This step is required for both manual control and on-line control. The difference between the two of them is how the information is fed to the control system. Here is the main outline:

#400F	$\triangle x$	#474F	Z_i	#416F	n	#432F	Y_j
#404F	$\triangle y$	#420F	$\triangle x_k$	#450F	X_{i-1}	#436F	Z_j
#482F	$\triangle z$	#424F	$\triangle y_k$	#454F	X_i	#490L	j
#408F	$\sqrt{\triangle x^2 + \triangle y^2 + \triangle z^2}$	#478F	$\triangle z_k$	#460F	Y_{i-1}	#492F	$\triangle l_j$
#412F	$\triangle L_{max}$	#428F	X_j	#470F	Z_{i-1}	D510L	v_{j1}

1. reads new target position and knowing the starting one, calculates (also for y and z-axis):

$$\#400F = \#454F - \#450F$$

2. it then evaluates the in the working space

$$\#408F = SQRT(\#400F * \#400F + \#404F * \#404F + \#482F * \#482F)$$

3. fixed $\triangle L_{max}$, it checks in how many intervals split the dL. It then sets j variable for the inner loop.

$IF \#408F > \#412F$
$\#416L = LONG(FUP(\#408F/\#412F))$
$ELSE$
$\#416L = K1$
$IEND$

4. inside the loop, it determines the increase value from the starting point for that iteration. It evaluates the target position for x, y and z, the time, for a given velocity (obtained in the F command or custom), to reach that position.

$$\#420F = \#400F/FLOAT(\#416L)$$
$$\#428F = (\#450F + \#420F * \#490L)$$
$$\#492F = SQRT(\#420F * \#420F + \#424F * \#424F + \#478F * \#478F)/\#496F[16]$$
$$\#490L = \#490L + K1$$

5. it then lunches a function that solves the inverse kinematics and the velocity for the linear delta.

$$D512F = FLOAT(D504L)/\#492F$$
$$D510L = LONG(D512F) * K60/K100$$

6. as it controls the guides, it moves the extruder according to the specification.

Continuous mode: The "CNC emulator" works by approximating the movement between two points with a trapezoidal motion law (exact stop mode). Such approximation can be overcome by means of few modification, that requires to be tested before being used inside a printing cycle, if a more continuous mode is required. This approach will be illustrated using only two control axises. Simulation in the Mitsubishi environment showed good result. This approach can only be implemented during interpolation between two consecutive point of the original G-code, unless there's no change in the direction of the movement of all the guides.

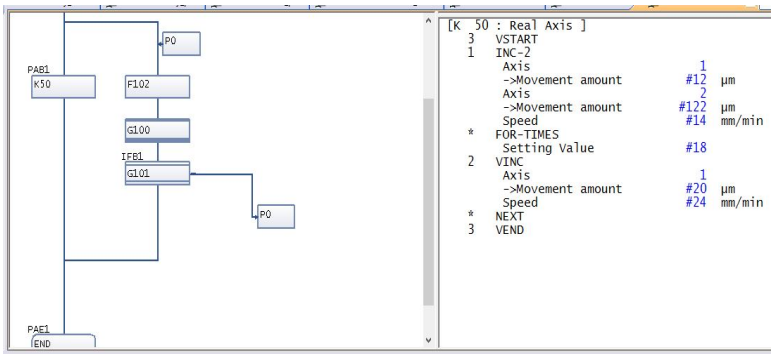


Figure 4.19: Continuous mode solution test

In the left branch the overall movement is split in different segment as many as they were obtained in the interpolation process. In the right part a if-case condition inside a for-loop cycle allows to select the proper interval in which the velocity stands. A wait condition allows to pace the feeding information to the left branch. The velocity should be defined along the resulting path: $V = V1\sqrt{d_1^2 + d_2^2 + d_3^3}/d_1$, where V1 is the velocity of the first linear guide. The time to wait should be reduced of a small percentage to allow the reading of the correct value in the left branch. Due to its dependency with the trajectory followed, it needs to be implemented only in the on-line control where data processing correctly determine the no-inversion condition. This procedure

[16] #496F is the velocity for that segment

can be useful if applied only to the extruder in order to guarantee a pseudo continuous deposition without any interruption between two real stop condition.

4.7 HMI and PLC programming

The 3D printer requires also an interface to select and coordinate the tasks. Such operation is done by means of a touch screen connected to the PLC unit.
The definition of the programs is required to be carried simultaneously helping to match the bit information unit. Once a variable inside the PLC is defined, assigning it to a specific memory unit, a pushbutton is created inside the interface related to that same unit (figure 4.20).

Figure 4.20: HMI and PLC programming example

The Ladder Diagram will feature that variable in one of its rungs controlling a particular Motion routine. A small window of the whole working program can be seen in figure 4.21 and 4.22.
The first version of the human-machine interface is made up of three pages. The first page was developed to feature all the main buttons that are: turning on/off of the motors, the homing command (the guides move back to the lower position obtaining the correct starting configuration again), water cooling and temperature module engagement pushbuttons and finally the printing start (the definition of the routines to run has to be changed inside the Motion, in future versions they might be separated in different blocks).
Four light indicators are placed in the upper side to show the state or alarms during the operation. The numerical display is activated only when using the synchronous control and it shows the cam-axis length that's being executed. In the lower right corner the reference to the other two pages are placed (they create an internal link to those screens).
The second page is the one dedicated to the extruder. Three load programs are available to transfer material from the first chamber to the extrusion one differing from their travel value. This may be useful to avoid reaching the maximum torque allowable; if this condition is met in spite of all that, and the punches are stuck, two programs are

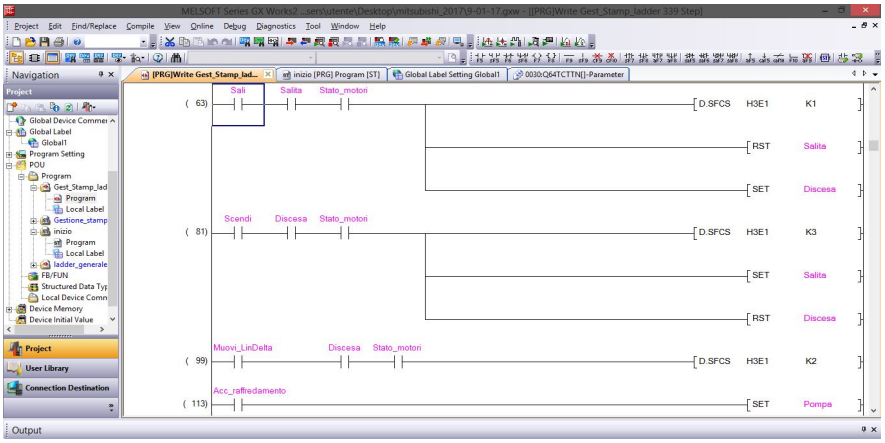


Figure 4.21: PLC working program example

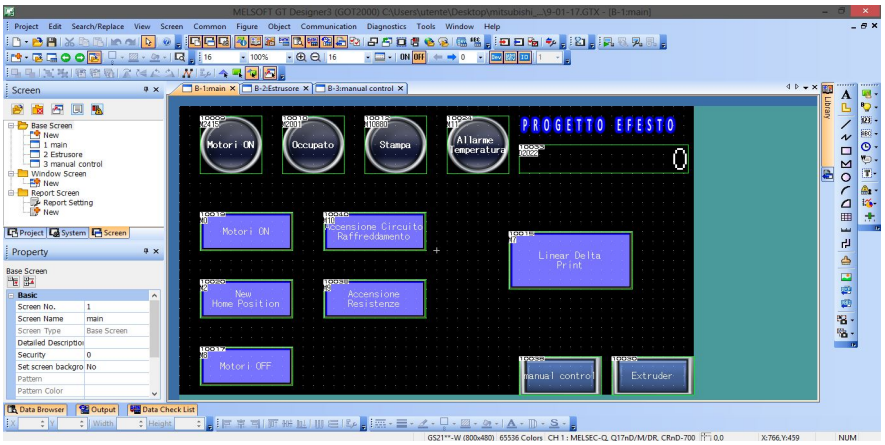


Figure 4.22: HMI working program example

available to force their retraction (*return* commands). The *extrusion* forces the emptying of the extrusion chamber fundamental step during the starting up of the printer and the cleaning of the extrusion chamber; similar command is the *extruder positioning* but the forwarding movement of the punch can be stopped and unstopped by repressing the same button (*alternate* type instead on *momentary*) designed to position the punch in contact with the material, making the system ready to extrude.
The final screen is related to the control of the Linear Delta (partially introduced in the *manual control* section). The additional pushbuttons to the ones of the testing manual control are related to the positioning of the moving plate in the z-axis direction. The main ones (lifting and lowering) increase the height of an acceptable value (26 cm for the first test, but it can be adjust according to experimental necessities), while the others offer the possibility to tune the height slightly, allowing a more delicate nozzle approach.

Chapter 5

PRINTING and EXPERIMENTAL RESULTS

5.1 Objective and Overview

In this chapter, all the previous information will be used to print. The first test is carried out using a generation trajectory algorithm that is briefly described (5.2). The algorithm is used to generate a hollow square with a continuous deposition approach (5.3) and parallel lines with changeable interfilament distance (5.4). The same programs can be used to create different 3D objects. Few consideration are presented for an oblique parallelepiped printing (5.6). The first results are outlined, marking the validation feedbacks of the control system programming defined in the previous chapter. In the end, the overall procedure from modeling to final object will be discussed differentiating from the *G-code* and *input equation* based case (5.8).

5.2 Trajectory generation

The ones used are parametric trajectory (S) because they can be represented by means of a parameter (λ). For this section refer to [7][10]. Then $S(\lambda)$ is defined as well as $\dot{S} = \left(\frac{\partial}{\partial\lambda}S\right)\dot{\lambda}$. In the previous thesis works, Bézier curves (lines and parabolas) were summed to obtain the final trajectory keeping the velocity constant. These, as the ones actually implemented and used for the first tests, are able to interpolate n-points into a desired path.
The linear motion is the one used for the first tests. Here is what can be found from literature: if a robot has to be moved from S_0 to S_1, then easily $S = S_0 + u\lambda$, where u is the vector direction. Both S_0 and u are constant, thus $\dot{P} = u\dot{\lambda}$ and $\ddot{P} = u\ddot{\lambda}$.
The programs that will later on be described only define the S_i (corner) points of the trajectory and normalized velocity vector. The interpolation is carried out in a second phase using a sampling space interval known to leave the trajectory bounded to the tolerance requirements. In addition to these general aspects, the programs presented

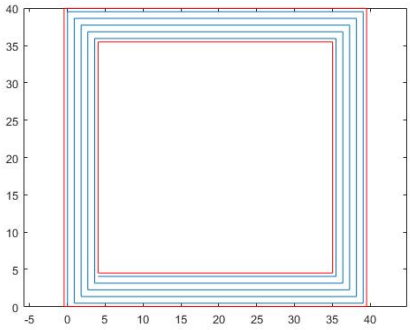


Figure 5.1: square with A=40mm, B=31mm, r=0,45mm

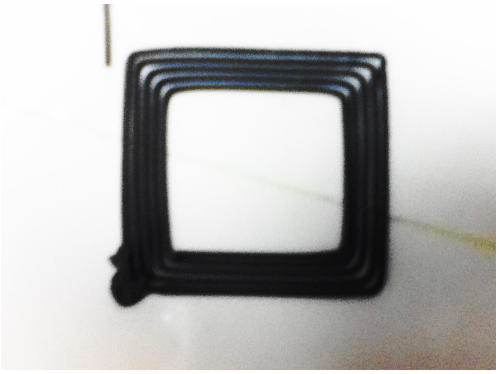


Figure 5.2: first printed result of an hollow square

create a normalized extrusion vector to resemble the one created from a slicer with a simple binary code to tell the two state of the extruder apart. The normalization allows to specify the trajectory parameters (velocity of the plate, filament dimension, interfilament gap and extrusion feed rate) without any change in the generation function. Once defined the geometrical parameters of the filament, theoretically, an infinity of ratios exist that solve such problem, so they are output still normalized, only later on the velocities are selected. These features will allow great flexibility in the testing phase.

5.3 Square generation

One of the first requests was the possibility to print a hollow cube. There were no program able to read the input file and move the extruder accordingly. So the simple way to test the machine and obtain the required object was developing a continuous deposition program. The idea was realizing a single layer without interruption. To do so any square needed to start from the center position and traveling around to complete the square, then, stopped the extruder, move back and down to center ready for a new layer (the only discontinuous movement). Thus a manual input was required, here are the equation that were used to describe the point of the square.

Square generation data:

A	outer dimension	r	adjacent segment gap	B	inner dimension	N[1]	$N = (A - B)/4r$

at the beginning of each turn:

$$As = A - r \quad ; \quad Bs = A - 2r \quad ; \quad a = r(i - 1)$$

1st turn:

$$x = [a, a, Bs - a, Bs - a] \qquad y = [a, As - a, As - a, ri]$$

[1] number of turns

others turn:

$$x = [a + r(i-1), a + r(i-1), Bs - a - r(i-1), Bs - a - r(i-1)]$$

$$y = [a + r(i-2), As - a - r(i-1), As - a - r(i-1), a + r(i)]$$

The geometry used for the first attempts where A=40 mm, B=31 mm and 2r=0.9 mm.

First test execution and result:

The very first test was conduct using the A equal to 40mm, but B to 30mm and r to 1 mm. The choice to use an higher gap was taken to account for unexpected behavior of the system in that early stage and positioning errors (right distance between plate and nozzle). The test is here presented because it shows perfectly the steps followed during the programming chapter.
Figure 5.2 shows the final result from the top view, while Figure 5.3 to 5.5 show the transition from one layer to the next. Once a layer is done, the Motion communicates the extruder motor to stop, activating the respective bit; the plate is then move back to the center ready to print the new layer without shedding a drop of material; the next layer is then start after resetting the motor stop flag.
This procedure is very important because it's the foundation algorithm used to execute the *G-code*. As can be seen from any slicing software, the machine will have many free movement to reset the position of the extruder. The execution of such step shows how the obtained *G-code* could be implemented without any particular drawback[2].
The program used was the one with velocity-displacement format using the "CNC emulator" with three cams, using a dL of 10 mm and a 2.5 mm/s work-space constant velocity (corrected with the trapezoidal method shown).

5.4 Interfilament gap script

In order to test the effect of the gap between two near extruded filaments, a few lines of coding were required. The script, that can be found in Appendix B (*line_close*), generates a series of segments according to the displacement imposed in the user input data part as it will be done for the square. The function can be easily expanded to house also the printed gap as a function of the z-axis gap making it also a possible script to test the effects of the layer thickness on this particular machine (calibrating the deposition process). This is an example of full user input discontinuous print (resembling a *G-code*).
The parameters are: *x0* and *y0* as starting position; *dl* as the maximum interpolating distance; l_{max} as the segment length; *dd* as the vector containing the interfilament gaps. This program can be adjusted to create a different infill and filament analysis, in order

[2]the temperature at the nozzle plays a crucial role here, if too high the material will drop down causing imperfections, if too low the material will take some time before being able to properly flow.

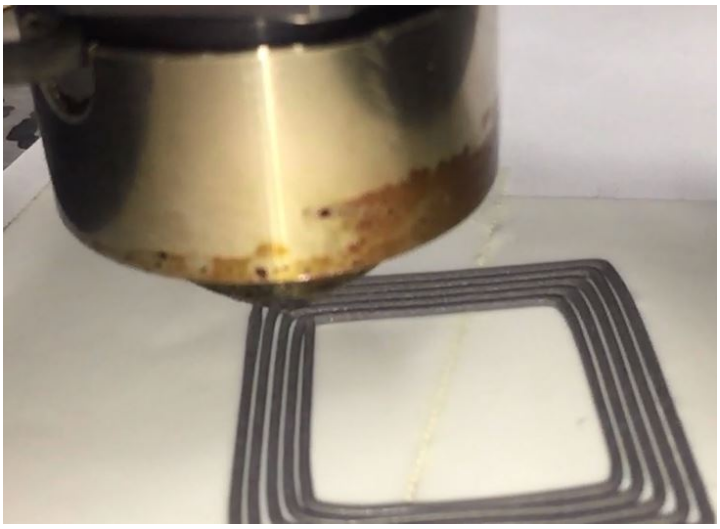


Figure 5.3: example of line breaking from one layer to the next: stop

Figure 5.4: example of line breaking from one layer to the next: rapid movement

Figure 5.5: example of line breaking from one layer to the next: restart

to carry out new initial test on the material and machine performance. The applications can be found in figures 5.7 and 5.8.

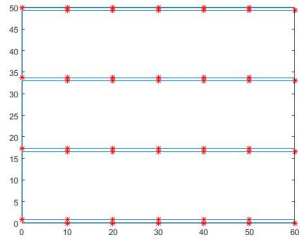


Figure 5.6: interfilament gap script example with different gap values

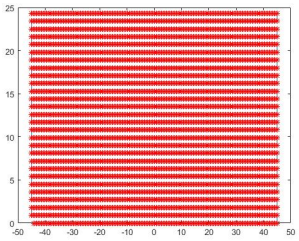


Figure 5.7: rectangular layer of 90x25 mm with interfilament of 0.9mm

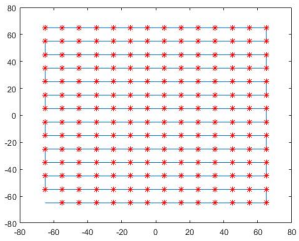


Figure 5.8: 130x130mm square with interfilament of 10mm

5.5 Cube

In the previous section one layer was drawn. The motion control program was developed giving the opportunity to select the number of layers and the return position to allow flexibility on the final object specifics. As it can be seen in the Matlab code, the return movement was split in half in order to separate the layer height from the centering position (this will be helpful to test with layer heights as well as in the next section). These values need to be placed manually on the motion controller or using a version completely automated.

5.6 Oblique parallelepiped

One of the main limitation of the machine (as a 3D printer) that is required to be known, is the maximum (or minimum depending on the point of view) taper angle. Realization of inclined walls requires understanding of the necessity to or not to use temporary reinforcement to sustain the construction. To do so, a hollow oblique parallelepiped was required to be printed. Calling θ the taper angle, supposing that it lies in the x-z plane and leads to small variation on the x-axis (otherwise it requires to solve the inverse kinematics also for new points since the travel value of the guides are due only to the start and end condition and not to the working space displacement), a relationship between them can be found:

$$d_x = \frac{d_z}{\tan\theta}$$

If high x-axis variation are required use a *G-code* approach.

5.7 Generic object

If no particular equations are available, as occurs the majority of the time, the only path is to draw the object with any CAD software and, using a slicer, obtain the relative *G-code* and following one of the approaches previously explained. Refer to *Printing process outline:G-code* for the exact procedure.
Here are presented a series of picture representing the realization steps of a specimen for bending test of 65x11x11 mm (square section).

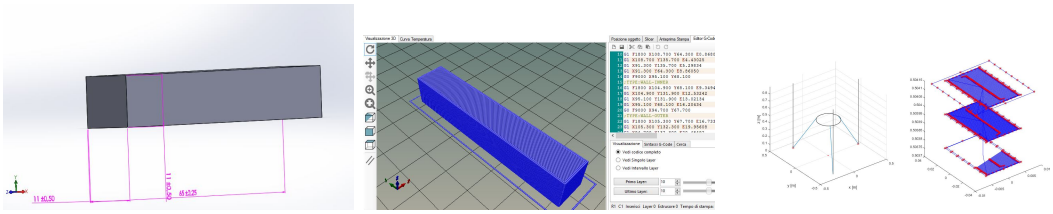


Figure 5.9: parallelepiped of 65x11x11 mm: from CAD to G-code reader output

5.8 Printing process outline

In this final section the procedure from designing to final object removal from the plate will be presented. To allow a better understanding of the topic as of all the codes written, two cases will be distinguished. This outline is intrinsically linked to the version of the system in this thesis work. It requires to be updated when future version of the programs will be available (all the function and script are in the alpha testing, beta-testing releases will be a development and optimization of such initial elements)

5.8.1 G-code

1. draw object on any CAD,CAM software that support *.stl*(stereolithography) file exportation. Make sure to well define (parametrically) any sketch or function.

2. save the part in the *.stl* format.

3. open it with any slicing engine (CURA, Slic3r). The following programs were designed using CURA (if using Repetier-Host make sure to follow the relative instruction).

4. if not already done, set the machine parameters. For our robot, the only one that need to be checked are: Nozzle size (0.9mm), Initial layer thickness (depend on the quality and adhesion of the first layer, no more than 3/4 of nozzle), Travel speed (check limit on the rail, for first tests 5mm/s was used), Layer height (lower or same as the initial layer thickness), Shell thickness (no less than twice the nozzle) and fill density.

5. run the slicing operation and save it in the *G-code* format.

6. unless changing some line in the Matlab code, open the file just saved and remove the empty line (that will cause a break in the layer read sub-function). Before the last G0 command, place the commented layer definition as if were the last layer [3].

7. save it without changing termination. Open Matlab script *MAIN_Gcode.m*.

8. in the setting part, adjust: Robot geometry, chosen layer partition, plotting option, the intermediate point interval, number of point per cam file (2000) and the saving name.

9. run the script. A window pops up asking the file to open. Select the one for the case interested in.

10. the program will ask several questions by means of a menu. A first request is either use all the layers (*G_code_layer1* function) or a smaller set. If plotting option flag is set to one, the menu will ask what kind of initial representation is required 2D or 3D. A second pop up window is related to a single layer, point by point, drawing showing if the extruder is on or not, use it to check the result.

11. it then asks what saving format has been chosen.

12. make sure that the correct ladder diagram and HMI program are uploaded on the control system (correct temperature. See guide for extrusion preparation). Open the relative *.mtw* file (at least the latest). On the left side, the program tree can be found. Select *Cam Data*. Right click on *No.001*, then click on *import*. Choose the first data file. Repeat the operation for all the files making sure to upload them in the correct order (use the name as guide).

13. compile the program and upload it on the motion controller.

14. on the HMI, select in the main page in this order: motor on, home position, cooling system on and resistance on.

15. in *GX Works2*, check that the temperature are reached and sustained according to the guide to prepare the extruder.

16. completed it, place the plate in contact with the extruder (or leaving a small gap). In the HMI push *Move Linear Delta*.

17. intermediate stops may be in order to refill the extruder chamber.

18. upon completion, remove object from the plate.

[3] it can be copied in the first row, as the total number of layers can be found

5.8.2 Manual input

1. open *MAIN_square.m*. Set robot geometry, square data (or for any other custom function), extruder velocity, maximum inter-point distance, number of point defining motion law and any other specification, saving title, next layer start (layer thickness, thus filament diameter). Below *%bazier_ret_par data*, it's possible to prepare the program *ret_per_bazier_script*.

2. use square generation or any custom function (use the first as reference) to create coordinate points.

3. run the *.m* script. A first option asks if either the Bazier approach or a motion law assignment is preferred.

4. select the first one and the bazier script is launched.
   Select the second one and a new pop up questions about the type of motion law desired. If none of them is desired, just select the first one.

5. plot options are available for debug.

6. a save menu opens. Upon pressing 'yes' a new option is given, continuing with synchronous control (master cam-displacement) or using the new solution (velocity-displacement).

7. selected the second, full motion law or full motion law with differences already calculated or one point every dL. A third saving menu opens asking if using constant velocity between two consecutive points (leading to some variation in the result) or directly a corrected velocity in order to constraint the execution time once performed by the motion controller (mean velocity is equal to the one with constant velocity).

8. the file is saved. Open the relative *.mtw* file (at least the latest).On the left side, the program tree can be found. Select *Cam Data*. Right click on *No.001*, then click on *import*.Choose the first data file. Repeat the operation for all the files making sure to upload them in the correct order (use the name as guide).

9. in the Matlab *Command Window* , Extruder velocity, *delta_pos1* and *delta_pos2* are found. The first needs to be placed in *K23* (speed), the three values of the second one in *K22* (in the same order), the last three in *K27* (they are the inter-layers distance).

10. compile the program and upload it on the motion controller.

11. on the HMI, select in the main page in this order: motor on, home position, cooling system on and resistance on.

12. in *GX Works2*, check that the temperature are reached and sustained according to the guide to prepare the extruder.

13. completed it, place the plate in contact with the extruder leaving a small gap. In the HMI push *Move Linear Delta* .

14. intermediate stops may be in order to refill the extruder chamber.

15. upon completion, remove object from the plate.

5.8.3 Extrusion system preparation

1. open the *.gxw* file. In *Tool* → *Intelligent Function Module Tool* → *Temperature Control Module* → *Auto tuning*, temperature can be set and checked.

2. control that the information reflects the one for the material that is being printed. Upload.

3. raise the robot using *Main Raise* in the HMI, adjust the height using other *Raise* or *Lower* to give enough room to prepare the printing system.

4. move the plate with *Manual Control* to work in outer region, if required.

5. place some material in the first chamber and wait a few minutes, let it warm up, then use a *Loader* button to transfer material from one chamber to the other. Be careful on the selection, longer distance could prove unbearable for the system (it reaches max torque capability leading to a fault).

6. push *Extrusion* to completely empty the main chamber.

7. repeat the two previous steps until the machine has been cleaned and the filament is made of the printing material only (some plastic residue from the cleaning procedure is still present in the first chamber).

8. use *Extruder Positioning* to place the punch in contact with the material. Once the flow is established repress the same key to stop the moving.

9. collocate the extruder in the center position.

Conclusion

This master thesis deals with the development and adjustment of an automated machine composed by two subsystems, a linear delta robot with 3 Dofs, and an extruder for MIM technology. The whole machine is a 3D printer for metal components based on the use of a MIM extruder contrary to typical 3D printers based on the use of laser or EBM. The project aims at the realization of a machine capable to provide the characteristic features of a 3D printer in order to permit the study and development of this new technology. The machine must be capable to ensure a manufacturing process inside precise technological constraints.
In this thesis work the primary objective was to permit the mastering of the printing process in order to study and develop the related technology.
In order to do so, some mechanical problem as well as electric and control related ones needed to be addressed.
The structure required to be made more rigid and at the same time feature elements to assure normality of the nozzle-moving plate interaction. This component was designed, realized and placed on the moving plate. An aimed analysis of the equation describing a general configuration were carried out to outline the possible sources of unexpected behavior. To correct such errors a calibrating script was developed that simulates the actual system, due to assembly delays such algorithm was not tested up to now.
The electrical cabinet fitting analysis was initially developed, but later on it was commissioned to an external factory. Either way, a solution has been delineated and the electrical layout transcribed to feature all the components used up to now.
As concerns the programming part, a new, for this particular machine, control method was developed to reduce memory consumption and to make it as close as possible to any other commercial 3D printer, allowing any user, with basic knowledge on such systems, to at least understand what the machine is actually carrying out. This required to differentiate between final usage of the machine and what was required in the early stages. A manual control was developed to directly operate the machine from the touch screen, in order to set the moving base during the material preparation before printing. Such program, was swollen up to directly process the data on the Motion itself.
All of this helped, finally, to print the first basic object. A single layer square was more than enough to test the CNC emulator developed with its versatility to control the printing process either by means of continuous or discontinuous deposition. Future and/or possible development:

1. further development of this technology
2. realization of a communication bus between a computer and the control unit to easily transfer data.
3. use the calibration tool to enhance the performance of the machine as a whole.
4. development of a system to track the position of the Linear Delta end effector and to be able to use the information to close a controlling loop.
5. measure increased rigidity.
6. Check feasibility to use heated flexible tube to connect the extruder to the moving plate, featuring now the actual nozzle, that will print on the base of the machine or a properly set bed. This solution could exploit all the advantages of a PKM machine, reducing the problem of a moving printed object.

Appendices

Appendix A

Technical sheets and drawings

Here are presented the technical data of the electromechanical and mechanical components used in the project [8]. The technical data of the designed components as well as the electrical layout and specification can be found.

A.1 Mitsubishi HG-KR43B electrical motor technical sheet

SERVOMOTORS AND AMPLIFIERS

Servo Motor Selection (Example Part No. = HG-KR053BG1)
Not all options available for every motor.

HG-□ □□□□□□□

Symbol	Shaft End
None	Standard (Straight shaft)(*7)
K	Key shaft (with/without key)(*8)
D	D-cut shaft (*8)

Symbol	Reducer (*5)
None	None
G1	With reducer for general industrial machines, flange mounting
G1H	With reducer for general industrial machines, foot mounting (*6)
G5	With flange-output type reducer for high precision applications, flange mounting
G7	With shaft-output type reducer for high precision applications, flange mounting

Symbol	Oil Seal
None	None
J	Installed (*2, *3, *4)

Symbol	Electromagnetic Brake
None	None
B	Installed (*1)

Symbol	Rated Speed [r/min]
1	1000
2	2000
3	3000

Symbol	Rated Output [kW]
05	0.05
1 to 8	0.1 to 0.85
10 to 70	1.0 to 7.0

Symbol	Inertia/Capacity
HG-KR	Low inertia, small capacity
HG-MR	Ultra-low inertia, small capacity
HG-SR	Medium inertia, medium capacity

Notes:
1. Refer to electromagnetic brake specifications of each servo motor series in this catalog for the available models and detailed specifications.
2. Available in 0.1 kW or larger HG-KR/HG-MR series and all HG-SR series.
3. Oil seal is not installed in the geared servo motor.
4. Dimensions for HG-KR/HG-MR/HG-SR series with an oil seal are different from the standard models. Contact your local sales office for more details.
5. Refer to "Geared Servo Motor Specifications" in this catalog for the available models and detailed specifications.
6. Available only in HF-SR 2000 r/min series.
7. Standard HG-SR G1/G1H has a key shaft (with key).
8. Refer to special shaft end specifications of each servo motor series in this catalog for the available models and detailed specifications.

Stocked Motors

Model Number	Model Number
HG-KR053(B)	HG-SR52(B)
HG-KR13(B)	HG-SR102(B)
HG-KR23(B)	HG-SR152(B)
HG-KR43(B)	HG-SR202(B)
HG-KR73(B)	HG-SR352(B)
HG-KR053(B)D	HG-SR-502(B)
HG-KR13(B)D	HG-SR702(B)
HG-KR23(B)K	HG-SR52(B)K
HG-KR43(B)K	HG-SR102(B)K
HG-KR73(B)K	HG-SR152(B)K
HG-MR053(B)	HG-SR202(B)K
HG-MR13(B)	HG-SR502(B)K
HG-MR23(B)	HG-SR702(B)K
HG-MR43(B)	
HG-MR73(B)	
HG-MR053(B)D	
HG-MR13(B)D	
HG-MR23(B)K	
HG-MR43(B)K	
HG-MR73(B)K	

Combinations of Rotary Servo Motor and Servo Amplifier
With MR-J4 Servo Amplifier

Rotary Servo Motor			Servo Amplifier
HG-KR	HG-MR	HG-SR	
053, 13	053, 13	-	MR-J4-10A/B
23	23	-	MR-J4-20A/B
43	43	-	MR-J4-40A/B
-	-	51, 52	MR-J4-60A/B
73	73	-	MR-J4-70A/B
-	-	81, 102	MR-J4-100A/B
-	-	121, 201, 152, 202	MR-J4-200A/B
-	-	301, 352	MR-J4-350A/B
-	-	421, 502	MR-J4-500A/B
-	-	702	MR-J4-700A/B

With MR-J4W2 Servo Amplifier

Rotary Servo Motor			Servo Amplifier	Axis (*1)
HG-KR	HG-MR	HG-SR		
053, 13, 23	053, 13, 23	-	MR-J4W2-22B	A/B
053, 13, 23, 43	053, 13, 23, 43	-	MR-J4W2-44B	A/B
43, 73	43, 73	51, 52	MR-J4W2-77B	A/B
43, 73	43, 73	51, 81, 52, 102	MR-J4W2-1010B	A/B

With MR-J4W3 Servo Amplifier

Rotary Servo Motor			Servo Amplifier	Axis (*2)
HG-KR	HG-MR	HG-SR		
053, 13, 23	053, 13, 23	-	MR-J4W3-222B	A/B/C
053, 13, 23, 43	053, 13, 23, 43	-	MR-J4W3-444B	A/B/C

Notes:
1. Any combination of the servo motors is available such as rotary servo motor for A-axis, and linear servo motor or direct drive motor for B-axis. Refer to "Combinations of Linear Servo Motor and Servo Amplifier" and "Combinations of Direct Drive Motor and Servo Amplifier" in the MR-J4 brochure.
2. Any combination of the servo motors is available such as rotary servo motor for A-axis, linear servo motor for B-axis, and direct drive motor for C-axis. Refer to "Combinations of Linear Servo Motor and Servo Amplifier" and "Combinations of Direct Drive Motor and Servo Amplifier" in the MR-J4 brochure.

HG-KR Series (Low Inertia, Small Capacity) Specifications

Servomotor Model HG-KR_		053(B)	13(B)	23(B)	43(B)	73(B)
Servo Amplifier Model	MR-J4-_	Refer to 'Combinations of Servo Motor and Servo Amplifier' in this guide.				
	MR-J4W_-_					
Power Supply Capacity (kVA) (*1)		0.3	0.3	0.5	0.9	1.3
Continuous Running Duty	Rated Output (kW)	5.0	100	200	400	750
	Rated Torque (N•m) (*3)	0.16	0.32	0.64	1.3	2.4
Maximum Torque (N•m)		0.56	1.1	2.2	4.5	8.4
Rated Speed (r/min)		3000				
Maximum Speed (r/min)		6000				
Permissible Instantaneous Speed (r/min)		6900				
Power Rate Continuous Rated Torque (kW/s)	Standard (kW/s)	5.63	13.0	18.3	43.7	45.2
	With Electromagnetic Brake (kW/s)	5.37	12.1	16.7	41.3	41.6
Rated Current (A)		0.9	0.8	1.3	2.6	4.8
Maximum Current (A)		3.2	2.5	4.6	9.1	17.2
Regenerative Braking Frequency (times/min) (*2)	MR-J4- (times/min)	(*4)	(*4)	453	268	157
	MR-J4W_- (times/min)	2540	1370	451	268	393
Moment of inertia J ($x10^{-4}$kg•m^2) [J (oz•in^2)]	Standard	0.0450	0.0777	0.221	0.371	1.26
	With Electromagnetic Brake	0.0472	0.0837	0.243	0.393	1.37
Recommended Load/Motor Inertia Moment Ratio		15 times or less		24 times or less	22 times or less	15 times or less
Speed/Position Detector		Absolute/incremental 22-bit encoder (resolution: 4194304 pulses/rev)				
Oil Seal		None	None (Servo motors with oil seal are available. (HG-KR_J))			
Insulation Class		130 (B)				
Structure		Totally enclosed, natural cooling (IP rating: IP65) (*2)				
Environment	Ambient Temperature	0 °C to 40 °C (non-freezing), storage: -15 °C to 70 °C (non-freezing)				
	Ambient Humidity	80% RH maximum (non-condensing), storage: 90% RH maximum (non-condensing)				
	Atmosphere	Indoors (no direct sunlight); no corrosive gas, inflammable gas, oil mist or dust				
	Elevation / Vibration (*5)	1000 m or less above sea level; X: 49 m/s² Y: 49 m/s²				
Vibration Rank		V10 (*6)				
Permissible Load for the Shaft (*5)	L (mm)	25	25	30	30	40
	Radial (N)	88	88	245	245	392
	Thrust (N)	59	59	98	98	147
Weight kg	Standard	0.34	0.54	0.91	1.4	2.8
	With Electromagnetic Brake	0.54	0.74	1.3	1.8	3.8

Notes:
1. Contact your local sales office if the load to motor inertia ratio exceeds the value in the table.
2. The shaft-through portion is excluded. IP67 for the servo motor with oil seal. Equivalent to IP44 for the reducer portion on the geared servo motor. Refer to this guide for the shaft-through portion.
3. When unbalanced torque is generated, such as in a vertical lift machine, it is recommended that the unbalanced torque of the machine be kept under 70% of the servo motor rated torque.
4. When the servo motor decelerates to a stop from the rated speed, the regenerative frequency will not be limited if the effective torque is within the rated torque range.
   When the servo motor decelerates to a stop from the maximum speed, the regenerative frequency will not be limited if the following requirements are met.
   • HG-KR053(B): The load to motor inertia ratio is 8 times or less, and the effective torque is within the rated torque range.
   • HG-KR13(B): The load to motor inertia ratio is 4 times or less, and the effective torque is within the rated torque range.
5. The vibration direction is shown in the diagram below. The numeric value indicates the maximum value of the component (commonly the bracket in the opposite direction of the motor shaft). Fretting of the bearing occurs easily when the motor stops, so maintain vibration to approximately one-half of the allowable value.

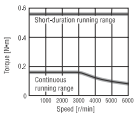
6. Refer to the MR-J4 Servo Amplifier and Motors brochure for more detailed specifications.

HG-KR Series Electromagnetic Brake Specifications (*1)

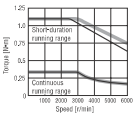
Servomotor Model HG-KR_		053B	13B	23B	43B	73B
Type		Spring actuated type safety brake				
Rated Voltage		24 VDC^{-10} 0%				
Power Consumption (W) at 20 °C		6.3	6.3	7.9	7.9	10
Electromagnetic Brake Static Friction Torque (N•m)		0.32	0.32	1.3	1.3	2.4
Permissible Braking Work	Per Braking (J)	5.6	5.6	22	22	64
	Per Hour (J)	56	56	220	220	640
Electromagnetic Brake Life (*2)	Number of Times (Times)	20000				
	Work Per Braking (J)	5.6	5.6	22	22	64

Notes:
1. The electromagnetic brake is for holding. It should not be used for deceleration applications.
2. Brake gap is not adjustable. Electromagnetic brake life is defined as the time period until the readjustment is needed.

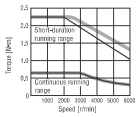
HG-KR053(B) (*1, *2)



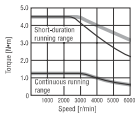
HG-KR13(B) (*1, *2)



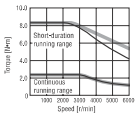
HG-KR23(B) (*1, *2)



HG-KR43(B) (*1, *2)



HG-KR73(B) (*1, *2)



Notes: 1. ═══ : For 3-phase 200 VAC or 1-phase 230 VAC.
2. —— : For 1-phase 200 VAC.
3. Torque drops when the power supply voltage is below the specified value.

A.2 Bonfiglioli TR 080 1 10 LOW 50C1 CD14 S5 OR SB KE reducer

4.1 ORDERING CODE

TR G 080 3 20 STD 95A CD 19 S1 OR SB KL UH1

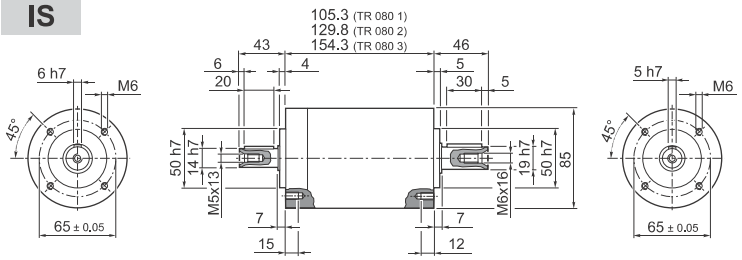
- **LUBRICANT**
  - **–** standard
  - **UH1** food grade synthetic lubricant
- **OUTPUT SHAFT CONFIGURATION**
  - **KL** smooth keyless shaft
  - **KE** keyed shaft (⛔ MB)
- **OUTPUT SHAFT BEARINGS**
  - **SB** standard
- **MOUNTING POSITION**
  - **OR** horizontal
  - **VA** vertical with motor up
  - **VB** vertical with motor down
- **DUTY**
  - **S1** continuous duty
  - **S5** intermittent duty
- **INPUT SHAFT BORE**
  - **6 ... 48** (⛔ IS)
- **MOTOR COUPLING**
  - **CD** clamping device (⛔ IS)
- **INPUT SECTION**
  - **25AH ... 180A1** motor adapter
  - **IS** solid imput shaft (TR 053 ... TR 160) (⛔ G - ⛔ MB)
  - **FM** without motor adapter
- **BACKLASH**

	1 reduction	2 reductions	3 reductions (MB, G only)	3 reductions	4 reductions (MB, G only)
STD	$\varphi_S \leq 5'$	$\varphi_S \leq 5'$	$\varphi_S \leq 5'$	$\varphi_S \leq 7'$	$\varphi_S \leq 7'$
LOW (⛔ TR 053)	$\varphi_R \leq 3'$	$\varphi_R \leq 3'$	$\varphi_R \leq 3'$	$\varphi_R \leq 5'$	$\varphi_R \leq 5'$

- **GEAR RATIO**
- **REDUCTIONS**
  - **1** **2** **3** **4**
- **FRAME SIZE**
  - **053** **060** **080** **105** **130** **160** **190**
- **VERSION**
  - **–** coaxial
  - **G** with input right-angle (TR 053 ... TR 160)
  - **MB** with output right-angle (TR 080 ... TR 160)
- **SERIES**
  - **TR**

TR 080

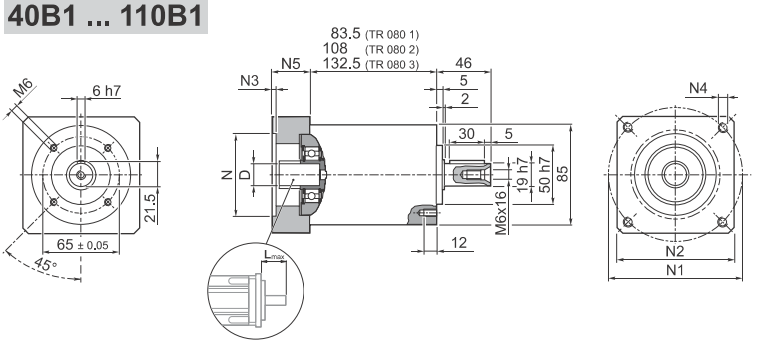
IS



	Kg
TR 080 1	4.0
TR 080 2	4.6
TR 080 3	5.2

TR

40B1 ... 110B1



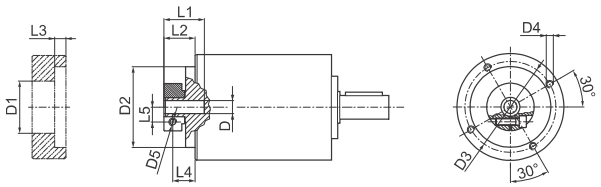
	Kg
TR 080 1	4.0
TR 080 2	4.6
TR 080 3	5.2

	D											N	N1	N2	N3	N4	N5	L_{max}
40B1	8	9	9.52	11	12	12.7	14	–	–	–	–	–	40	63	80	4	M4x12	34	40
45A	8	9	9.52	11	12	12.7	–	–	–	–	–	–	45	63	80	4	M4x12	34	40
50B1	8	9	9.52	11	12	12.7	14	–	–	–	–	–	50	65	80	4	M5x16	34	40
50BH1	8	9	9.52	11	12	12.7	14	–	–	–	–	–	50	65	80	4	5.5	34	40
50C1	8	9	9.52	11	12	12.7	14	–	–	–	–	–	50	70	80	4	M4x10	34	40
50D	8	9	9.52	11	12	12.7	14	–	–	–	–	–	50	95	80	4	M6x20	34	40
55A	8	9	9.52	11	12	12.7	14	15.875	16	17	19	19.05	55.5	125.7	105	4	M6x20	34	40
60A2	8	9	9.52	11	12	12.7	14	–	–	–	–	–	60	75	80	4	M5x16	34	40
60AH2	8	9	9.52	11	12	12.7	14	–	–	–	–	–	60	75	90	4	6.5	34	40
60B1	8	9	9.52	11	12	12.7	14	15.875	16	–	–	–	60	85	80	4	M5x16	34	40
60C1	8	9	9.52	11	12	12.7	14	15.875	16	–	–	–	60	90	80	4	M5x16	34	40
70A1	8	9	9.52	11	12	12.7	14	15.875	16	17	19	19.05	70	85	80	4	M6x20	34	40
70AH1	8	9	9.52	11	12	12.7	14	15.875	16	17	19	19.05	70	85	90	4	6.5	34	40
70B1	8	9	9.52	11	12	12.7	14	15.875	16	17	19	19.05	70	90	80	4	M5x16	34	40
73A1	8	9	9.52	11	12	12.7	14	–	–	–	–	–	73	98.4	85	4	M5x16	34	40
80A1	8	9	9.52	11	12	12.7	14	15.875	16	17	19	19.05	80	100	90	4	M6x16	34	40
95A	8	9	9.52	11	12	12.7	14	15.875	16	17	19	19.05	95	115	100	4	M8x20	34	40
95B	8	9	9.52	11	12	12.7	14	15.875	16	17	19	19.05	95	130	115	4	M8x20	34	40
110A	8	9	9.52	11	12	12.7	14	15.875	16	17	19	19.05	110	130	115	4	M8x20	34	40
110B	8	9	9.52	11	12	12.7	14	15.875	16	17	19	19.05	110	145	120	6.5	M8x20	44	50
110B1	8	9	9.52	11	12	12.7	14	15.875	16	17	19	19.05	110	145	120	6.5	M8x20	54	60

Please contact us for other motor adapters and input shaft bore.

TR 080

FM



D				D1	D2	D3	D4	D5	L1	L2	L3	L4	L5
8	9	9.52		38	68	76.5	M6x12	M6	32.2	26.3	9.5	19.3	10.5
11	12	12.7		43	68	76.5	M6x12	M6	32.2	26.3	9.5	19.3	12.5
14	15.875	16	17	48	68	76.5	M6x12	M6	32.2	26.3	9.5	19.3	14.5
19	19.05			51	68	76.5	M6x12	M6	32.2	26.3	9.5	19.3	16.5

i	M_{n2} [Nm]	M_{a2} [Nm]	M_{p2} [Nm]	n_1 [min^{-1}]	$n_{1\,max}$ [min^{-1}]	φ_S [arcmin]	φ_R [arcmin]	C_t [Nm/arcmin]	$R_{1\,max}$ [N]	$R_{2\,max}$ [N]	$A_{2\,max}$ [N]	η %	J_G [kgcm^2] D 8 ... 12.7	J_G [kgcm^2] D 14 ... 19.05
TR 080 1_3	40	80	180	3500	3500	5'	3'	8.0	400	2500	3000	97	0.50	0.59
TR 080 1_4	50	80	200	4500	4500	5'	3'	8.0	400	2500	3000	97	0.34	0.43
TR 080 1_5	50	80	200	4500	4500	5'	3'	8.0	400	2500	3000	97	0.28	0.37
TR 080 1_6	50	80	200	4500	4500	5'	3'	8.0	400	2500	3000	97	0.21	0.30
TR 080 1_7	50	80	200	6000	6000	5'	3'	8.0	400	2500	3000	97	0.23	0.32
TR 080 1_10	40	80	180	6000	6000	5'	3'	8.0	400	2500	3000	97	0.20	0.29
TR 080 2_9	40	80	180	3500	3500	5'	3'	6.5	400	2500	3000	94	0.49	0.58
TR 080 2_12	70	100	250	3500	3500	5'	3'	6.5	400	2500	3000	94	0.47	0.56
TR 080 2_15	70	100	250	3500	3500	5'	3'	6.5	400	2500	3000	94	0.46	0.55
TR 080 2_16	70	100	250	4500	4500	5'	3'	6.5	400	2500	3000	94	0.32	0.41
TR 080 2_20	70	100	250	4500	4500	5'	3'	6.5	400	2500	3000	94	0.27	0.36
TR 080 2_25	70	100	250	4500	4500	5'	3'	6.5	400	2500	3000	94	0.27	0.36
TR 080 2_28	70	100	250	6000	6000	5'	3'	6.5	400	2500	3000	94	0.22	0.31
TR 080 2_30	40	80	180	6000	6000	5'	3'	6.5	400	2500	3000	94	0.20	0.29
TR 080 2_35	70	100	250	6000	6000	5'	3'	6.5	400	2500	3000	94	0.22	0.31
TR 080 2_36	50	80	200	4500	4500	5'	3'	6.5	400	2500	3000	94	0.20	0.29
TR 080 2_40	70	100	250	6000	6000	5'	3'	6.5	400	2500	3000	94	0.20	0.29
TR 080 2_50	70	100	250	6000	6000	5'	3'	6.5	400	2500	3000	94	0.19	0.28
TR 080 2_70	70	100	250	6000	6000	5'	3'	6.5	400	2500	3000	94	0.19	0.28
TR 080 2_100	40	80	180	6000	6000	5'	3'	6.5	400	2500	3000	94	0.19	0.28
TR 080 3_48	70	100	250	4500	4500	7'	5'	5.5	400	2500	3000	91	0.33	0.42
TR 080 3_64	70	100	250	4500	4500	7'	5'	5.5	400	2500	3000	91	0.32	0.41
TR 080 3_75	70	100	250	4500	4500	7'	5'	5.5	400	2500	3000	91	0.27	0.36
TR 080 3_80	70	100	250	4500	4500	7'	5'	5.5	400	2500	3000	91	0.32	0.41
TR 080 3_84	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.23	0.32
TR 080 3_90	40	80	180	6000	6000	7'	5'	5.5	400	2500	3000	91	0.20	0.29
TR 080 3_120	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.20	0.29
TR 080 3_125	70	100	250	4500	4500	7'	5'	5.5	400	2500	3000	91	0.27	0.36
TR 080 3_140	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.22	0.31
TR 080 3_150	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.20	0.29
TR 080 3_160	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.20	0.29
TR 080 3_175	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.22	0.31
TR 080 3_200	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.20	0.29
TR 080 3_210	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.20	0.29
TR 080 3_216	70	100	250	4500	4500	7'	5'	5.5	400	2500	3000	91	0.20	0.29
TR 080 3_250	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.19	0.28
TR 080 3_280	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.19	0.28
TR 080 3_350	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.19	0.28
TR 080 3_400	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.19	0.28
TR 080 3_500	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.19	0.28
TR 080 3_700	70	100	250	6000	6000	7'	5'	5.5	400	2500	3000	91	0.19	0.28
TR 080 3_1000	40	80	180	6000	6000	7'	5'	5.5	400	2500	3000	91	0.19	0.28

TR

A.3 Rollon ELM 80-SP

12 ELM 80 SP - ELM 80CI

ELM 80 SP - Con guide standard a ricircolo di sfere
ELM 80 CI - Con guide a rotelle ad arco gotico

ELM 80 SP - With standard ball bearings guide
ELM 80 CI - With lancet arch bearing guides

Dati tecnici / ***Technical data***

	ELM 80 SP	ELM 80 CI
Lunghezza corsa utile min. [mm] - *Min. useful stroke length [mm]*	100	100
Lunghezza corsa utile max. [mm] - *Max. useful stroke length [mm]*	6000*1	6000*1
Ripetibilità max. di posizionamento [mm] *2 - *Max. positioning repeatability [mm] *2*	0,05	0,05
Velocità max. di traslazione [m/s] - *Max. speed [m/s]*	5,0	1,5
Accelerazione max. [m/s²] - *Max. acceleration [m/s²]*	50	1,5
Tipo di cinghia - *Type of belt*	32 AT 10	32 AT 10
Tipo di puleggia - *Type of pulley*	Ø 60 - Z 19 - Gioco 0	Ø 60 - Z 19 - Gioco 0
Spostamento carro per giro puleggia [mm] - *Carriage displacement per pulley turn [mm]*	190	190
Peso del carro [kg] - *Carriage weight [kg]*	2,7	2,5
Peso corsa zero [kg] - *Zero travel weight [kg]*	10,5	9,5
Peso per ogni 100 mm di corsa utile [kg] - *Weight for 100 mm useful stroke [kg]*	1,0	0,8

*1) È possibile realizzare corse fino a 11000 mm tramite speciali giunzioni Rollon
1) It is possible to obtain strokes up to 11000 mm by means of special Rollon joints

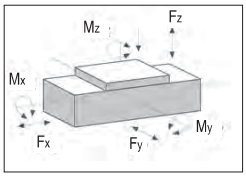
*2) La ripetibilità di posizionamento dipende dal tipo di trasmissione applicato
2) The positioning repeatability depends upon the type of transmission used

ELM 80 - Carichi teorici massimi e consigliati / *ELM 80 - Theoric and maximum permissible loads*

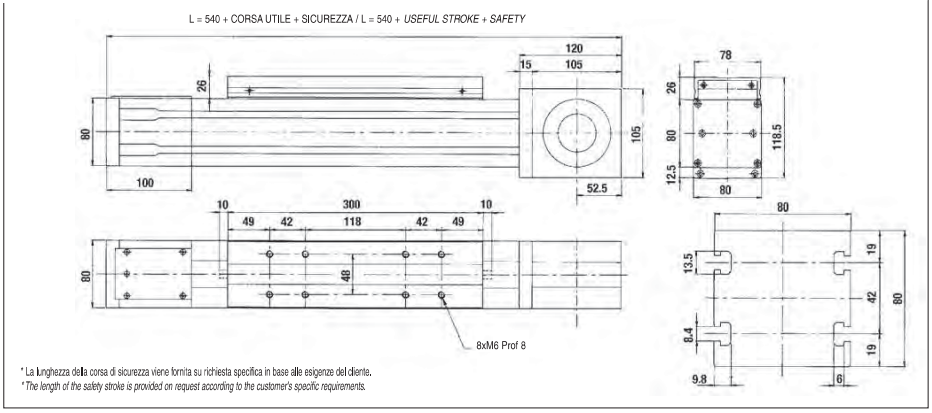
ELM 80 SP	Teorico - *Theoric*		Consigliato - *Permissible**	
	stat	din. / dyn.	stat	din. / dyn.
Fx [N]	2230	1670	1780	1340
Fy [N]	43400	34800	8680	4180
Fz [N]	43400	34800	8680	4180
Mx [Nm]	620	480	120	58
My [Nm]	3170	2540	630	300
Mz [Nm]	3170	2540	630	300

ELM 80 CI	Teorico - *Theoric*		Consigliato - *Permissible**	
	stat	din. / dyn.	stat	din. / dyn.
Fx [N]	2230	1670	1780	1340
Fy [N]	8500	17000	1700	1700
Fz [N]	4740	8700	950	950
Mx [Nm]	140	250	28	28
My [Nm]	390	710	78	78
Mz [Nm]	700	1390	140	140

*) *Con i valori riportati si ottengono una ragionevole durata ed una sufficiente sicurezza statica.*
*) *Reasonable operating life and system rigidity can be obtained from the values given.*

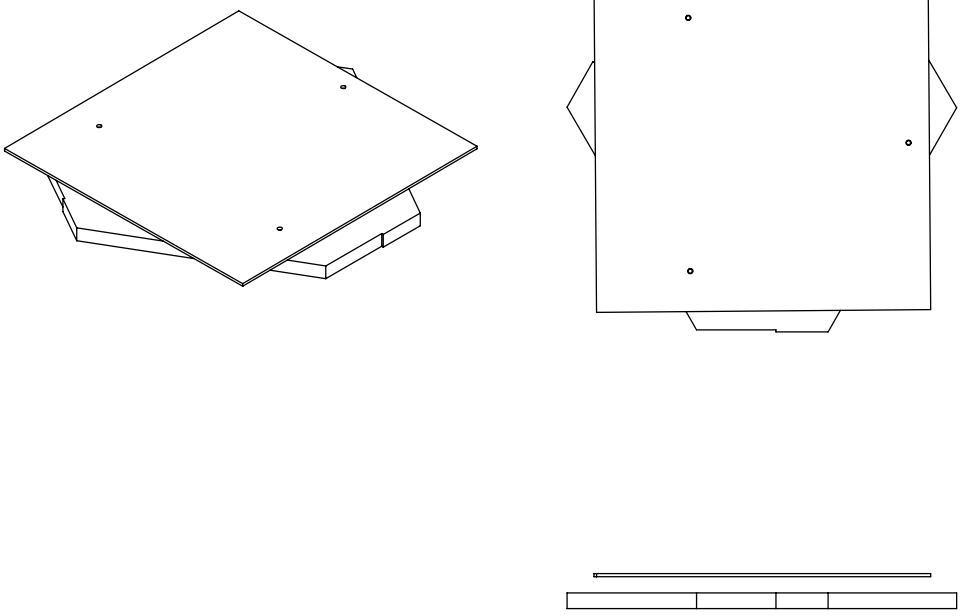


Dimensioni ELM 80 SP - ELM 80 CI / ***ELM 80 SP - ELM 80 CI dimensions***



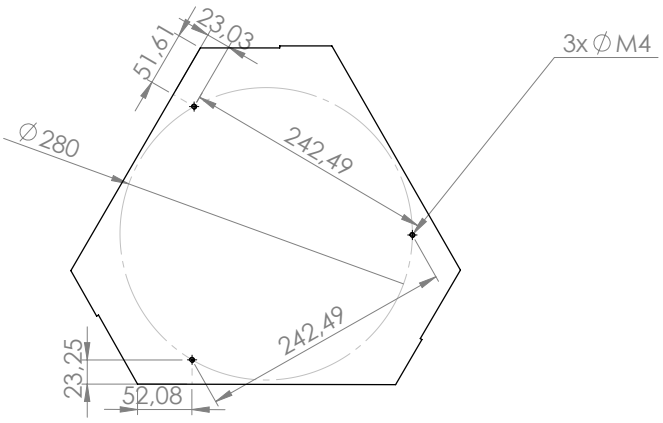
* La lunghezza della corsa di sicurezza viene fornita su richiesta specifica in base alle esigenze del cliente.
* *The length of the safety stroke is provided on request according to the customer's specific requirements.*

A.4 adjustable plate



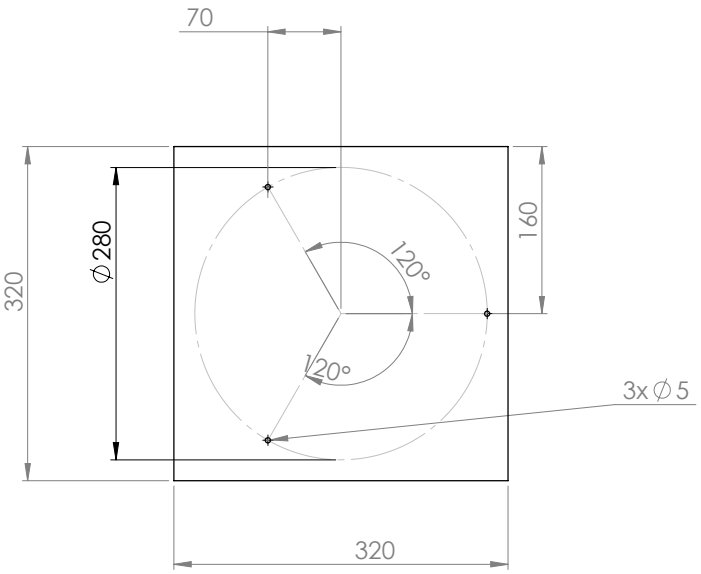
SE NON SPECIFICATO: QUOTE IN MILLIMETRI FINITURA SUPERFICIE: TOLLERANZE: LINEARE: ANGOLARE:		FINITURA: $PRPSHEET:{Fine}				SBAVATURA E INTERRUZIONE DEI BORDI NETTI	NON SCALARE IL DISEGNO	REVISIONE
	NOME	FIRMA	DATA				TITOLO: $PRPSHEET:{Descrizione}	
DISEGNATO								
VERIFICATO								
APPROVATO								
FATTO								
QUALITA'				MATERIALE: $PRPSHEET:{Materiale}			N. DISEGNO Disegno31	A4
				PESO: $PRPSHEET:{Peso}			SCALA:1:5	FOGLIO 1 DI 1

A.4 adjustable plate



SE NON SPECIFICATO: QUOTE IN MILLIMETRI FINITURA SUPERFICIE: TOLLERANZE: LINEARE: ANGOLARE:	FINITURA: $PRPSHEET:{Fine}		SBAVATURA E INTERRUZIONE DEI BORDI NETTI	NON SCALARE IL DISEGNO	REVISIONE
	NOME	FIRMA	DATA	TITOLO: $PRPSHEET:{Descrizione}	
DISEGNATO					
VERIFICATO					
APPROVATO					
FATTO	SolidWorks Student Edition. Solo per uso accademico.				
QUALITA'			MATERIALE: $PRPSHEET:{Materiale}	N. DISEGNO Disegno31	A4
			PESO: $PRPSHEET:{Peso}	SCALA:1:1	FOGLIO 1 DI 1

A.4 adjustable plate



SE NON SPECIFICATO: QUOTE IN MILLIMETRI FINITURA SUPERFICIE: TOLLERANZE: LINEARE: ANGOLARE:	FINITURA: $PRPSHEET:{Fine}			SBAVATURA E INTERRUZIONE DEI BORDI NETTI	NON SCALARE IL DISEGNO	REVISIONE
	NOME	FIRMA	DATA		TITOLO: $PRPSHEET:{Descrizione}	
DISEGNATO						
VERIFICATO						
APPROVATO						
FATTO	SolidWorks Student Edition. Solo per uso accademico.					
QUALITA'				MATERIALE: $PRPSHEET:{Materiale}	N. DISEGNO Disegno31	A4
				PESO: $PRPSHEET:{Peso}	SCALA:1:5	FOGLIO 1 DI 1

A.5 electrical layout for the cabinet

COMPONENTS LIST:

1 PLC rack
5 MR-J4
1 HMI
5 SSCNET III cables
1 ethernet cable HMI to PLC
1 magnetothermic switch 25 A [4 067 76] (S1)
2 power supply 24V (T1-T2)
1 emergency stop (EMG1)
1 security relay 3NO [Schneider Electric XPSAC5121 XPS] (K1)
1 NC red push buttons (OFF)
1 NO green push buttons (ON)
5 contactor realays NO (MC1-MC2-MC3-MC4-MC5)
5 relays NO 6A (RA1-RA2-RA3-RA4-RA5)
5 relays NO 6A (RB1-RB2-RB3-RB4-RB5)
5 pin terminal (PT1-PT2-PT3-PT4-PT5)
5 magnetothermic switch 6A (SW1-SW2-SW3-SW4-SW5)
3 relays NO 6A (RT1-RT2-RT3)
2 relays NO 6A (RP1-R_AIR)
5 varistor [Nippon] (R1-R2-R3-R4-R5)
terminals

CABINET IN

1) 220V plug
2) compressed-air

CABINET OUT

1) compressed-air
2) to motor cables (power, encoder, break)

CABINET TERMINALS (post-realization connection)

1) proximity sensor
2) pump power
3) thermocouple cables
4) extrusion resistor [lower part of the cabinet]

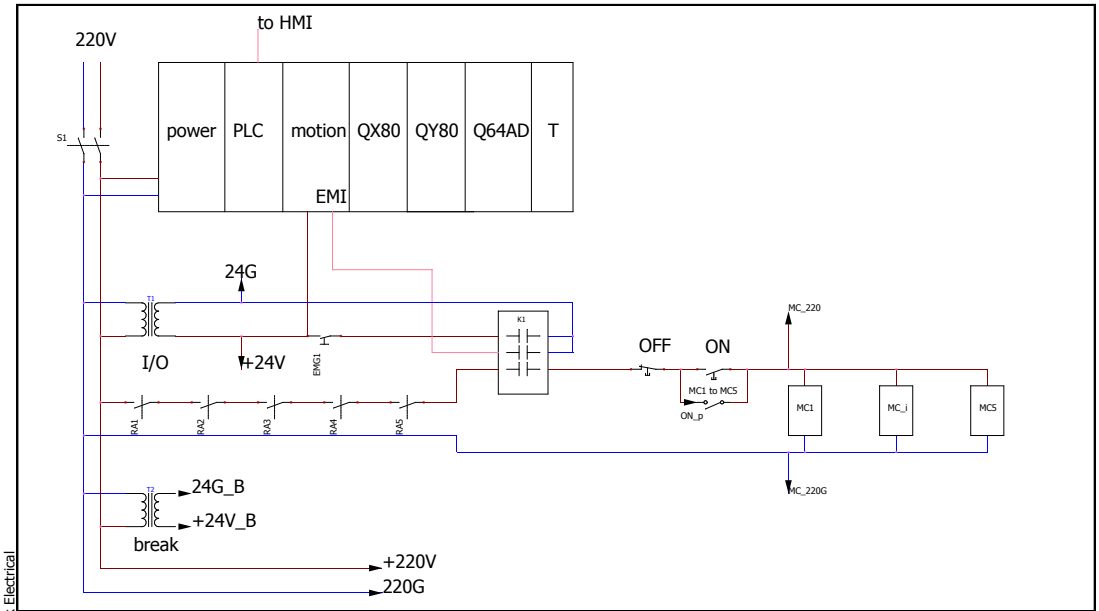
TO CONTROL PANEL

1) HMI
2) emergency stop (EMG1)
3) 7x USB female connector (1 for PLC, 1 for HMI, 5 MR-J4)

DesignSpark Electrical

REVISION	REV.	DATE	NAME	CHANGES	SCHEME
0	0	15/02/2017	utente		01

CONTRACT: LOCATION: L1 Main electrical closet User data 1 User data 2

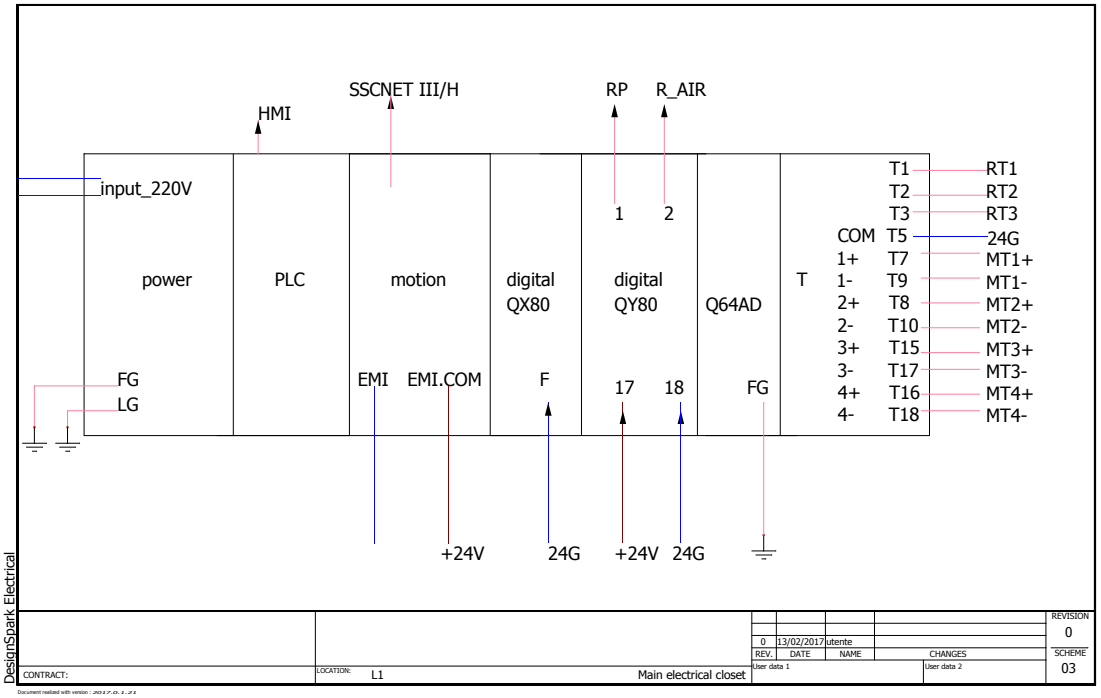


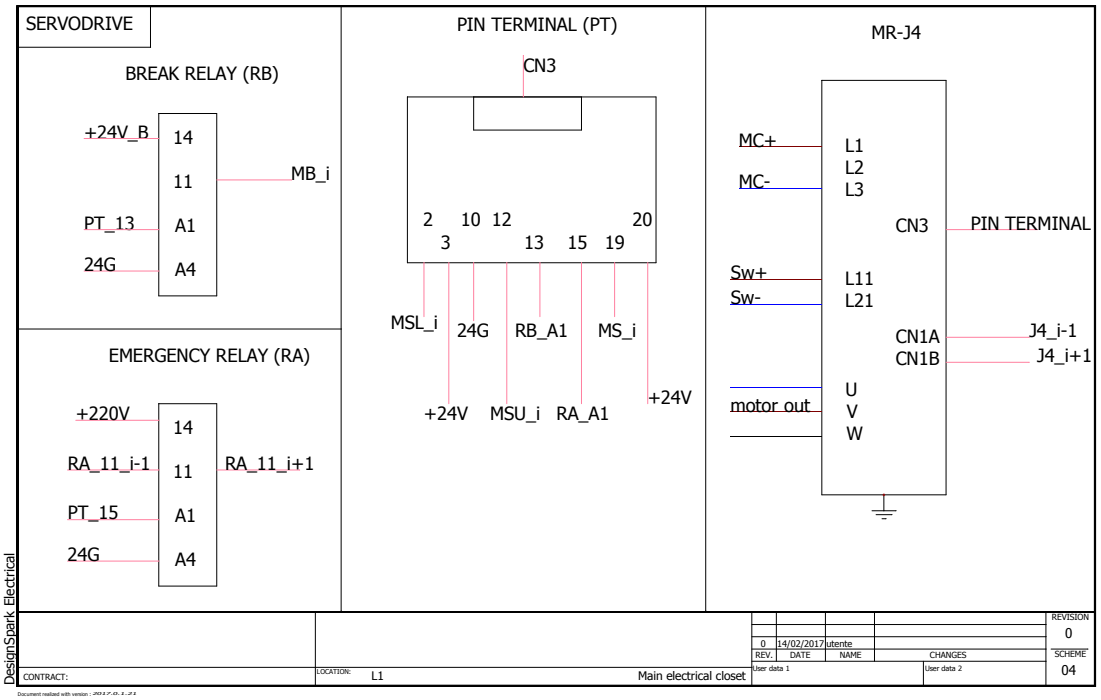
DesignSpark Electrical

REVISION	REV.	DATE	NAME	CHANGES	SCHEME
0	0	13/02/2017	utente		02

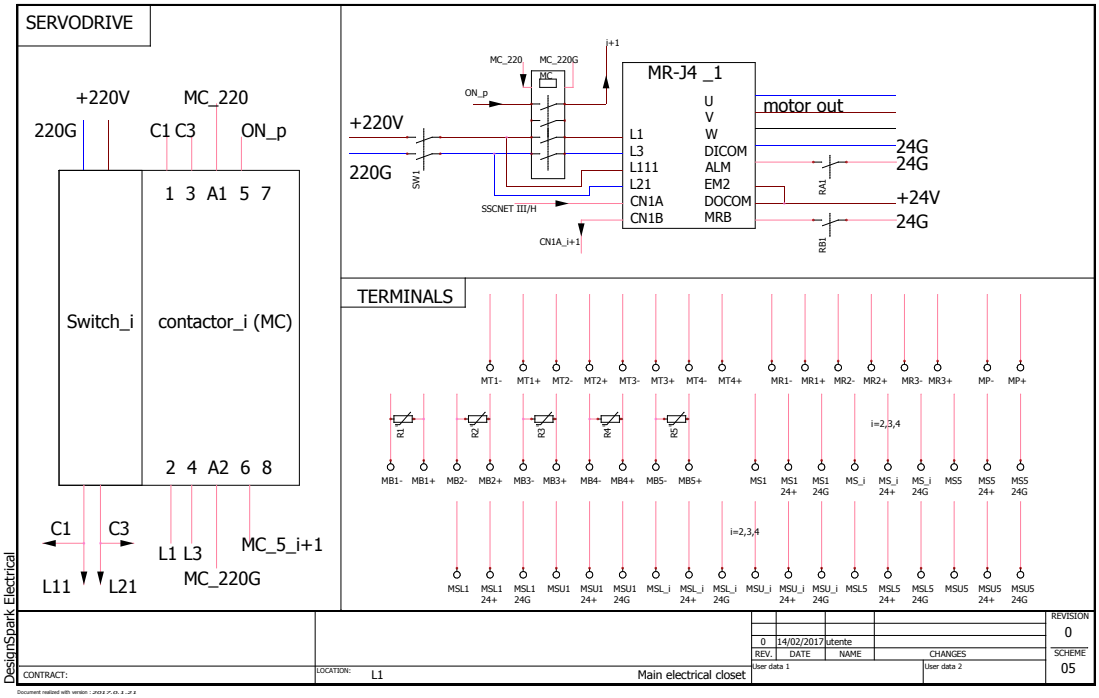
CONTRACT: LOCATION: L1 Main electrical closet User data 1 User data 2

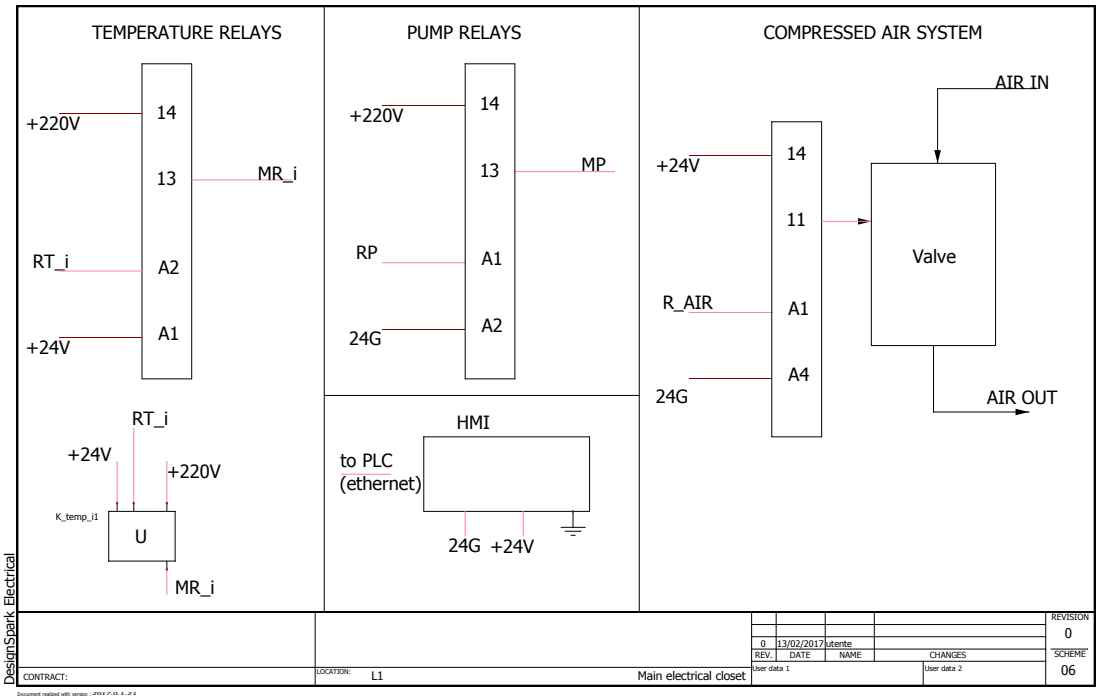
A.5 electrical layout for the cabinet



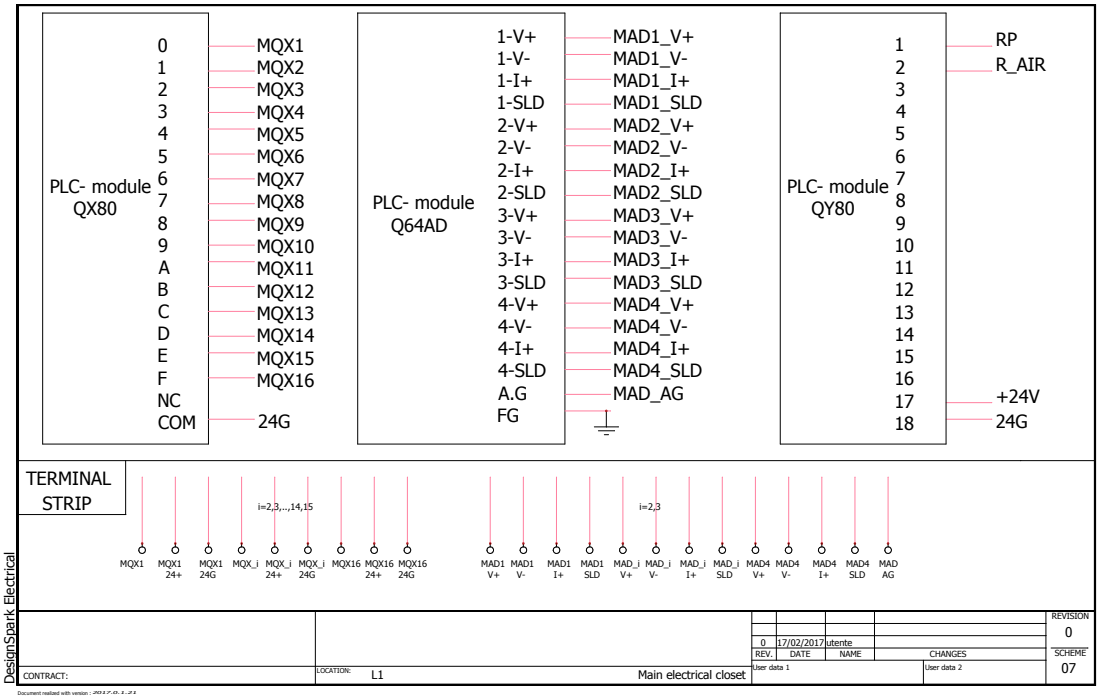


A.5 electrical layout for the cabinet





A.5 electrical layout for the cabinet



24-pins connector hook up table

TCM terminal	24Pin terminal	TCM terminal	24Pin terminal
T1	1	T8	7(+)
T2	2	T10	19(-)
T3	3	T15	8(+)
T5(GND)	13-14-15 GND	T17	20(-)
T7	6(+)	T16	10(+)
T9	18(-)	T18	19(-)

A.6 electrical mounting specification

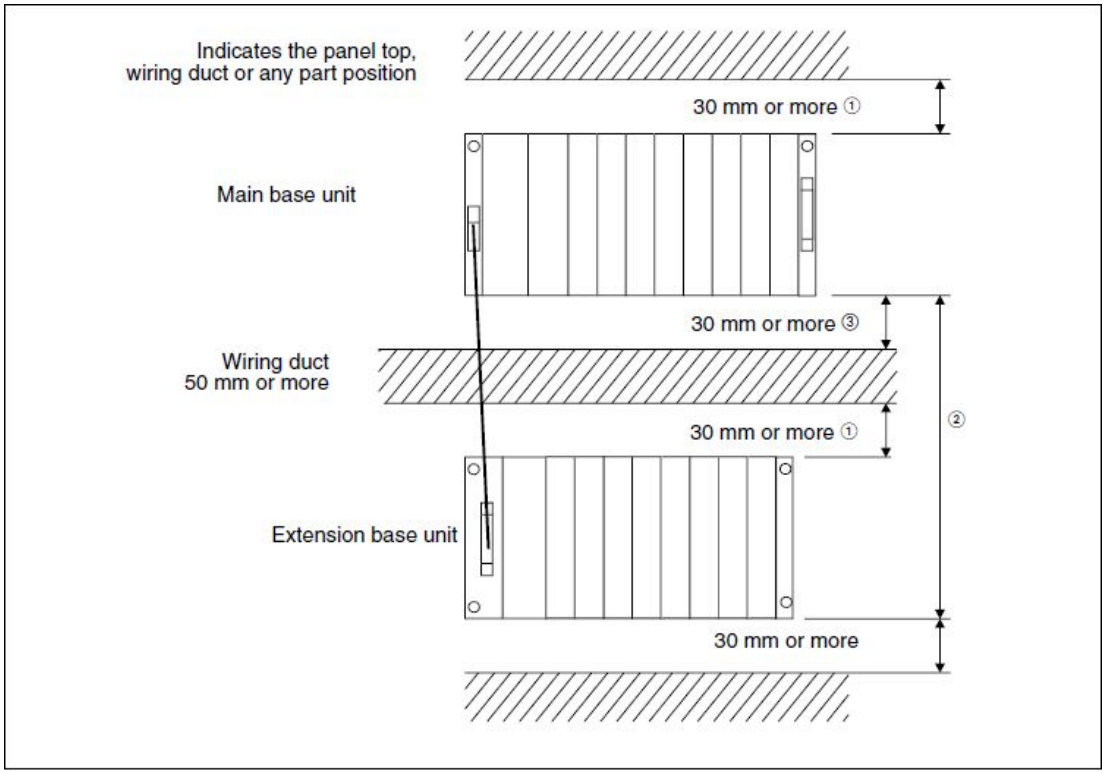


Figure A.1: PLC specification

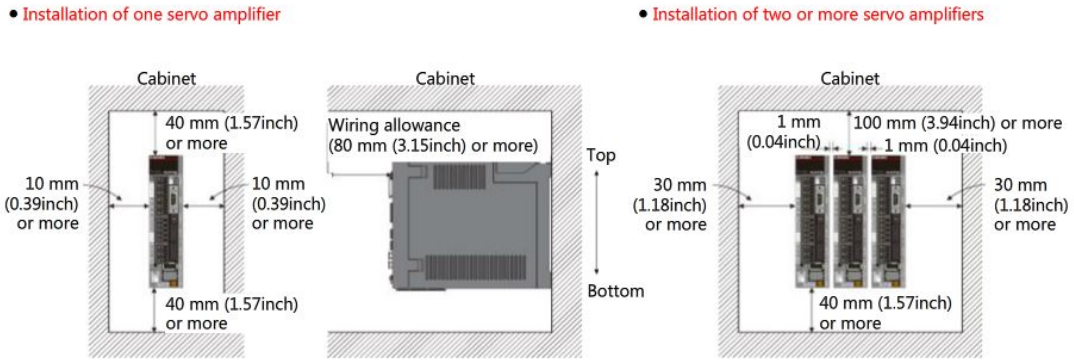


Figure A.2: servo amplifier specification

A.7 bolt selection for the structure

In order to assemble the structure as can be seen in Figure 2.3, the screws and bolts needed to be chosen to accommodate the specifics of the design. The design was developed to increase rigidity after some experimental analysis that showed a first frequency at around 5 Hz. Simulation carried out in previous thesis work showed that this configuration with crossed horizontal strips could significantly increase the first eigenvalue of the system. Here is presented a table with the screws and bolts required[1]:

PIECE	n	THEO. Dn	THEO. Lnt [mm]	Lt [mm]	L_tot	N_tot
bolts for LOWER STRIP	12	M6	12	12	24	36
bolts for HIGHER STRIP	3	M6	66	12	78	9
	4	M6	12	12	24	12
Scres for Upper C closing	4	M6	10	9	19	8
screws for linear guide extruder holder	8	M6	6	9	15	24
screws for linear guide crossed strip	16	M6	6	9	15	48
screws for support	4	M6	15	9	24	12
	2	M6	90	15	105	6
	2	M6	96	15	111	6

				Hexagon head screws				Hexagon head screws			
	bolt	L_tot	N_tot		ISO4017	DIN 975	L_tot+washer		ISO4017	DIN 975	
M6	yes	24	48		25		28		30		
M6	yes	78	9		80/90(comp_or_part)		82		80/90(comp_or_part)		
M6	no	20	12		20		22		25		1,6/2 mm washer
M6	no	19	8		20(25)	washer need	20,6		20(25)		
M6	no	15	72		16(20)	washer need	16,6		16(20)		
M6	no	24	12		25(?)		25,6		25(?)		1,6/2 mm washer
M6	-	105	6		x	threaded rod	-		x	threaded rod	-->
M6	-	111	6		x	threaded rod	-		x	threaded rod	-->
			tot_num								
			173								

[1] only the additional fasteners are here included

Appendix B

Matlab scripts

B.1 robot behavior analysis

B.1.1 MAIN_X_Z

Listing B.1: MAIN_X_Z

```
clear;clc;
format short g;
l=0.595;     Rb=0.198;     s=0.45651;
vMaxQ=[0.05;0.05;0.05];          %[m/s]
aMaxQ=[0.2;0.2;0.2];             %[m/s^2]
vector=5:5:20;
for jj=vector
p0=[8e-3;0;0.6];  %initial position
p1=[jj*1e-3;0;0.6];
zEstrusore=0.8;
alpha_0=linear_inverse_kinematics(p0,l,s,Rb);
alpha_1=linear_inverse_kinematics(p1,l,s,Rb);
(alpha_1-alpha_0)*10^6;
format;
n=50;
dx=(alpha_1-alpha_0)/n;
DX=(p1(1)-p0(1))/n;
p_flag=p0(1):DX:p1(1);
p_temp=[p_flag;p1(2)*ones(1,length(p_flag));p1(3)*ones(1,length(p_flag))];

vect=[alpha_0,alpha_1];
ma=max(vect,[],2);
mi=min(vect,[],2);

for j=1:length(p_flag)
    S(:,j)=linear_inverse_kinematics(p_temp(:,j),l,s,Rb);
end
j=1;
for i=1:n+1;
        alpha_p=alpha_0+(i-1)*dx;
```

```
31          alpha_1(j)=alpha_p(1);
32          alpha_2(j)=alpha_p(2);
33          alpha_3(j)=alpha_p(3);
34          q=[alpha_1(j);alpha_2(j);alpha_3(j)];
35          zoom=0;
36          [tc_x(j),tc_y(j),tc_z(j)]=plotLD(q,l,s,Rb,zEstrusore,zoom);
37          subplot(1,2,2)
38          plot(tc_x,tc_y);
39          axis equal
40          j=j+1;
41  end
42  %%
43  tc_X{jj}=tc_x;tc_Y{jj}=tc_y;tc_Z{jj}=tc_z;
44  end
45  %%
46  figure()
47  for jj=vector
48  plot(tc_X{jj},(tc_Z{jj}-p0(3))*1e3);
49  hold on
50  end
51  figure()
52  for jj=vector
53  plot(tc_Y{jj},(tc_Z{jj}-p0(3))*1e3);
54  hold on
55  end
```

B.1.2 MAIN_X_Z_fl f(l)

Listing B.2: MAIN_X_Z_fl

```
1   clear;clc;
2   format short g;
3   l=0.595;      Rb=0.198;     s=0.45651;
4   vMaxQ=[0.05;0.05;0.05];          %[m/s]
5   aMaxQ=[0.2;0.2;0.2];             %[m/s^2]
6   L=[0.595;0.595;0.595];
7   low=0.98;
8   up=1.02;
9   dt=5e-4;
10  L1=0.590:dt:0.600;
11  vector=1:length(L1);
12
13  zEstrusore=0.8;
14  p0=[0;0;0.6];  %initial position
15  p1=[100*1e-3;0;0.6];
16  alpha_0=linear_inverse_kinematics1(p0,L,s,Rb);
17  alpha_1=linear_inverse_kinematics1(p1,L,s,Rb);
18  q=alpha_1;
19  dq=alpha_1-alpha_0;
20  format;
21
22  for jj=vector
```

```
23 p0(:,jj)=linear_direct_kinematics1(0.25*ones(3,1),L1(jj)*ones(3,1),s,Rb);
24 p(:,jj)=linear_direct_kinematics1(0.25*ones(3,1)+dq,L1(jj)*ones(3,1),s,Rb)
       ;
25 tc_X{jj}=p(1,jj);tc_Y{jj}=p(2,jj);tc_Z{jj}=p(3,jj);
26 end
27
28 figure()
29 for jj=vector
30 plot([0,tc_X{jj}],[0,p(3,jj)-p0(3,jj)]*1e3);
31 hold on
32 a{jj}=num2str(L1(jj));
33 end
34 xlabel('[m]');ylabel('[mm]')
35 legend(a{1:vector(end)})
```

B.1.3 MAIN_X_Z_3 f(Rb and/or s)

Listing B.3: MAIN_X_Z_3

```
1  clear;clc;
2  format short g;
3  A=ones(3);
4  syms a b c q1 q2 q3
5  B=[2*a,0,-2*q1;-b,b*sqrt(3),-2*q2;-c,-c*sqrt(3),-2*q3];
6  syms q t r
7  C=[q;t;r];
8  syms x y w
9  xt=[x,y,w];
10 xn=[x;y;w];
11 syms S
12 S=solve(xt*A*xn+B*xn==C)
13 X(a,b,c,q,t,r,q1,q2,q3)=S.x;
14 Y(a,b,c,q,t,r,q1,q2,q3)=S.y;
15 Z(a,b,c,q,t,r,q1,q2,q3)=S.w;
16
17 l=0.59;    Rb=0.15;    s=0.45651;
18 vector=5:5:20;
19 %%
20 n=25;
21 c=1/3; x=0:1/n:1;    j=1;
22 for i=x
23 [d(j),s1,a]= const_a_motlaw(i,c);
24 j=j+1;
25 end
26  %%
27 for jj=vector
28 dx=[];DX=[];
29 p_falg=[]; p_temp=[];vect=[];
30 l=0.59;    Rb=0.15;    s=0.45651;
31 p0=[0;0;0];  %initial position
32 p1=[jj*1e-3;0;0];
33 alpha_0=linear_inverse_kinematics(p0,l,s,Rb);
```

```
alpha_1=linear_inverse_kinematics(p1,l,s,Rb);

alpha_temp(1,:)=alpha_0(1)+(alpha_1(1)-alpha_0(1)).*d;
alpha_temp(2,:)=alpha_0(2)+(alpha_1(2)-alpha_0(2)).*d;
alpha_temp(3,:)=alpha_0(3)+(alpha_1(3)-alpha_0(3)).*d;
j=1;
l=0.59;  Rb=0.15*[0.999,1,1.001];    s=0.45651*ones(3,1);
for i=1:n+1;
q=alpha_temp(:,i);
p=[];
p=linear_direct_kinematics2(q,l*ones(3,1),s,Rb,X,Y,Z);
tc_x(j)=p(1); tc_y(j)=p(2); tc_z(j)=p(3);
j=j+1;
end
tc_X{jj}=tc_x;tc_Y{jj}=tc_y;tc_Z{jj}=tc_z;
end
%%
figure()
for jj=vector
plot3(tc_X{jj}-tc_X{jj}(1),(tc_Y{jj}-tc_Y{jj}(1))*1e3,(tc_Z{jj}-tc_Z{jj
    }(1))*1e3);
hold on
end
xlabel('l_x [m]');ylabel('y [mm]');zlabel('z [mm]')
legend(num2str(vector(1)),num2str(vector(2)),num2str(vector(3)),num2str(
    vector(4)))
```

B.2 tuning_tot_main

Listing B.4: tuning_tot_main

```
 clear;clc
[filename, pathname] = uigetfile('.csv');
DATA1=load([pathname,filename]);
data1=DATA1(6:end)';
[filename, pathname] = uigetfile('.csv');
DATA2=load([pathname,filename]);
data2=DATA2(6:end)';
[filename, pathname] = uigetfile('.csv');
DATA3=load([pathname,filename]);
data3=DATA3(6:end)';
vect=[data1;data2;data3];

time=vect(:,1:2:end-1);
posi=vect(:,2:2:end);

LAYER=1;
DT_motlaw_lay{LAYER}=time(1,:)/1000/1000;
E_motlaw_lay{LAYER}=ones(1,length(DT_motlaw_lay{LAYER}));
alpha_motlaw1_final{LAYER}=cumsum(posi(1,:)*10^-7);
alpha_motlaw2_final{LAYER}=cumsum(posi(2,:)*10^-7);
alpha_motlaw3_final{LAYER}=cumsum(posi(3,:)*10^-7);
```

```
%%
l=[0.595,0.595,0.595]; Rb=0.198;        s=0.45651;   zEstrusore=0.8;
y_max_ref=39.5*1e-3;
x_max_ref=39*1e-3;
y_max=38*1e-3;
x_max=39*1e-3;
z_max=0.5*1e-3;

Z=ones(1,length(DT_motlaw_lay))*1e-3;

for i=1:length(alpha_motlaw1_final{LAYER})
    q=[alpha_motlaw1_final{LAYER}(i);alpha_motlaw2_final{LAYER}(i);
        alpha_motlaw3_final{LAYER}(i)];
    p(:,i)=linear_direct_kinematics1(q,l,s,Rb);
end
x0=p(1,:);
y0=p(2,:);
z0=p(3,:);
P0=[x0;y0;z0];
figure()
plot(x0,y0)
axis equal

opt=menu('calibration?','yes','no');
if opt==1
Rb=0.198;    s=0.45651;   zEstrusore=0.8;
L=[0.595,0.595,0.595];
Z=ones(1,length(DT_motlaw_lay))*1e-3;
dt=3.0*1e-3;
low=0.75;
up=0.9;
tol=5e-4;
tolz=5e-2;
j=1;
zz=[];xx=[];
yy=[];T=[];
h = waitbar(0,'Please wait...');
step=1;
steps=length(low*L(1):dt:up*L(1))*length(low*L(2):dt:up*L(2))*length(low*L
    (3):dt:up*L(3))*length(low*Rb:dt*10:up*Rb)*length(low*s:dt*10:up*s);

for a=low*L(1):dt:up*L(1)
    for b=low*L(2):dt:up*L(2)
        for c=low*L(3):dt:up*L(3)
            for d=low*Rb:dt*10:up*Rb
                for e=low*s:dt*10:up*s

            l=[a,b,c];
            for i=1:length(alpha_motlaw1_final{LAYER})
            q=[alpha_motlaw1_final{LAYER}(i);alpha_motlaw2_final{LAYER}(i)
                ;alpha_motlaw3_final{LAYER}(i)];
```

```
70              p(:,i)=linear_direct_kinematics1(q,l,e,d);
71              end
72              x=p(1,:)-p(1,1);
73              y=p(2,:)-p(2,1);
74              z=p(3,:);
75              waitbar(step / steps,h,sprintf('%12.9f',length(T)));
76              step=step+1;
77                          if  abs(max(x)-x_max)<tol && abs(max(y)-y_max)<tol
                                && abs((max(z)-min(z))-z_max)<tolz
78              T{j}=[max(x);max(y);max(p(3,:))-min(p(3,:));a;b;c;d;e];
79              j=j+1;
80                          end
81                  end
82              end
83
84          end
85      end
86  end
87
88  close(h)
89  length(T)
90  %%
91  if length(T)>=1
92       figure()
93      for i=1:length(T)
94          plot(i,T{i}(1),'b*')
95          hold on
96           plot(i,T{i}(2),'r*')
97           plot(i,T{i}(3),'k*')
98           xx(i)=T{i}(1);
99           yy(i)=T{i}(2);
100          zz(i)=T{i}(3);
101
102     end
103     temp=abs(xx-x_max).*abs(yy-y_max);
104          plot([1,length(T)],[x_max,x_max],'b')
105          plot([1,length(T)],[y_max,y_max],'r')
106          plot([1,length(T)],[z_max,z_max],'k')
107
108 [min_xval,pos_minx]=min(abs(xx-x_max));
109 [min_yval,pos_miny]=min(abs(yy-y_max));
110 [min_zval,pos_minz]=min(abs(zz-z_max));
111 plot(pos_minx,T{pos_minx}(1),'bO')
112 plot(pos_miny,T{pos_miny}(2),'rO')
113 plot(pos_minz,T{pos_minz}(3),'kO')
114 save('data_tuning1.mat','T')
115 hold off
116
117 %%
118 figure()
119 flag=pos_minx;
```

```
l=[T{flag}(4),T{flag}(5),T{flag}(6)]
Rb=T{flag}(7)
s=T{flag}(8)
for i=1:length(alpha_motlaw1_final{LAYER})
            q=[alpha_motlaw1_final{LAYER}(i);alpha_motlaw2_final{LAYER}(i)
                ;alpha_motlaw3_final{LAYER}(i)];
            p(:,i)=linear_direct_kinematics1(q,l,s,Rb);
end
x=p(1,:)-p(1,1);
y=p(2,:)-p(2,1);
z=p(3,:);
P=[x;y;z];
plot3(x,y,z)
hold on
plot3(x0,y0,z0,'k')

figure()
plot(x,y)
hold on
plot(x0,y0,'k')
axis equal
end
else
end
```

B.3 G-code reader

B.3.1 MAIN_Gcode

Listing B.5: MAIN_Gcode

```
clear;clc;
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% robot data
l=0.59;     Rb=0.15;    s=0.45651;   zEstrusore=0;
a_max=1200000;%[mm/min^2]
d_max=2*a_max;
%use to refer the velocity at the extruder from
%slicer to Efesto
Aratio_slicer=(1.75/0.9)^2; %filament/nozzle diameter ration
%cura engine uses 1.75mm filament, if stated otherwise change it
Aratio_extr=(0.9/14)^2; %nozzle/punch diameter ration

% file data
[file, pathname] = uigetfile('.GCode');
flag_plot=1;
nr_layer=15; %if you choose a partition
LAYER=1:nr_layer;
gcode_flag=1;

```

```
% intermediate points generatio data
dL=10; %[mm] do not change for now

%slicer
nr_slicer_points=2000;

% saving data
title1='a';
flag_layer_save=1:2; %save option layers partition

% plot data
layer_flag2=2;
z_estrusore=0.8;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% G_code reader
[X,Y,Z,E,F,ext_on,LAYER,delta_time,F_ext, DT_uniform]=G_code_reader(file,
    nr_layer,flag_plot,LAYER);

%intermediate points generatio
[XX_final,YY_final,EE_final,FF_final,FF_ext_final]=G_code_inter_points(
    LAYER,X,Y,Z,E,ext_on,dL,F,F_ext);

% delta time generation 1
[alpha1,alpha2,alpha3,l_seg,DT_motlaw_lay,dt_gcode]=G_code_time1(LAYER,
    XX_final,YY_final,EE_final,Z,zEstrusore,l,s,Rb,FF_final);

%g_code analysis
% layer_reducer

% motion law assignement
alpha_motlaw1_final=alpha1;
alpha_motlaw2_final=alpha2;
alpha_motlaw3_final=alpha3;
E_motlaw_lay=EE_final;

%mot_law_plot
% G_code_mot_law_plot(layer_flag2,alpha_motlaw1_final,alpha_motlaw2_final,
    alpha_motlaw3_final,valpha_motlaw1_final,valpha_motlaw2_final,
    valpha_motlaw3_final,aalpha_motlaw1_final,aalpha_motlaw2_final,
    aalpha_motlaw3_final,DT_motlaw_lay);

%animation
% G_code_anaimation1(DT_motlaw_lay,E_motlaw_lay,alpha_motlaw1_final,
    alpha_motlaw2_final,alpha_motlaw3_final,layer_flag2,l,s,Rb,zEstrusore,
    Z);
%%
```

```
59 % saving
60  [A1,A2,A3,EA1,DT1]=G_code_slicer(LAYER,DT_motlaw_lay,nr_slicer_points,
        E_motlaw_lay,alpha_motlaw1_final,alpha_motlaw2_final,
        alpha_motlaw3_final,l_seg,delta_time,l,s,Rb,zEstrusore,Z);
61  opt_save=menu('save?','x-y format','vel-disp','G-code','no');
62  if opt_save==1
63     xx=[];yy=[];zz=[];fext=[];frob=[];
64          for i=1:length(X)
65              xx=[xx,X{i}];yy=[yy,Y{i}];
66              zz=[zz,-(Z{i}(1)-Z{1}(1))*ones(1,length(X{i}))];
67              fext=[fext,0,FF_ext_final{i}(1:end-1)*Aratio_slicer*
                    Aratio_extr];
68              frob=[frob,0,F{i}(1:end-1)];
69
70          end
71           temlp=find(EE==0);   fext(temlp)=0;
72           for i=length(fext):-1:1
73                if fext(i)==0 && EE(i)~=0
74                    fext(i)=fext(i+1);
75                end
76           end
77          P_flag1=[xx;yy;zz];  x_flag=cumsum([0,frob]);
78     save2melsoft1(x_flag,P_flag1,strcat(title1,'_x_y_format'),fext)
79  elseif opt_save==2
80       AA1=[];AA2=[];AA3=[];EE=[];dt1=[];P_flag1=[];fext=[];frob=[];
81       LAYER=1;
82              for i=1:length(A1)
83                  AA1=[AA1,A1{i}]; AA2=[AA2,A2{i}];
84                  AA3=[AA3,A3{i}]; EE=[EE,EA1{i}];
85                  dt1=[dt1,DT1{i}];
86              end
87              for i=1:length(FF_ext_final)
88          fext=[fext,0,FF_ext_final{i}(1:end-1)*Aratio_slicer*Aratio_extr];
89          frob=[frob,0,F{i}(1:end-1)];
90              end
91           temlp=find(EE==0);  fext(temlp)=0;
92           for i=length(fext):-1:1
93                if fext(i)==0 && EE(i)~=0
94                    fext(i)=fext(i+1);
95                end
96           end
97     P_flag1=[AA1-AA1(1);AA2-AA2(1);AA3-AA3(1)];
98     x_flag=1:length(P_flag1);
99     ee{1}=EE;
100    aa1{1}=AA1-AA1(1);
101    aa2{1}=AA2-AA2(1);
102    aa3{1}=AA3-AA3(1);
103    ddtt1{1}=dt1;
104    G_code_anaimation1(ddtt1,ee,aa1,aa2,aa3,1,l,s,Rb,zEstrusore,Z);
105   aa1=[];aa2=[];aa3=[];
106   new_cam1
```

```
107     aa1{1}=cumsum(P_flag1(1,:));
108     aa2{1}=cumsum(P_flag1(2,:));
109     aa3{1}=cumsum(P_flag1(3,:));
110     save2melsoft1(DX_cam_V,P_flag1,strcat(title1,'_x_y_new_cam_red'),fext
            *100);
111     elseif opt_save==3
112     flag_layer_save=2:3;
113     xx=[];yy=[];zz=[];fext=[];frob=[]; ee=[];
114     opt_nr_layer1=menu('how many layers?','all','selection');
115     if opt_nr_layer1==1
116         nr_of_layer=1:length(XX_final);
117          for i=nr_of_layer
118              xx=[xx,XX_final{i}*1e3];yy=[yy,YY_final{i}*1e3];
119              zz=[zz,Z{i}(1)*ones(1,length(XX_final{i}))];
120              fext=[fext,0,FF_ext_final{i}(1:end-1)*Aratio_slicer*
                     Aratio_extr];
121              frob=[frob,0,F{i}(1:end-1)];
122              ee=[ee,[E_motlaw_lay{i},0]];
123              layers='all layers are saved';
124      end
125     else
126         nr_of_layer=flag_layer_save;
127         j=1;
128         for i=nr_of_layer
129              xx=[xx,XX_final{i}*1e3];yy=[yy,YY_final{i}*1e3];
130              zz=[zz,j*Z{1}(1)*ones(1,length(XX_final{i}))];
131              fext=[fext,0,FF_ext_final{i}(1:end-1)*Aratio_slicer*
                     Aratio_extr];
132              frob=[frob,0,F{i}(1:end-1)];
133              ee=[ee,[E_motlaw_lay{i},0]];
134              j=j+1;
135         end
136          templp=find(EE==0); fext(templp)=0;
137           for i=length(fext):-1:1
138               if fext(i)==0 && EE(i)~=0
139                  fext(i)=fext(i+1);
140               end
141           end
142 layers=['from ',num2str(nr_of_layer(1)),' to ',num2str(nr_of_layer(end))];
143    end
144 opt_test=menu('test_animation?','yes','no');
145 if opt_test==1
146      figure()
147      for i=1:length(xx)
148          plot3(xx(1:i),yy(1:i),zz(1:i))
149          if ee(i)==1
150          plot3(xx(i),yy(i),zz(i),'*r')
151          end
152           hold on
153          drawnow
154      end
```

```
155 else
156 end
157  M1=save2csv_gcode(xx,yy,zz,ee,frob,fext,title1,layers);
158   else
159   end
```

B.3.2 G_code_reader1

Listing B.6: G_code_reader1

```
1  function[X,Y,Z,E,F,ext_on,LAYER,dt_gcode,F_ext, DT_uniform]=G_code_reader(
       file,nr_layer,flag_plot,Layer)
2  d_filament=1.5; %mm
3  fileID = fopen(file);
4  status = 0;
5  X=[];Y=[];E=[];Z=[];
6  ext_on=[]; %on 1
7  flag_base=0; %if layer lower than 1... 1st layer -1 be care. in case of
       multiple layer with same number; else =0
8  opt11=menu('which slicer?','Cura Engine','Repetier-Host');
9  if opt11==1
10    opt10=menu('how many layer?','all','up to you');
11    if opt10==1
12    [layer_count]=G_code_layer1(1+flag_base,status,fileID);
13    nr_layer=layer_count;
14    LAYER=1:nr_layer;
15    else
16    LAYER=Layer;
17    end
18 % X,Y as Gcode
19 for layer=LAYER;
20 status = 0;
21 [TX,TY,TE,TZ,TF,T_ext_on,TX_ext,TY_ext]=G_code_layer(layer+flag_base,
       status,fileID);
22 %use this only if points are too close to be told apart
23 % trefx=diff(TX);
24 % trefy=diff(TY);
25 % trewx=find(trefx==0);
26 % trewy=find(trefy==0);
27 % if length(trewx)>1
28 %     TX(trewx+1)=TX(trewx+1)+0.001;
29 % end
30 % if length(trewy)>1
31 %     TY(trewy+1)=TY(trewy+1)+0.001;
32 % end
33 x_min(layer)=min(TX);
34 y_min(layer)=min(TY);
35 dx(layer)=(max(TX)-min(TX))/2;
36 dy(layer)=(max(TY)-min(TY))/2;
37
38 X{layer}=TX;Y{layer}=TY;E{layer}=TE;
39 Z{layer}=TZ;F{layer}=TF;
```

```
40
41 ext_on{layer}=T_ext_on;
42 X_ext{layer}=TX_ext;
43 Y_ext{layer}=TY_ext;
44
45 end
46 X_min=min(x_min)+max(dx);
47 Y_min=min(y_min)+max(dy);
48 %X,Y refered as the robot
49 dx=[];dy=[];
50
51 for layer=1:nr_layer
52 TX=X{layer}-X_min;
53 TY=Y{layer}-Y_min;
54
55 TX_ext=X_ext{layer}-X_min;
56 TY_ext=Y_ext{layer}-Y_min;
57
58 X{layer}=TX;
59 Y{layer}=TY;
60 X_ext{layer}=TX_ext;
61 Y_ext{layer}=TY_ext;
62 end
63 % G_code plot
64 if flag_plot==1
65 G_code_plot_initial(nr_layer,X,Y,X_min,Y_min,Z);
66 end
67 e_old=0;
68 for  layer=1:nr_layer
69     for i=1:length(F{layer})
70         if  F{layer}(i)==0 && i>1
71             F{layer}(i)=F{layer}(i-1);
72         end
73         if  F{layer}(i)==0 && i==1
74             F{layer}(i)=F_old;
75         end
76     F_old=F{layer}(end);
77     end
78    dx=diff([0,X{layer}]);
79    dy=diff([0,Y{layer}]);
80
81    DE=diff([e_old,E{layer}]);
82    flag=find(ext_on{layer}==1);
83    de=zeros(1,length(X{layer}));
84    de(flag)=DE;
85
86    e_old=E{layer}(end);
87    l=[];
88    amaxlaw=1000*60^2;
89    for i=1:length(F{layer})
90        if ext_on{layer}(i)==1
```

```
    l(i)=sqrt(dx(i).^2+dy(i).^2);
        else
    l(i)=sqrt(dx(i).^2+dy(i).^2);
        end
    l(1)=sqrt(dx(1).^2+dy(1).^2);
    end
    t1=l./F{layer};
    dt_gcode{layer}=t1;
    DT_uniform{layer}=[0,cumsum(t1)];
    F_ext{layer}=4*de.*F{layer}./(pi*l*d_filament^2);
%   F_ext{layer}=de./t1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
else
     opt12=menu('which?','Cura','slic3r');
     if opt12==2
         opt10=menu('how many layer?','all','up to you');
         if opt10==1
         [layer_count]=G_code_layer1(1+flag_base,status,fileID);
         nr_layer=layer_count;
         LAYER=1:nr_layer;
         else
     LAYER=1;
     end
for layer=LAYER;
status = 0;
[TX,TY,TE,TZ,TF,T_ext_on,TX_ext,TY_ext]=G_code_layer_rep(layer+flag_base,
    status,fileID);

flag_h=find(TX~=0);
flag_j=find(TY~=0);
flag_k=find(TZ~=0);
flag_e=find(T_ext_on~=-1);
lay=diff(flag_k);

TXX=TX(flag_h);
TYY=TY(flag_j);
TZZ=TZ(flag_k);
TEE=T_ext_on(flag_e);
%use this only if points are too close to be told apart
% trefx=diff(TXX);
% trefy=diff(TYY);
% trewx=find(trefx==0);
% trewy=find(trefy==0);
% if length(trewx)>1
%     TXX(trewx+1)=TXX(trewx+1)+0.001;
% end
% if length(trewy)>1
%     TYY(trewy+1)=TYY(trewy+1)+0.001;
% end
```

```
141 x_min(layer)=min(TXX);
142 y_min(layer)=min(TYY);
143 dx(layer)=(max(TXX)-min(TXX))/2;
144 dy(layer)=(max(TYY)-min(TYY))/2;
145
146 X{layer}=TXX;
147 Y{layer}=TYY;
148 E{layer}=TE(flag_h);
149 Z{layer}=TZZ;
150 F{layer}=TF(flag_h);
151
152 ext_on{layer}=TEE;
153 if layer~=1
154 ext_on{layer}=TEE(2:end);
155 else
156   ext_on{layer}=TEE;
157  end
158 X_ext{layer}=TX_ext;
159 Y_ext{layer}=TY_ext;
160
161 end
162 X_min=min(x_min)+max(dx);
163 Y_min=min(y_min)+max(dy);
164
165 dx=[];dy=[];
166 for layer=LAYER
167 TX=X{layer}-X_min;
168 TY=Y{layer}-Y_min;
169 X{layer}=TX;
170 Y{layer}=TY;
171 X_ext{layer}=TX_ext;
172 Y_ext{layer}=TY_ext;
173 end
174 if flag_plot==1
175       G_code_plot_initial(nr_layer,X,Y,X_min,Y_min,Z);
176 end
177 e_old=0;
178 for   layer=LAYER
179     for i=1:length(F{layer})
180        if   F{layer}(i)==0 && i>1
181             F{layer}(i)=F{layer}(i-1);
182        end
183        if   F{layer}(i)==0 && i==1
184             F{layer}(i)=F_old;
185        end
186     F_old=F{layer}(end);
187     end
188     for i=1:length(E{layer})
189        if   E{layer}(i)==0 && i>1
190             E{layer}(i)=E{layer}(i-1);
191        end
```

```
            if  E{layer}(i)==0 && i==1
                E{layer}(i)=e_old;
            end
        e_old=E{layer}(end);
        end

      if layer==1
          DE=diff([0,E{layer}]);
          e_old=E{layer}(end);
      else
          e_old=E{layer}(end);
          DE=diff([e_old,e_old+E{layer}]);
      end
      dx=diff([0,X{layer}]);
      dy=diff([0,Y{layer}]);

      l=[];
      for i=1:length(F{layer})
            if ext_on{layer}(i)==1
      l(i)=sqrt(dx(i).^2+dy(i).^2);
            else
      l(i)=sqrt(dx(i).^2+dy(i).^2);
            end
      l(1)=sqrt(dx(1).^2+dy(1).^2);
      end
      t1=l./F{layer};
      dt_gcode{layer}=t1;
      DT_uniform{layer}=[0,cumsum(t1)];
      F_ext{layer}=DE./t1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        else
        opt10=menu('how many layer?','all','up to you');
             if opt10==1
             [layer_count]=G_code_layer1(1+flag_base,status,fileID);
             nr_layer=layer_count;
             LAYER=1:nr_layer;
             else
             LAYER=Layer;
             end
flag_base=0;
% X,Y as Gcode
for layer=LAYER;
status = 0;
[TX,TY,TE,TZ,TF,T_ext_on,TX_ext,TY_ext]=G_code_layer(layer+flag_base,
    status,fileID);

x_min(layer)=min(TX);
y_min(layer)=min(TY);
dx(layer)=(max(TX)-min(TX))/2;
dy(layer)=(max(TY)-min(TY))/2;
```

```
242
243 X{layer}=TX;Y{layer}=TY;E{layer}=TE;
244 Z{layer}=TZ;F{layer}=TF;
245
246 ext_on{layer}=T_ext_on;
247 X_ext{layer}=TX_ext;
248 Y_ext{layer}=TY_ext;
249
250 end
251 X_min=min(x_min)+max(dx);
252 Y_min=min(y_min)+max(dy);
253 dx=[];dy=[];
254     for layer=1:nr_layer
255 TX=X{layer}-X_min;
256 TY=Y{layer}-Y_min;
257 TX_ext=X_ext{layer}-X_min;
258 TY_ext=Y_ext{layer}-Y_min;
259 X{layer}=TX;
260 Y{layer}=TY;
261 X_ext{layer}=TX_ext;
262 Y_ext{layer}=TY_ext;
263     end
264 if flag_plot==1
265 G_code_plot_initial(nr_layer,X,Y,X_min,Y_min,Z);
266 end
267 e_old=0;
268 for   layer=1:nr_layer
269      for i=1:length(F{layer})
270         if   F{layer}(i)==0 && i>1
271              F{layer}(i)=F{layer}(i-1);
272         end
273         if   F{layer}(i)==0 && i==1
274              F{layer}(i)=F_old;
275         end
276      F_old=F{layer}(end);
277      end
278     dx=diff([0,X{layer}]);
279     dy=diff([0,Y{layer}]);
280
281     DE=diff([e_old,E{layer}]);
282     flag=find(ext_on{layer}==1);
283     de=zeros(1,length(X{layer}));
284     de(flag)=DE;
285
286     e_old=E{layer}(end);
287     l=[];
288     amaxlaw=1000*60^2;
289     for i=1:length(F{layer})
290          if ext_on{layer}(i)==1
291     l(i)=sqrt(dx(i).^2+dy(i).^2);
292          else
```

```
293     l(i)=sqrt(dx(i).^2+dy(i).^2);
294         end
295     l(1)=sqrt(dx(1).^2+dy(1).^2);
296     end
297     t1=l./F{layer};
298     dt_gcode{layer}=t1;
299      DT_uniform{layer}=[0,cumsum(t1)];
300     F_ext{layer}=de./t1;
301 end
302      end
303 end
```

B.3.3 G_code_layer

Listing B.7: G_code_layer

```
1  function [TX,TY,TE,TZ,TF,T_ext_on,TX_ext,TY_ext]=G_code_layer(layer,status,
       fileID)
2  TX=[];TY=[];TZ=[];TF=[];
3  TX_ext=[];TY_ext=[];TE=[];
4  flag_iter=1;
5  T_ext_on=[]; %on 1
6  while status<1
7      tline = fgetl(fileID);
8      flag=0;
9      if tline(1)=='G' && tline(2)=='1'&& tline(3)==' '
10         i=1;
11         while flag<1
12         if tline(i)=='X'
13              TX=[TX,str2double(tline(i+1:i+6))];
14              TX_ext=[TX_ext,str2double(tline(i+1:i+6))];
15              flag=2;
16         end
17         i=i+1;
18         if i>=length(tline)
19             flag=2;
20         end
21         end
22     end
23     flag=0;
24     if tline(1)=='G' && tline(2)=='1'&& tline(3)==' '
25         i=1;
26             while flag<1
27             if tline(i)=='Y'
28                 TY=[TY,str2double(tline(i+1:i+6))];
29                 TY_ext=[TY_ext,str2double(tline(i+1:i+6))];
30                 T_ext_on=[T_ext_on,1];
31                 flag=2;
32             end
33             i=i+1;
34             if i>=length(tline)
35                 flag=2;
```

```
36              end
37              end
38      end
39      flag=0;
40      if tline(1)=='G' && tline(2)=='0'&& tline(3)==' '
41          i=1;
42              while flag<1
43              if tline(i)=='X'
44                  TX=[TX,str2double(tline(i+1:i+6))];
45                  flag=2;
46              end
47              i=i+1;
48              if i>=length(tline)
49                  flag=2;
50              end
51              end
52      end
53      flag=0;
54      if tline(1)=='G' && tline(2)=='0'&& tline(3)==' '
55          i=1;
56              while flag<1
57              if tline(i)=='Y'
58                  TY=[TY,str2double(tline(i+1:i+6))];
59                  T_ext_on=[T_ext_on,0];
60                  flag=2;
61              end
62              i=i+1;
63              if i>=length(tline)
64                  flag=2;
65              end
66              end
67      end
68      flag=0;
69      if tline(1)=='G' && tline(2)=='1'&& tline(3)==' '
70          i=1;
71              while flag<1
72              if tline(i)=='E'
73                  TE=[TE,str2double(tline(i+1:i+6))];
74                  flag=2;
75              end
76              i=i+1;
77              if i>=length(tline)
78                  flag=2;
79              end
80              end
81      end
82      flag=0;
83      if tline(1)=='G' && tline(2)=='0'&& tline(3)==' '
84          i=1;
85              while flag<1
86              if tline(i)=='Z'
```

```
87                  TZ=[TZ,str2double(tline(i+1:end))];
88                  flag=2;
89              end
90              i=i+1;
91              if i>=length(tline)
92                  flag=2;
93              end
94              end
95      end
96      flag=0;
97      if tline(1)=='G' && tline(2)=='0'&& tline(3)==' '
98          i=1;
99              while flag<1
100             if tline(i)=='F'
101                 TF=[TF,str2double(tline(i+1:i+4))];
102                 flag=2;
103             end
104             if i>=length(tline)
105                 TF=[TF,0];
106                 flag=2;
107             end
108             i=i+1;
109             end
110     end
111     flag=0;
112     if tline(1)=='G' && tline(2)=='1'&& tline(3)==' '
113         i=1;
114             while flag<1
115             if tline(i)=='F'
116                 TF=[TF,str2double(tline(i+1:i+4))];
117                 flag=2;
118             end
119             if i>=length(tline)
120                 TF=[TF,0];
121                 flag=2;
122             end
123             i=i+1;
124             end
125     end
126
127     if tline(1)==-1
128         status=2;
129     else
130         if layer>=0
131             if length(tline)>3 && tline(1)~='M'
132             if tline(2:6)=='LAYER'
133                 if length(tline)==8
134                 if tline(8)==num2str(layer)
135                     status=2;
136                 end
137                 end
```

```
138                 if length(tline)==9
139                 if tline(8:9)==num2str(layer)
140                     status=2;
141                 end
142                 end
143                 if length(tline)==10
144                 if tline(8:10)==num2str(layer)
145                     status=2;
146                 end
147                 end
148             end
149             end
150         end
151         if layer<0
152             if length(tline)>3 && tline(1)~='M'
153                 if tline(2:6)=='LAYER'
154                 if length(tline)==9
155                 if tline(8:9)==num2str(layer)
156                     status=2;
157                 end
158                 end
159                 end
160             end
161         end
162     end
163 end
164 flag_iter=flag_iter+1;
165 end
```

B.3.4 G_code_layer_rep (Slic3r)

Listing B.8: G_code_layer_rep

```
1  function [TX,TY,TE,TZ,TF,T_ext_on,TX_ext,TY_ext]=G_code_layer_rep(layer,
       status,fileID)
2  TX=[];TY=[];TZ=[];TF=[];
3  TX_ext=[];TY_ext=[];TE=[];
4  flag_iter=1;
5  T_ext_on=[]; %on 1
6  jj=1;
7  while status<1
8      tline = fgetl(fileID);
9      if length(tline)==3 && tline(1)~='G'
10         if tline=='END'
11          status=2;
12         end
13     else
14     flag=0;
15     if tline(1)=='G' && tline(2)=='1'&& tline(3)==' '
16          i=1;
17         while flag<1
18         if tline(i)=='X'
```

```
19                  TX(jj)=str2double(tline(i+1:i+6));
20                  TX_ext(jj)=str2double(tline(i+1:i+6));
21                  flag=2;
22          end
23          i=i+1;
24          if i>=length(tline)
25                  TX(jj)=0;
26                  TX_ext(jj)=0;
27                  flag=2;
28          end
29          end
30      end
31      flag=0;
32      if tline(1)=='G' && tline(2)=='1'&& tline(3)==' '
33           i=1;
34          while flag<1
35          if tline(i)=='Y'
36                  TY(jj)=str2double(tline(i+1:i+6));
37                  TY_ext(jj)=str2double(tline(i+1:i+6));
38                  flag=2;
39          end
40          i=i+1;
41          if i>=length(tline)
42                  TY(jj)=0;
43                  TY_ext(jj)=0;
44                  flag=2;
45          end
46          end
47       end
48       flag=0;
49  if tline(1)=='G' && tline(2)=='1'&& tline(3)==' '&& tline(4)=='X'
50           i=1;
51          while flag<1
52          if tline(i)=='E'
53                  TE(jj)=str2double(tline(i+1:i+6));
54                  T_ext_on(jj)=1;
55                  flag=2;
56          end
57          i=i+1;
58          if i>=length(tline)
59                  TE(jj)=0;
60                  T_ext_on(jj)=0;
61                  flag=2;
62          end
63          end
64  end
65       flag=0;
66       if tline(1)=='G' && tline(2)=='1'&& tline(3)==' '
67           i=1;
68          while flag<1
69          if tline(i)=='Z'
```

```
70                  TZ(jj)=str2double(tline(i+1:i+6));
71                  T_ext_on(jj)=-1;
72                  flag=2;
73          end
74          i=i+1;
75          if i>=length(tline)
76                  TZ(jj)=0;
77                  flag=2;
78          end
79          end
80       end
81       flag=0;
82       if tline(1)=='G' && tline(2)=='1'&& tline(3)==' '
83          i=1;
84          while flag<1
85          if tline(i)=='F'
86                  TF(jj)=str2double(tline(i+1:i+4));
87                  flag=2;
88          end
89          if i>=length(tline)
90                  TF(jj)=0;
91                  flag=2;
92          end
93          i=i+1;
94          end
95        end
96      end
97
98      if tline(1)==-1
99          status=2;
100     else
101         if layer>=0
102         if length(tline)>3 && tline(1)~='M'
103             if tline(2:6)=='LAYER'
104                 if length(tline)==8
105                 if tline(8)==num2str(layer)
106                     status=2;
107                 end
108                 end
109                 if length(tline)==9
110                 if tline(8:9)==num2str(layer)
111                     status=2;
112                 end
113                 end
114                 if length(tline)==10
115                 if tline(8:10)==num2str(layer)
116                     status=2;
117                 end
118                 end
119             end
120         end
```

```
121         end
122         if layer<0
123         if length(tline)>3 && tline(1)~='M'
124            if tline(2:6)=='LAYER'
125               if length(tline)==9
126                  if tline(8:9)==num2str(layer)
127                     status=2;
128                  end
129               end
130            end
131         end
132         end
133      end
134      jj=jj+1;
135   end
136   flag_iter=flag_iter+1;
```

B.3.5 G_code_inter_points

Listing B.9: G_code_inter_points

```
1  function[XX_final,YY_final,EE_final,FF_final,FF_ext_final]=
       G_code_inter_points(LAYER,X,Y,Z,E,ext_on,dL,F,F_ext)
2  for layer=LAYER;
3      flag_x=[0,X{layer}*1e-3,0]; %[m];
4      flag_y=[0,Y{layer}*1e-3,0]; %[m];
5      dx{layer}=diff(flag_x);
6      dy{layer}=diff(flag_y);
7      l_seg{layer}=sqrt(dx{layer}.^2+dy{layer}.^2);
8      L(layer)=sum(l_seg{layer});
9  end
10 for layer=LAYER
11     temp=l_seg{layer};
12     temp2=[0,ext_on{layer},0];
13     flag_x=[0,X{layer}*1e-3,0]; %[m];
14     flag_y=[0,Y{layer}*1e-3,0]; %[m];
15     flag_f=[0,F{layer},F{layer}(end)];
16     flag_fe=[0,F_ext{layer},F_ext{layer}(end)];
17     X_final=[];Y_final=[];E_final=[];
18     F_final=[];F_ext_final=[];
19 for i=1:length(temp)
20     py_flag=[];px_flag=[];pz_flag=[];
21     pf_flag=[];pfe_flag=[];
22 if temp(i)>dL*1e-3
23     flag(i)=ceil(temp(i)*1e3/dL)+1; % dL=10 means 1cm
24     py_flag=linspace(flag_y(i),flag_y(i+1),flag(i));
25     px_flag=linspace(flag_x(i),flag_x(i+1),flag(i));
26     pf_flag=flag_f(i+1)*ones(flag(i),1)';
27     pfe_flag=flag_fe(i+1)*ones(flag(i),1)';
28     if temp2(i+1)==0
29        pz_flag=zeros(flag(i),1)';
30     end
```

```
31      if temp2(i+1)==1
32          pz_flag=ones(flag(i),1)';
33      end
34      X_final=[X_final,px_flag(1:end-1)];
35      Y_final=[Y_final,py_flag(1:end-1)];
36      E_final=[E_final,pz_flag(2:end)];
37      F_final=[F_final,pf_flag(2:end)];
38      F_ext_final=[F_ext_final,pfe_flag(2:end)];
39  else
40      X_final=[X_final,flag_x(i)];
41      Y_final=[Y_final,flag_y(i)];
42      E_final=[E_final,temp2(i+1)];
43      F_final=[F_final,flag_f(i+1)];
44      F_ext_final=[F_ext_final,flag_fe(i+1)];
45  end
46  end
47      XX_final{layer}=[X_final,0];
48      YY_final{layer}=[Y_final,0];
49      EE_final{layer}=[0,E_final(1:end-1)];
50      FF_final{layer}=[F_final,F_final(end)];
51      FF_ext_final{layer}=[F_ext_final,F_ext_final(end)];
52  end
```

B.3.6 G_code_time1 (data processing)

Listing B.10: G_code_time1

```
1  function[alpha1,alpha2,alpha3,l_seg,DT_motlaw_lay,dt_gcode]=G_code_time1(
       LAYER,XX_final,YY_final,EE_final,Z,zEstrusore,l,s,Rb,FF_final)
2      for layer=LAYER
3      flag_x=[];flag_y=[];
4      flag_e=[];flag_z=[];
5      flag_x=XX_final{layer}; %[m];
6      flag_y=YY_final{layer}; %[m];
7      flag_e=EE_final{layer};
8      flag_z=zEstrusore-Z{layer}(1)*ones(1,length(flag_x))*1e-3; %[m];
9      tc_x=[];
10     tc_y=[];
11     alpha_1=[];alpha_2=[];alpha_3=[];
12     for j=1:length(flag_x)
13     p=[flag_x(j);flag_y(j);flag_z(j)];
14     alpha_p=linear_inverse_kinematics(p,l,s,Rb);
15     alpha_1(j)=alpha_p(1);
16     alpha_2(j)=alpha_p(2);
17     alpha_3(j)=alpha_p(3);
18     end
19     alpha1{layer}=alpha_1;
20     alpha2{layer}=alpha_2;
21     alpha3{layer}=alpha_3;
22     kit{layer}=[length(alpha1{layer}),length(flag_x)];
23     dx{layer}=diff(flag_x);
24     dy{layer}=diff(flag_y);
```

```
25      l_seg{layer}=sqrt(dx{layer}.^2+dy{layer}.^2);
26      dx=[];dy=[];
27      dx=diff([0,XX_final{layer}]);
28      dy=diff([0,YY_final{layer}]);
29      lll=[];
30      for i=1:length(FF_final{layer})
31      lll(i)=sqrt(dx(i).^2+dy(i).^2);
32      lll(1)=sqrt(dx(1).^2+dy(1).^2+(Z{layer}(1)*1e-3).^2);
33      end
34      t1=lll./(FF_final{layer}*1e-3);
35      dt_gcode{layer}=t1;
36      DT_motlaw_lay{layer}=[cumsum(t1)];
37      dx=[];dy=[];
38  end
```

B.3.7 G_code_slicer (agglomeration)

Listing B.11: G_code_slicer

```
1   function [A1,A2,A3,EA1,DT1]=G_code_slicer(LAYER,DT_motlaw_lay,
        nr_slicer_points,E_motlaw_lay,alpha_motlaw1_final,alpha_motlaw2_final,
        alpha_motlaw3_final,l_seg,delta_time,l,s,Rb,zEstrusore,Z)
2   A_1=[];A_2=[];A_3=[];EA1_1=[];
3   DT1_1=[];L1_1=[];
4   vect_lay=0;
5   j=1;
6   jkp=0;
7   A1=[];A2=[];A3=[];EA1=[];DT1=[];
8   for i=LAYER
9       temp_now(i)=length(DT_motlaw_lay{i});
10      if sum(temp_now)<nr_slicer_points
11          DT1_1=[DT1_1,jkp+DT_motlaw_lay{i}];
12          A_1=[A_1,alpha_motlaw1_final{i}];
13          A_2=[A_2,alpha_motlaw2_final{i}];
14          A_3=[A_3,alpha_motlaw3_final{i}];
15          EA1_1=[EA1_1,[E_motlaw_lay{i},0]];
16          L1_1=[L1_1,l_seg{i}];
17           jkp=DT1_1(end);
18      else
19          A1{j}=A_1;
20          A2{j}=A_2;
21          A3{j}=A_3;
22          EA1{j}=EA1_1;
23          DT1{j}=DT1_1;
24          L1{j}=L1_1;
25          temp_now(1:i-1)=0;
26          vect_lay(j)=i-1;
27          A_1=[];A_2=[];A_3=[];
28          DT1_1=[];EA1_1=[];L1_1=[];
29          cont=1;
30          DT1_1=[DT1_1,jkp+DT_motlaw_lay{i}];
31          A_1=[A_1,alpha_motlaw1_final{i}];
```

```
32          A_2=[A_2,alpha_motlaw2_final{i}];
33          A_3=[A_3,alpha_motlaw3_final{i}];
34          EA1_1=[EA1_1,[E_motlaw_lay{i},0]];
35          L1_1=[L1_1,l_seg{i}];
36          jkp=DT1_1(end);
37          j=j+1;
38      end
39       if i==LAYER(end)
40           A1{j}=A_1;
41          A2{j}=A_2;
42          A3{j}=A_3;
43          EA1{j}=EA1_1;
44          DT1{j}=DT1_1;
45          L1{j}=L1_1;
46          vect_lay(j)=i;
47       end
48  end
49      disp('splitting completed! here are the layer subdivision:')
50      layer_subdivision=vect_lay
```

B.3.8 save2melsoft1 (saving function)

Listing B.12: save2melsoft1

```
1  function save2melsoft(tempo,Q,figura_geometrica,e_ext)
2  if length(Q)<2040
3  dimension_cam_master=length(tempo);
4
5  M=[2;101;dimension_cam_master;0;0];
6  M1=[];  M2=[];  M3=[]; ME=[];
7  for j=1:length(Q)
8      M1=[M1;tempo(j)*100;Q(1,j)*10^7];
9      M2=[M2;tempo(j)*100;Q(2,j)*10^7];
10     M3=[M3;tempo(j)*100;Q(3,j)*10^7];
11     ME=[ME;tempo(j)*100;e_ext(j)];
12 end
13 fprintf('\n\n ************************* \n SAVE 2 MELSOFT FUNCTION: \n')
14 %joint1
15 Mjoint1=[M;M1;0];
16
17 dlmwrite(strcat(figura_geometrica,'_J1.csv'),Mjoint1,'precision','%.0f');
18 display(strcat('salvato file :',figura_geometrica,'_J1.csv'))
19 beep
20
21 %joint2
22 Mjoint2=[M;M2;0];
23 dlmwrite(strcat(figura_geometrica,'_J2.csv'),Mjoint2,'precision','%.0f');
24 display(strcat('salvato file :',figura_geometrica,'_J2.csv'))
25 beep
26
27 %joint1
28 Mjoint3=[M;M3;0];
```

```
29 dlmwrite(strcat(figura_geometrica,'_J3.csv'),Mjoint3,'precision','%.0f');
30 display(strcat('salvato file :',figura_geometrica,'_J3.csv'))
31 beep
32 %ext
33 Mjointe=[M;ME;0];
34 dlmwrite(strcat(figura_geometrica,'_EXT.csv'),Mjointe,'precision','%.0f');
35 display(strcat('salvato file :',figura_geometrica,'_EXT.csv'))
36 beep
37
38
39 %stampa altri dati
40 fprintf('\n')
41 fprintf('numero punti camma master = %d \n', dimension_cam_master)
42 fprintf('lunghhezza tempo = %f micrometri \n', tempo(end))
43 % fprintf('velocità camma master = %f mm/min \n', vel_camma_master)
44 else
45     flag= ceil(length(Q)/2040);
46     for jj=1:flag
47         j=0;
48         if jj~=flag
49         dimension_cam_master=2040;
50         M=[2;101;dimension_cam_master;0;0];
51         M1=[];  M2=[];  M3=[]; ME=[];
52         for j=1+2040*(jj-1):2040*jj
53             M1=[M1;tempo(j)*100;Q(1,j)*10^7];
54             M2=[M2;tempo(j)*100;Q(2,j)*10^7];
55             M3=[M3;tempo(j)*100;Q(3,j)*10^7];
56             ME=[ME;tempo(j)*100;e_ext(j)];
57         end
58         else
59         M1=[];  M2=[];  M3=[]; ME=[];
60         for j=1+2040*(jj-1):length(Q)
61             M1=[M1;tempo(j)*100;Q(1,j)*10^7];
62             M2=[M2;tempo(j)*100;Q(2,j)*10^7];
63             M3=[M3;tempo(j)*100;Q(3,j)*10^7];
64             ME=[ME;tempo(j)*100;e_ext(j)];
65         end
66         dimension_cam_master=length(M1)/2+1;
67         M=[2;101;dimension_cam_master;0;0;0;0];
68
69         end
70
71
72 fprintf('\n\n ************************* \n SAVE 2 MELSOFT FUNCTION: \n')
73 %joint1
74 Mjoint1=[];
75 Mjoint1=[M;M1;0];
76
77 dlmwrite(strcat(figura_geometrica,'_part_',num2str(jj),'_J1.csv'),Mjoint1,
       'precision','%.0f');
78 display(strcat('salvato file :',figura_geometrica,'_part_',num2str(jj),'
```

```
    _J1.csv'))
beep

%joint2
Mjoint2=[];
Mjoint2=[M;M2;0];
dlmwrite(strcat(figura_geometrica,'_part_',num2str(jj),'_J2.csv'),Mjoint2,
    'precision','%.0f');
display(strcat('salvato file :',figura_geometrica,'_part_',num2str(jj),'
    _J2.csv'))
beep

%joint3
Mjoint3=[];
Mjoint3=[M;M3;0];
dlmwrite(strcat(figura_geometrica,'_part_',num2str(jj),'_J3.csv'),Mjoint3,
    'precision','%.0f');
display(strcat('salvato file :',figura_geometrica,'_part_',num2str(jj),'
    _J3.csv'))
beep

%ext
Mjointe=[M;ME;0];
dlmwrite(strcat(figura_geometrica,'_part_',num2str(jj),'_EXT.csv'),Mjointe
    ,'precision','%.0f');
display(strcat('salvato file :',figura_geometrica,'_part_',num2str(jj),'
    _EXT.csv'))
beep

%stampa altri dati
fprintf('\n')
fprintf('numero punti camma master = %d \n', dimension_cam_master)
fprintf('lunghhezza tempo = %f micrometri \n', tempo(end))
% fprintf('velocità camma master = %f mm/min \n', vel_camma_master)
    end
    flag_title_v=[figura_geometrica,'_part_'];
end
```

B.3.9 save2txt_gcode (saving function)

Listing B.13: save2txt_gcode

```
function M1=save2csv_gcode(xx,yy,zz,ee,frob,fext,figura_geometrica,layers)
frob;
fext;
dimensione_camma_master=length(xx);
M={';txt file similar to G-code';[';used layers:',0,layers];num2str(
    dimensione_camma_master);}
M1=[];  M2=[];  M3=[]; ME=[];
for i=1:length(xx)
    if abs(xx(i))<1e-7
        xx(i)=0;
```

```
10      end
11       if abs(yy(i))<1e-7
12          yy(i)=0;
13      end
14  end
15  for j=1:length(xx)
16          if ee(j)==1
17      M1{j}=['G1',0,'X',num2str(xx(j),6),'',0,'Y',num2str(yy(j),6),0,'Z',
            num2str(zz(j),6)];
18      elseif ee(j)==0
19      M1{j}=['G0',0,'X',num2str(xx(j),6),'',0,'Y',num2str(yy(j),6),0,'Z',
            num2str(zz(j),6)];
20      end
21      clc
22  end
23  fprintf('\n\n ************************* \n SAVE 2 MELSOFT FUNCTION: \n')
24  %joint1
25  Mjoint1{1}=M{1};
26  Mjoint1{2}=M{2};
27  Mjoint1{3}=M{3};
28  for i=1:length(M1)
29  Mjoint1{i+3}=M1{i};
30  end
31  fid = fopen( [figura_geometrica,'.txt'], 'w' ) ;
32   for cId = 1 : numel( Mjoint1 )
33      fprintf( fid, '%s', Mjoint1{cId} ) ;
34      fprintf( fid, '\r\n' ) ;
35   end
36  display(strcat('salvato file :',figura_geometrica,'_J1.csv'))
37   beep
```

B.3.10 G_code_plot_initial

Listing B.14: G_code_plot_initial

```
1  function[]=G_code_plot_initial(nr_layer,X,Y,X_min,Y_min,Z)
2  dx=[];dy=[];
3  opt=menu('2d or 3d','2D','3D','exit');
4  if opt==1;
5  for layer=2:nr_layer;
6  TX=X{layer}-X_min;
7  TY=Y{layer}-Y_min;
8  plot(TX,TY)
9  hold on
10 grid on
11 axis equal
12 drawnow
13 end
14 end
15 if opt==2
16 for layer=2:1:nr_layer;
17 TX=X{layer}-X_min;
```

```
TY=Y{layer}-Y_min;
ZZ=Z{layer};
plot3(TX,TY,ZZ*ones(length(TX)))
hold on
grid on
axis equal
drawnow
end
end
if opt==3
end
```

B.3.11 layer_reducer

Listing B.15: layer_reducer

```
L_old=length(alpha1{2});
L_index=[];L_counter=[];counter_i=0;
for i=3:length(alpha1)
L_alpha(i)=length(alpha1{i});
if abs(L_alpha(i)-L_old)>2
    L_index=[L_index,i];
    L_counter=[L_counter,counter_i+1];
    counter_i=0;
else
    counter_i=counter_i+1;
end
if i==length(alpha1)
    L_index=[L_index,i];
    L_counter=[L_counter,counter_i+1];
    counter_i=0;
end
L_old=L_alpha(i);
end
disp('first layer of transition:')
L_fist=L_index-L_counter
disp('number of repetition')
L_counter=L_counter
```

B.4 square (input equation)

B.4.1 MAIN_SQUARE

Listing B.16: MAIN_SQUARE

```
clear;clc;

%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% robot data
l=0.595;    Rb=0.198;    s=0.45651;   zEstrusore=0.8;
```

```
6  vMaxQ=[0.05;0.05;0.05];                %[m/s]
7  aMaxQ=[0.200;0.200;0.200];             %[m/s^2]
8  v_max=0.1;
9  a_max=1200000;%[mm/min^2]
10 d_max=2*a_max;
11
12 d_p=14; %[mm]
13
14 % square data
15 A=40; %[mm];
16 B=31; %[mm];
17 r=0.45; %[mm] d/2
18 z=1.5;  %[mm]
19 v_ext=2.5*1e-3;% 10e-3 %[m/s];
20
21 LAYER=1;
22
23 % intermediate points generatio data
24 dL=10; %[mm]
25
26 % motion law data
27 n_motlaw=50;
28 c1=1/3;
29
30 % saving data
31 title1='picture';%print_05_12_df_v_fm_50point_dL_v=2.5e-3
32
33 %next start data
34 dz=0.8*(2*r); %[mm];
35
36 %circular section
37 % A_robot=(pi*dz^2/4);
38 % v_robot=(pi*(2*r)^2/4)/(A_robot)*v_ext;
39 % r_gap=dz/2;
40
41 %square
42 v_robot=v_ext;
43 r_gap=0.5*pi*(2*r)^2/(4*dz)*(v_ext/v_robot);
44 A_robot=r_gap*dz;
45
46 N=(A-B)/4/r_gap;
47
48 flag_taper=0;
49 teta=pi/4;
50
51 %bazier_ret_par data
52 opt=menu('Bazier or mot.law?','bazier','motlaw');
53 if opt==1
54 flag_method=1;
55 else
56  flag_method=0;
```

```
end
delta=0.5e-3;         %[m] indica il comportamento del raggio di curvatura
    nella traiettoria rette-para
dt=0.001;             %[s] periodo campionamento. 0.001 normale. 0.00001
    memorizzato.
vDep=v_ext;            %[m/s] velocità curvilinea
aAcc=60e-3;           %[m/s^2] modulo accelerazione curvilinea iniziale
aDec=60e-3;           %[m/s^2] modulo decelerazioione curvilinea finale

salto_punti_decampionatore=125;
title='square_bazier_mit';

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%bazier ret-par
if flag_method==1
    ret_per_bazier_script
    % next layer
    pos_ini=[0,0,-dz*1e-3]; %flag_z -2*r*
    alpha_pf=Q_dec(:,end);
    alpha_pi=[0;0;0]-dz*1e-3;
    format short g
    delta_pos=(alpha_pi-alpha_pf)*1e6
    format
else

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%square gen
[X,Y,Z,T_1_temp,V]=square_gen(A,B,r_gap,N,v_ext,z);

%intermediate points generatio
[XX_final,YY_final,EE_final,V_final]=square_inter_points(LAYER,X,Y,Z,dL,V,
    v_ext);

%adim. motion law generation
[x,d2,s2,acc2,d1,s1,acc,ca,cv]=square_mot_law_adim_cv(n_motlaw,c1);

%delta time generation 1
[DT_motlaw,delta_time,alpha1,alpha2,alpha3,l_seg]=square_time1(LAYER,
    XX_final,YY_final,EE_final,Z,zEstrusore,l,s,Rb,vMaxQ,aMaxQ,cv,ca,
```

```
    v_robot, V_final);
96
97  %motion law assignement
98
99  [alpha_motlaw1_final, alpha_motlaw2_final, alpha_motlaw3_final,
        valpha_motlaw1_final, valpha_motlaw2_final, valpha_motlaw3_final,
        aalpha_motlaw1_final, aalpha_motlaw2_final, aalpha_motlaw3_final,
        DT_motlaw_lay,  E_motlaw_lay]=square_mot_law_ass(LAYER, alpha1, alpha2,
        alpha3, x, d2, s2, acc2, d1, s1, acc, ca, cv, delta_time, aMaxQ, vMaxQ, EE_final);
100 VEL_max=[max(abs(valpha_motlaw1_final{LAYER})),max(abs(
        valpha_motlaw2_final{LAYER})),max(abs(valpha_motlaw3_final{LAYER}))]
101 ACC_max=[max(abs(aalpha_motlaw1_final{LAYER})),max(abs(
        aalpha_motlaw2_final{LAYER})),max(abs(aalpha_motlaw3_final{LAYER}))]
102
103 %mot_law_plot
104 G_code_mot_law_plot(LAYER, alpha_motlaw1_final, alpha_motlaw2_final,
        alpha_motlaw3_final, valpha_motlaw1_final, valpha_motlaw2_final,
        valpha_motlaw3_final, aalpha_motlaw1_final, aalpha_motlaw2_final,
        aalpha_motlaw3_final, DT_motlaw_lay);
105
106 %animation
107 G_code_anaimation1(DT_motlaw_lay, E_motlaw_lay, alpha_motlaw1_final,
        alpha_motlaw2_final, alpha_motlaw3_final, LAYER, l, s, Rb, zEstrusore, Z);
108
109 %diameter as function of the vel
110 d_dep_check
111
112 % saving
113 save_menu
114
115 %extruder vel
116 extruder_vel
117
118 % next layer
119 %%
120 pos_fin=[XX_final{LAYER}(end), YY_final{LAYER}(end), zEstrusore-z];
121 pos_ini=[0,0, zEstrusore-z-dz*1e-3];
122 alpha_pf=linear_inverse_kinematics(pos_fin', l, s, Rb);
123 alpha_pi=linear_inverse_kinematics(pos_ini', l, s, Rb);
124 format short g
125
126 if flag_taper==1
127 taper_1=dz*1e-3/tan(teta);
128 pos_fin=[XX_final{LAYER}(end), YY_final{LAYER}(end), zEstrusore-z];
129 pos_ini=[taper_1, 0, zEstrusore-z];
130 alpha_pf=linear_inverse_kinematics(pos_fin', l, s, Rb);
131 alpha_pi=linear_inverse_kinematics(pos_ini', l, s, Rb);
132 delta_pos1=-(alpha_pf-alpha_pi)*1e6
133 else
134 delta_pos1=-([alpha_motlaw1_final{LAYER}(end)-alpha_motlaw1_final{LAYER
        }(1); alpha_motlaw2_final{LAYER}(end)-alpha_motlaw2_final{LAYER}(1);
```

```
    alpha_motlaw3_final{LAYER}(end)-alpha_motlaw3_final{LAYER}(1)])*1e6
135
136 end
137 delta_pos2=-dz*ones(3,1)*1e3
138 format
139 end
```

B.4.2 square_gen (square generation)

Listing B.17: square_gen

```
1  function [X,Y,Z,T_1_temp,V]=square_gen(A,B,r,N,v_ext,z)
2  X=[];Y=[];Vx=[];Vy=[];Vz=[];V=[];
3  for i=1:N
4   As=A-r;
5   Bs=A-2*r;
6   a=r*(i-1);
7   if i==1
8   x=[a,a,Bs-a,Bs-a];
9   y=[a,As-a,As-a,r*i];
10  X=[X,x];
11  Y=[Y,y];
12  elseif i==2
13
14  x=[a+r*(i-1),a+r*(i-1),Bs-a-r*(i-1),Bs-a-r*(i-1)];
15  y=[a,As-a-r*(i-1),As-a-r*(i-1),a+r*(i)];
16  X=[X,x];
17  Y=[Y,y];
18  else
19  x=[a+r*(i-1),a+r*(i-1),Bs-a-r*(i-1),Bs-a-r*(i-1)];
20  y=[a+r*(i-2),As-a-r*(i-1),As-a-r*(i-1),a+r*(i)];
21  X=[X,x];
22  Y=[Y,y];
23  end
24
25 end
26  X=[X,X(end)-B-r];
27  Y=[Y,Y(end)];
28  for i=1:N
29  vx=[0,1,0,1];
30  vy=[1,0,1,0];
31  Vx=[Vx,vx];
32  Vy=[Vy,vy];
33  Vz=[Vz,[0,0,0,0]];
34
35  end
36  V=[Vx;Vy;Vz];
37  L_1_temp=sqrt((diff(X)*1e-3).^2+(diff(Y)*1e-3).^2);
38  T_1_temp=L_1_temp/v_ext;
39  T_1_temp=[0,cumsum(T_1_temp)];
40  Z=z*ones(1,length(X));
41
```

```
42 %%
43 figure()
44 plot(X,Y)
45 axis equal
46 hold on
47 C=(A-B)/2;
48 plot([-r,-r],[0,A],'r',[-r,A-r],[A,A],'r',[A-r,A-r],[0,A],'r',[-r,A-r
       ],[0,0],'r')
49 plot([C-r,C-r],[C,A-C],'r',[C-r,A-C-r],[A-C,A-C],'r',[A-C-r,A-C-r],[C,A-C
       ],'r',[C-r,A-C-r],[C,C],'r')
```

B.4.3 shape8_gen (eight shape generation)

N=1 eight shape, N=2 leaning x eight shape, N=3 leaning y eight shape

Listing B.18: shape8_gen

```
1  function [X,Y,Z,T_1_temp,V]=shape8_gen(A,B,r,N,v_ext,z)
2  X=[];Y=[];
3  Vx=[];Vy=[];Vz=[];
4  if N==1
5  x=[0,-A,A,-A,A,0];
6  y=[0,B,B,-B,-B,0];
7  vy=[1,0,1,0,1];
8  vx=[1,1,1,1,1];
9  X=[X,x];Y=[Y,y];
10 Vx=[Vx,vx];Vy=[Vy,vy];Vz=[Vz,[0,0,0,0,0]];
11 V=[Vx;Vy;Vz];
12 elseif N==2
13 x=[0,0,A,-A,0,0];
14 y=[0,B,B,-B,-B,0];
15 vy=[1,0,1,0,1];
16 vx=[1,1,1,1,1];
17 X=[X,x];Y=[Y,y];
18 Vx=[Vx,vx];Vy=[Vy,vy];Vz=[Vz,[0,0,0,0,0]];
19 V=[Vx;Vy;Vz];
20 elseif N==3
21 x=[0,-A,-A,A,A,0];
22 y=[0,0,B,-B,0,0];
23 vy=[1,0,1,0,1];
24 vx=[1,1,1,1,1];
25 X=[X,x];Y=[Y,y];
26 Vx=[Vx,vx];Vy=[Vy,vy];Vz=[Vz,[0,0,0,0,0]];
27 V=[Vx;Vy;Vz];
28 end
29  L_1_temp=sqrt((diff(X)*1e-3).^2+(diff(Y)*1e-3).^2);
30  T_1_temp=L_1_temp/v_ext;
31  T_1_temp=[0,cumsum(T_1_temp)];
32  Z=z*ones(1,length(X));
33  figure()
34  plot(X,Y)
35  axis equal
```

B.4.4 square_mot_law_adim_cv (motion law definition)

Listing B.19: square_mot_law_adim_cv

```
function [x,d2,s2,acc2,d1,s1,acc,ca,cv]=square_mot_law_adim_cv(n_motlaw,c1
    )
% do not change this
    c=1/2;
    x=0:1/n_motlaw:1;
    j=1;
    for i=x
    [d2(j),s2(j),acc2(j)]= const_a_motlaw (i,c);
    j=j+1;
    end
    % do please change it to your necess. accordingly
    opt=menu('mot_law','v=cost','const_a_motlaw','par','asym_a','cyclo');
    if opt==1
    c=c1;
    j=1;
    for i=x
    [d1(j),s1(j),acc(j)]= const_v(i);
    j=j+1;
    end
    ca=0;cv=1;
    end
    if opt==2
    c=c1;
    j=1;
    for i=x
    [d1(j),s1(j),acc(j)]= const_a_motlaw (i,c);
        j=j+1;
    end
    ca=1/(c*(1-c));cv=1/((1-c));
    end
    if opt==3
    c=c1;
    j=1;
    for i=x
    [d1(j),s1(j),acc(j)]= par_motlaw(i);
    j=j+1;
    end
    ca=6;cv=1.5;
    end
     if opt==4
    c=c1;
    j=1;
    for i=x
    [d1(j),s1(j),acc(j)]= asym_a_motlaw(i,c);
    j=j+1;
    end
    ca=2/((1-c));cv=2;
    end
```

```
48      if opt==5
49      c=c1;
50      j=1;
51      for i=x
52      [d1(j),s1(j),acc(j)]= cycloidal_motlaw(i);
53      j=j+1;
54      end
55      ca=2*pi;cv=2;
56      end
```

B.4.5 square_mot_law_assignment

Listing B.20: square_mot_law_ass

```
1  function[alpha_motlaw1_final,alpha_motlaw2_final,alpha_motlaw3_final,
       valpha_motlaw1_final,valpha_motlaw2_final,valpha_motlaw3_final,
       aalpha_motlaw1_final,aalpha_motlaw2_final,aalpha_motlaw3_final,
       DT_motlaw_lay, E_motlaw_lay]=square_mot_law_ass(LAYER,alpha1,alpha2,
       alpha3,x,d2,s2,acc2,d1,s1,acc,ca,cv,delta_time,aMaxQ,vMaxQ,EE_final)
2  % ass. main mot. law if not feasible uses a bang-bang profile.
3  layer=LAYER;
4  a1=[];a2=[];a3=[];
5  layer_flag=layer;
6       for i=layer_flag
7        a1=alpha1{i};
8        a2=alpha2{i};
9        a3=alpha3{i};
10       dx1=diff(a1);
11       dx2=diff(a2);
12       dx3=diff(a3);
13       max_da=[max(abs(dx1)),max(abs(dx2)),max(abs(dx3))];
14       min_da=[min(abs(dx1)),min(abs(dx2)),min(abs(dx3))];
15       alpha_motlaw1=[];alpha_motlaw2=[];
16       alpha_motlaw3=[];valpha_motlaw1=[];
17       valpha_motlaw2=[];valpha_motlaw3=[];
18       aalpha_motlaw1=[];aalpha_motlaw2=[];
19       aalpha_motlaw3=[];E_motlaw=[];
20       DT_motlaw=0;
21       for j=1:length(dx1)
22       alpha_temp1=a1(j)+dx1(j).*d1;
23       alpha_temp2=a2(j)+dx2(j).*d1;
24       alpha_temp3=a3(j)+dx3(j).*d1;
25       valpha_temp1=dx1(j).*s1/delta_time(j);
26       valpha_temp2=dx2(j).*s1/delta_time(j);
27       valpha_temp3=dx3(j).*s1/delta_time(j);
28       aalpha_temp1=dx1(j).*acc/delta_time(j)^2;
29       aalpha_temp2=dx2(j).*acc/delta_time(j)^2;
30       aalpha_temp3=dx3(j).*acc/delta_time(j)^2;
31     if (max(aalpha_temp1)>aMaxQ(1) || max(aalpha_temp2)>aMaxQ(1) || max(
         aalpha_temp3)>aMaxQ(1) ) %|| (max(valpha_temp1)>vMaxQ(1) || max(
         valpha_temp2)>vMaxQ(1) || max(valpha_temp3)>vMaxQ(1))
32       alpha_temp1=a1(j)+dx1(j).*d2;
```

```
33        alpha_temp2=a2(j)+dx2(j).*d2;
34        alpha_temp3=a3(j)+dx3(j).*d2;
35        valpha_temp1=dx1(j).*s2/delta_time(j);
36        valpha_temp2=dx2(j).*s2/delta_time(j);
37        valpha_temp3=dx3(j).*s2/delta_time(j);
38        aalpha_temp1=dx1(j).*acc2/delta_time(j)^2;
39        aalpha_temp2=dx2(j).*acc2/delta_time(j)^2;
40        aalpha_temp3=dx3(j).*acc2/delta_time(j)^2;
41      end
42        alpha_motlaw1=[alpha_motlaw1,alpha_temp1(1:end-1)];
43        alpha_motlaw2=[alpha_motlaw2,alpha_temp2(1:end-1)];
44        alpha_motlaw3=[alpha_motlaw3,alpha_temp3(1:end-1)];
45        valpha_motlaw1=[valpha_motlaw1,valpha_temp1(1:end-1)];
46        valpha_motlaw2=[valpha_motlaw2,valpha_temp2(1:end-1)];
47        valpha_motlaw3=[valpha_motlaw3,valpha_temp3(1:end-1)];
48        aalpha_motlaw1=[aalpha_motlaw1,aalpha_temp1(1:end-1)];
49        aalpha_motlaw2=[aalpha_motlaw2,aalpha_temp2(1:end-1)];
50        aalpha_motlaw3=[aalpha_motlaw3,aalpha_temp3(1:end-1)];
51        DT_motlaw=[DT_motlaw,DT_motlaw(end)+x(2:end)*delta_time(j)];
52        E_motlaw=[E_motlaw,EE_final{i}(j)*ones(1,length(x)-1)];
53
54        end
55        alpha_motlaw1_final{i}=[alpha_motlaw1,alpha_temp1(end)];
56        alpha_motlaw2_final{i}=[alpha_motlaw2,alpha_temp2(end)];
57        alpha_motlaw3_final{i}=[alpha_motlaw3,alpha_temp3(end)];
58        valpha_motlaw1_final{i}=[valpha_motlaw1,valpha_temp1(end)];
59        valpha_motlaw2_final{i}=[valpha_motlaw2,valpha_temp2(end)];
60        valpha_motlaw3_final{i}=[valpha_motlaw3,valpha_temp3(end)];
61        aalpha_motlaw1_final{i}=[aalpha_motlaw1,aalpha_temp1(end)];
62        aalpha_motlaw2_final{i}=[aalpha_motlaw2,aalpha_temp2(end)];
63        aalpha_motlaw3_final{i}=[aalpha_motlaw3,aalpha_temp3(end)];
64        DT_motlaw_lay{i}=[DT_motlaw(1:end)];
65        E_motlaw_lay{i}=[E_motlaw,1];
66      end
```

B.4.6 save_menu

Listing B.21: save_menu

```
1  opt2=menu('save','yes','no');
2  if opt2==1
3        close all
4        P_flag=[alpha_motlaw1_final{LAYER}-alpha_motlaw1_final{LAYER}(1);
             alpha_motlaw2_final{LAYER}-alpha_motlaw2_final{LAYER}(1);
             alpha_motlaw3_final{LAYER}-alpha_motlaw3_final{LAYER}(1)];
5  opt=menu('cam .csv save option?','master_cam-disp','vel-disp(new way)');
6  if opt==1
7        save2melsoft1(DT_motlaw_lay{LAYER},P_flag,title1,E_motlaw_lay{LAYER})
8  else
9        opt3=menu('reduced or not?','full mot. law','f.m.l. diff','only 1
             every dL');
10       if opt3==1
```

```
11        new_cam
12        save2melsoft3(DX_cam_V,P_flag,strcat(title1,'_new_cam'))
13        disp('')
14        disp('make sure to add/eliminate F18_mit *1000 factor on vel')
15        elseif opt3==2
16           new_cam2
17           save2melsoft3(DX_cam_V,P_flag2,strcat(title1,'_new_cam_diff'))
18           disp('')
19           disp('make sure to add/eliminate F18_mit *1000 factor on vel')
20        else
21        new_cam1
22        save2melsoft3(DX_cam_V,P_flag1,strcat(title1,'_new_cam_red'))
23        disp('')
24        disp('make sure to add/eliminate F18_mit *1000 factor on vel')
25        end
26  end
27  end
```

B.4.7 new_cam1

Listing B.22: new_cam1

```
1  P_flag3=[alpha1{LAYER}-alpha1{LAYER}(1);alpha2{LAYER}-alpha2{LAYER}(1);
       alpha3{LAYER}-alpha3{LAYER}(1)];
2  t_i=diff(DT_motlaw)/60; %[min]
3  P_max=(max(abs(diff(P_flag3,[],2))))*1e3; %[mm]
4  v_i=[];
5  Dt=[];t1=[];t2=[];t3=[];s1=[];s2=[];s3=[];dt=[];ds=[];
6  v_i=P_max./t_i;
7  TT=0;
8  t1=v_i/a_max;
9  t2=t1/2;
10 dt=t_i-t1-t2;
11 s1=0.5*a_max*t1.^2;
12 s2=0.5*2*a_max*t2.^2;
13 ds=P_max-s1-s2;
14 t3=ds./v_i;
15 Dt=t1+t2+t3-t_i;
16 tr=t1+t2+t3;
17
18 DT_med=sum(Dt)*60
19
20 v_ext_new=sum(l_seg{LAYER})/sum(tr)/60;
21
22 if DT_med>0.01
23 opt=menu('what vel','v_med<v_ext','v_med=v_ext');
24 if opt==1
25 v_i=[0,v_i];
26 DX_cam_V=cumsum(v_i);
27 else
28 deter=(2*a_max*t_i/3).^2-4*P_max*a_max/3;
29      djk=find(deter>0);
```

```
30      djk1=find(deter<0);
31      if length(djk1)>1
32          disp('deter<0')
33      end
34      v_ii=zeros(1,length(v_i));
35
36   v_ii(djk)=2*a_max*t_i(djk)/3-sqrt(deter(djk));
37   v_ii(djk1)=sqrt(4*a_max*P_max(djk1)/3);
38
39  t1=v_ii/a_max;
40  t2=v_ii/(2*a_max);
41  dt=t_i-t1-t2;
42  s1=0.5*v_ii.^2/a_max;
43  s2=0.5*v_ii.^2/(2*a_max);
44  ds=P_max-s1-s2;
45  t3=ds./v_ii;
46  Dt=t1+t2+t3-t_i;
47  DT_aum=sum(Dt)*60
48  tr=t1+t2+t3;
49  v_ext_new=sum(l_seg{LAYER})/sum(tr)/60;
50  v_ii=[0,v_ii];
51  v_i=[0,v_i];
52  figure()
53  plot(v_i)
54  hold on
55  plot(v_ii)
56  DX_cam_V=cumsum(v_ii);
57  title1=[title1,'_Vm_eq_Vext'];
58  end
59  else
60  v_i=[0,v_i];
61  DX_cam_V=cumsum(v_i);
62  end
63
64  P_flag1=[diff(P_flag3(1,:));diff(P_flag3(2,:));diff(P_flag3(3,:))];
65  P_flag1=[zeros(3,1),P_flag1];
66  if length(find(DX_cam_V*100>2147483647))>0
67      disp('X-camma master larger than upper value:')
68      disp('reduce factor or split up the cam into more parts')
69  end
```

B.4.8 extruder_vel (velocity of the extruder punch)

Listing B.23: extruder_vel

```
1  V_extruder=((2*r)/d_p)^2*v_ext; %[m/s]
2  V_extruder=V_extruder*60*1e3 %[mm/min]
```

B.4.9 G_code_anaimation1

Listing B.24: G_code_anaimation1

```
1  function[]=G_code_anaimation1(DT_motlaw_lay,E_motlaw_lay,
```

```
    alpha_motlaw1_final,alpha_motlaw2_final,alpha_motlaw3_final,LAYER,l,s,
    Rb,zEstrusore,Z)
opt=menu('do you wanna plot','yes(it wil take alot [ctrl+c to exit]','no')
    ;
if opt==1
figure()
j=1;
tc_x=[];tc_y=[];tc_z=[];
close all
for i=1:length(DT_motlaw_lay{LAYER});
flag_e(j)=E_motlaw_lay{LAYER}(i);
alpha_1(j)=alpha_motlaw1_final{LAYER}(i);
alpha_2(j)=alpha_motlaw2_final{LAYER}(i);
alpha_3(j)=alpha_motlaw3_final{LAYER}(i);
q=[alpha_1(j);alpha_2(j);alpha_3(j)];
zoom=0;
[tc_x(j),tc_y(j),tc_z(j)]=plotLD(q,l,s,Rb,zEstrusore,zoom);
subplot(1,2,2)
plot3(tc_x,tc_y,tc_z,'b');
if flag_e(j)==1
hold on
plot3(tc_x(j),tc_y(j),tc_z(j),'r*');
end
drawnow
j=j+1;
   end
 end
```

B.4.10 d_dep_check

Listing B.25: d_dep_check

```
for i=1:length(DT_motlaw_lay{LAYER})
q=[alpha_motlaw1_final{LAYER}(i);alpha_motlaw2_final{LAYER}(i);
    alpha_motlaw3_final{LAYER}(i)];
p(:,i)=linear_direct_kinematics(q,l,s,Rb);
qp=[valpha_motlaw1_final{LAYER}(i);valpha_motlaw2_final{LAYER}(i);
    valpha_motlaw3_final{LAYER}(i)];
inv_J=linear_inverse_Jacobian(p(:,i),q,s,Rb);
new_v_plat(:,i)=((inv_J)^-1)*qp;
end
sp_abs=sqrt(new_v_plat(1,:).^2+new_v_plat(2,:).^2);
% temp=find(sp_abs==0);
% sp_abs(temp)=0.1;
d_dep=sqrt(v_ext*(2*r)^2./sp_abs);
figure()
plot(d_dep)
d_dep_mean=mean(d_dep)
d_dep_cov=cov(d_dep)
d_dep_max=max(d_dep)
d_dep_min=min(d_dep)
dep_error=[d_dep_max-2*r,2*r-d_dep_min]
```

B.4.11 line_close

Listing B.26: line_close

```
X=[];Y=[];E=[];
for i=1:N
X1=[];X2=[];Y1=[];Y2=[];E1=[];
if i==1
xoi=x0;yoi=y0;
temp_lc=0:lmax:lmax;
X1=xoi+temp_lc;
Y1=yoi*ones(1,length(temp_lc));
E1=[0,ones(1,length(temp_lc)-1)];
% E1=[ones(1,length(temp_lc))];
X2=X1(end)-temp_lc;
Y2=(Y1(end)+dd(i))*ones(1,length(temp_lc));
else
xoi=x0;
yoi=Y(end)+ll+dd(i);
temp_lc=0:lmax:lmax;
X1=xoi+temp_lc;
Y1=yoi*ones(1,length(temp_lc));
E1=[0,ones(1,length(temp_lc)-1)];
% E1=[ones(1,length(temp_lc))];
X2=X1(end)-temp_lc;
Y2=(Y1(end)+dd(i))*ones(1,length(temp_lc));
end
X=[X,X1,X2];Y=[Y,Y1,Y2];E=[E,E1,E1];     % E(1)=0;
end
figure()
plot(X,Y)
hold on
for i=1:length(X)
if E(i)==1
    plot(X(i),Y(i),'r*')
end
end
V=[E;E;E];
Z=z*ones(length(X));
```

Bibliography

[1] A. Dolenc, An Overview Of Rapid Prototyping Technologies In Manufacturing,1994, Industrial automated institute of Helsinki university of technology

[2] Ian Gibson, David W Rosen, Brent Stucker et al. Additive manufacturing technologies. Springer, 2015.

[3] E. Fiore, H. Giberti e L. Sbaglia. "Dimensional synthesis of a 5-DOF parallel kinematic manipulator for a 3d printer". In: Research and Education in Mechatronics (REM), 2015 16th International Conference on. Nov. 2015, pp. 41-48.

[4] Han W. et al. "Tool Path-Based Deposition Planning in Fused Deposition Processes". In: Journal of Manufacturing Science and Engineering 124.2 (2002), pp. 462-472.

[5] Jin et al. "Optimization of tool-path generation for material extrusion-based additive manufacturing technology". In: Additive Manufacturing 1 (2014), pp. 32-47.

[6] S.-H. Suh, S.-K. Kang, D.-H. Chung, I. Stroud, Theory and Design of CNC Systems, Springer, 2008.

[7] Giovanni Legnani, Irene Fassi e Antonio Visioli. Robotica industriale. CEA, 2003.

[8] Luca Sbaglia. "Ottimizzazione e Progettazione di un robot 5 gdl per stampa 3D". Tesi di laurea. Politecnico di Milano, 2014/2015.

[9] Marco Lotterio. "Progetto e Sviluppo di una Testa di Iniezione MIM applicata ad una Stampante 3D". Tesi di laurea. Politecnico di Milano, 2014/2015.

[10] Marco Parabiaghi. "Pianificazione di traiettorie di deposizione per una stampante 3D innovativa a 3 gdl a portata di estrusione costante". Tesi di laurea. Politecnico di Milano, 2014/2015.

[11] Richard G. Budynas, J. Keith Nisbett. Shigley's mechanical engineering design. McGraw-Hill, 2011

[12] Mitsubishi Electric industrial automation, cur. MELSEC A/Q series: Programmable Logic Controllers, Programming Manual. 2004.

[13] Mitsubishi Electric industrial automation, cur. MELSEC System Q: Programmable Logic Controllers User's Manual Hardware Description. 2011.

[14] Mitsubishi Electric industrial automation, cur. MELSEC System Q: PLC Manuale del Principiante. 2009.

[15] Mitsubishi Electric motion controllers, cur. MOTION CONTROLLER Qseries User's Manual (Q173D(S)CPU/Q172(S)CPU). 2011.

[16] Mitsubishi Electric motion controllers, cur. MOTION CONTROLLER Qseries Progrmming Manual (COMMON) (Q173D(S)CPU/Q172D(S)CPU). 2012.

[17] Mitsubishi Electric motion controllers, cur. MOTION CONTROLLER Qseries SV13/SV22 Programming Manual (Motion SFC) (Q173D(S)CPU/Q172D(S)CPU). 2013.

[18] Mitsubishi Electric, cur. General-Purpose AC Servo MELSERVO (HG-MR/HG-KR/HG-SR/HG-JR/HG-RR/HG-UR) Servo motor instruction manual (Vol. 3). 2012.

[19] Mitsubishi Electric, cur. General-Purpose AC Servo MELSERVO-J4 SSCNET III/H Interface AC Servo (MR-J4-B(-RJ)/MR-J4-B4(-RJ)) Servo amplifier Instruction Manual. 2012.

[20] P.L. Magnani, G. Ruggieri. Meccanismi per Macchine Automatiche, ed. UTET, Torino, Italia, 1986.

[21] Peter Zamiska PM et al. "LITERATURE REVIEW 3D PRINTER". In: (2013).

[22] Enrique Canessa, Carlo Fonda e Marco Zennaro. Low-cost 3D Printing for Science, Education & Sustainable Development. ICTP-The Abdus Salam International Centre for Theoretical Physics, 2013.