POLITECNICO DI TORINO

M.Sc. in Mechatronic Engineering

Master Thesis

Design and implementation of a planar robot for food cutting

Supervisors
Prof. Alessandro Rizzo

Ing. Emanuele Giardi

Candidate
Alberto Pianigiani

Summary

This thesis has been developed in collaboration with MAINIT S.r.l , a company located in Sinalunga (SI) that has been for years in the field of industrial automation.

The aim of this thesis is to develop an unconventional cartesian plotter 2D for food cutting (generally in every type of 2D cutting) using high pressurized water. In particular we have developed an innovative placement of motor that is capable to fix the motor at the machine chassis and avoid its linear movement. The tray is moved using a special configuration of belt that transmits the motion. The mechanical part takes in to consideration the constraint analyzed, in order to develop a machine that suits the FDA, HACCP, and current European regulations, in the field of food manipulation. Also, we have developed the electrical and software part of the machine inspiring to the conventional CNC machine. The final result will be a functioning 2D plotter specialized in the food production field, with ad-hoc Human-Machine Interface. In the thesis is not consider the design of the water pump, but only the robot for the movement of the nozzle.

The thesis is structured is this way

- The first chapter is dedicated to some useful preliminary notion. Without going into details, in this chapter is show the water jet technology and, the G-Code language.
- The second chapter deals with the analysis requirement of the mechanical part, in particular the solution adopted to build, the machine and which are the materials required in the food industry field. Then, the components are selected according to our constraints that are carried out by the requirement analysis.
- The third chapter concerns the electronic hardware that is essential to control the motors. First of all, we have to create an analysis of the requirements and then the components are chosen.
- The fourth chapter is entirely dedicated to developing the HMI and the control software of the motors. This phase concerns an analysis of which tool a CNC machine has and then shows how is made the implementation of the software.
- The fifth chapter is about the electrical wiring diagram used to build the electrical panel.
- The sixth chapter shows a scale prototype built. Then, some pattern used to cut food are performed, showing the machine on work.
- The seventh chapter concerns the conclusion of this project, the problems that came up, and what we could implement to improve some aspects in the future.

Contents

List of figures

List of tables

1 Introduction

1.1 Water Jet technology

A water jet cut is an industrial tool that is able to cut a large variety of materials. It works pressurizing the water at high pressure using a pump and release the jet against the target material to be cut.
So, we will obtain the cut slowly moving the jet on the target material that produces a continuous abrasion between the material and the jet.
The first advantage of using water jet technology instead of other types of cutting is that we can cut the chosen material without interfering on its internal structure. The second benefit is that we can use small nozzles (about 0.18mm) in order to reduce the waste material
The first use of high-pressure water was dated back in 1800 where was used in the field of mineral extractions. Then the industrial application starts at the beginning of the 1930, where the jet could only cut soft materials as paper. The water jet technology evolves in 1960, when the progress done in develop more powerful pump, making the jet hypersonic and adding abrasive material to the fluid, such as sand, glass, or ceramic particles, increase even more the cutting power. Nowadays the water jet technology is used to cut various material, metal, stone, wood, plastic etc.

1.2 Advantages and disadvantages

The use of the water jet technology has its advantages and disadvantages, in particular the advantages are:

- The water jet technology is considered an eco-friendly resource since the cutter does not create hazardous waste and scrap material is easy to recycle. Furthermore, using a closed water loop, provided with proper filter, the machine consumes very small amount of water.
- Due to the high pressure jet the technology cut almost every type of material, metal glass etc.
- The technology makes it possible not to heat the cutting area
- The cut is clean, on the grounds that the cut material comes into contact only with uncontaminated water
- The cut is very precise and can be achieve very intricate cut pattern

Disadvantages:

- The water jet cut takes more time than the traditional cutter.
- Buying a water jet machine is very expansive and would affect the starting costs
- The machine consumes more power than the traditional cutter.

The water jet technology provides advantages and disadvantages that do not state if it is generally convenient. So, we have to evaluate our requirements in order to check if the technologies fit our application. In our case with the increase grow of health demand in the food industry, cleaner machines are request. Nowadays most of the food cutter use blade that does not guarantee a satisfactory clean cut. In fact, the blade comes into contact with the food and bacteria and germs could be transmitted from piece to piece of food. Furthermore, analyzing physically how the blade performs the cut, we can see a crushing force on the food, smashing the cutting edge, of the food making it less attractive. So, the request of a clean cut and the ability to perform an attractive cut directly using a water jet technology is mandatory in our application [1].

1.3 G-Code

The state of art of movement control of the machine is the same of 2D CNC machine. In fact, if in the end effector is a nozzle (as water jet machine) or a drill or whatever tool, the control of the movement remains the same. The motor generally used is the stepper, that provide a high precision of positioning and a reliable repetition of movement. The path that the end effector must execute is written in G-code, that is the acronym of "geometrical code". This type of code is the most widely used in CNC machine with computer aided. It provides a written description of the main geometric figure (straight line and circle). The more complicated are obtained dividing it in much simply figure. For example, a curve line, that cannot be associated to a circle is divided in a high number of small lines. Taking care of the number of little lines is much than the machine resolution the result on the end effector will be the same. The G-code instruction refers to a CNC controller the movement direction and speed, and which path has to follow. Without going into detail, an example of G-code command is:

G00 X10 Y4 F40

In this command, **G00** means rapid positioning in X=10 and Y=4 (**X10 Y4**) with a feed velocity of 40 units (**F40**). Units can be mm/sec cm/sec or inch/sec and is set as parameter in the machine. There are more than hundred types of first parameter (**G00**) but we are interested only in:

- **G00** rapid positioning, that moves to the specified point without cutting.
- **G01** linear interpolation, that concerns a straight movement by interpolation of the point that has been set before and the point next to the **G01** command.
- **G02** circular interpolation clockwise, that is made from a high number of linear interpolations. Next to **G02** there is the final position. Next is present the letter **R** to set the radius or **IJK** to set the vector from the start point to the arc center (the offset from the axes). The arc will start from the point that is in the command before and finish in the point next to **G02**. So, for instance, the point next to the command must be the same of the previous command to draw a full circle.

2 Mechanic

The law number 1935/2004 of the European union [2] establishes which material can be used in the realization of a machine designed to be employed in food industry. In particular, it states that the metal that touches or is near the food must not affect the food property, must not contaminate it or extend the expiry date. Furthermore, the mechanic's assembly of the machine must not create cavities where dirty can enter. In our case, the cleaning aspect is extremely important, and the machine must be clean in depth using pressurizing water periodically, in order to prevent germs and bacteria from settling. For this reason, the motors and in general the whole machine must satisfy a water grade protection to avoid electrical o mechanical fault. In terms of performance and cleaning, an assessment of the requirements has been made in the design of the mechanic part related to the trays, belt, pulleys, motors, and chassis [3]. Finally, all components are selected from the commercial offer, respecting the constraint of current regulations.

2.1 Requirement analysis

Having regard to the working machine environment, the actual laws on treatment of food, the structural constraints of the chassis and the velocity and acceleration of the robot end effector in this section is made an analysis of the requirement in terms of the materials and performance of components. Furthermore, the placement of the motor is treated, computing the motor-end frame transformation displacement.

2.1.1 Work environment

Using a high-pressure pump, the fluid that compose the water jet reaches about 2000 bar of pressure. The contact between the water jet and the food produces water splashes on the surrounding and even some pieces of food can attach to the machine. The organic material dust and the water splashes, that contaminates the area, must be taken into consideration for every choice of components. For example, the motor and the limit switch must be insulated and have a satisfy IP grade, the mechanical design.

2.1.2 Cleanability

The cleanability is crucial for our application since the machine must be clean and germless in order to not contaminate the food. Furthermore, some residual food can attach on the machine's chassis in working condition and for this reason the operation of cleaning is particularly important to guarantee a clean environment [4]. The washing process is performed by using a pressure washer to ensure a depth clean of the machine's surface and due to the washing method, all electronics must be satisfied an IP grade to ensure reliability and avoid faults. The motor and the limit switch must have a satisfy IP grade and the motor driver, the PC, the motor controller, and the HMI must be placed in a special locker outside the working area of the machine. The locker must be insulated from the environment and the pressure inside must be higher than the external, in the way to pressurize it and avoid the entrance of non-desired corps (water of food).

2.1.3 Motor

The movement of the tray is delegated to the motor. The motors are electromechanical devices that convert the electrical power into mechanical power. Nowadays, there are many types of motor that can be generally divided into direct current and alternate current motors.

In our application we need a motor type that can perform precise and repetitive displacements and have a good starting torque. So. our choice is the stepper motor. This choice is made even because the stepper motor is widely used for years in CNC machines and since our machine belongs to this machine type, it fits our requirement of precise displacements and guarantees reliability. The stepper motor divides a full rotation into an exact number of steps. To every step, corresponds the angular displacement gives by 360 degrees over the number of steps in order to perform a full rotation. For instance, if we need 180 steps to perform a full rotation, then one step corresponds to an angular displacement of 2 degree. Analyzing the internal design of a stepper motor, we can find a gear shaped rotor, with the number of teeth equal to the needed steps to perform a full rotation. Taking into consideration the example made before, we have a rotor with 180 teeth. The stator is arranged around the rotor, and is composed by multiple toothed electromagnets, for example 4 electromagnet with 3 teeth each. To make the motor shaft (rotor) turn we give power to the first electromagnet which magnetically attracts the rotor's teeth. When the rotor's teeth are aligned to the first electromagnet, they are slightly offset from the next electromagnet. This means that when the next electromagnet is turned on and the first is turned off, the rotor rotates slightly to align with the next one. From there, the process is repeated obtaining the motor rotation. The step is defined as the process of align the rotor tooth to the stator. Without going into details, the torque curve of a stepper motor is in function of their supply voltage. In fact, applying a higher voltage, the magnetic attraction between stator and rotor teeth increases generating more momentum and in the end more torque. So, the choice of the motor must take into consideration the needed power supply.

The motion is transmitted attaching a pulley on each motor's shaft; then, the pulleys move the belt that is attached on the tray. This chain of movement introduces a gear ratio, where an angular displacement of each motor corresponds to a linear displacement of the tray. We must consider the gear ratio to obtain the correct displacement from the G-code to the willed displacement. Furthermore, the stepper must satisfy a proper IP grade protection. The IP grade indicates the level of protection of a device. Generally, it is composed by 2 digits (for instance IP69), where the first digit gives the protection of the enclosure against the access of solid object and the second digit indicates the level of protection against ingress of water. In our case since the presence of water we need at least an IP protection of 65. This grade ensures a total not ingress of dust and a resistance to the water jet. Having choose the motor type as stepper motor and the need IP protection, we still must choose the specific component with the correct torque. The required value of torque is computed in 2.2.5 chapter. Finally, according to the previous analysis we must choose the pulleys, in particular the pulleys radius.

2.1.4 Motor placement and tray displacement

The design of a conventional cartesian robot is made by 2 prismatic joints in case of 2D move. The general construction presupposes the placement of the motor on each arm, where the rotational movement of the motor is converted into linear movement by using the pulleys. As we can see in the Figure 1 the robot is made up two groups of belt pulleys. The first, that is the so-called “primary”, is composed by the motor U and pulley number 1, it is fixed to the machine’s chassis and moves the nozzle in Y direction. The second, that is the so-called “secondary”, is composed by the motor V and pulley number 2 and it is attached rigidly to the primary group on the belt and move the nozzle in X direction. The robot achieves every point in the 2D plane in this way: acting on the motor U, the secondary group is moved in Y direction and operating on the motor V, the nozzle is moved in the X direction. This simply solution conducts to a simple transformation of the motor position nozzle position, that is:

- $d_y = k \cdot \theta_U$
- $d_x = h \cdot \theta_V$

Where k and h are the constant carried out by the gear box (if present) and the pulley radius

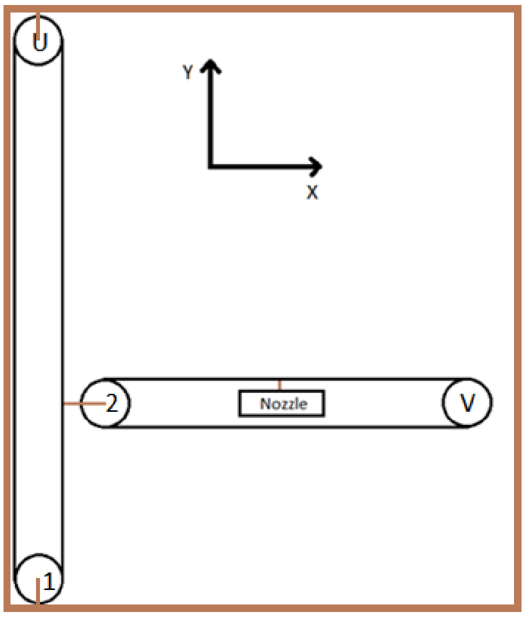


Figure 1: Conventional cartesian plotter

This widely tested and reliable configuration does not correctly fit our cleaning requirement. In fact, the control cable of the motor V on the secondary tray are carried with the movement of tray itself. Also, the cables are near the water jet, where the food cut dust can splash on the cable becoming a potential hazard for germs and bacteria. Furthermore, the clean operation will become more difficult since the cable must be clean in tidy way.

This problem is overcame placing the two motors outside on the fix chassis and using a different belt configuration, transmits the movement in order to move the end effector. As we can appreciate in the Figure 2 the motors are positioned outside and the end effector movement is done with a different combination of motor rotation.

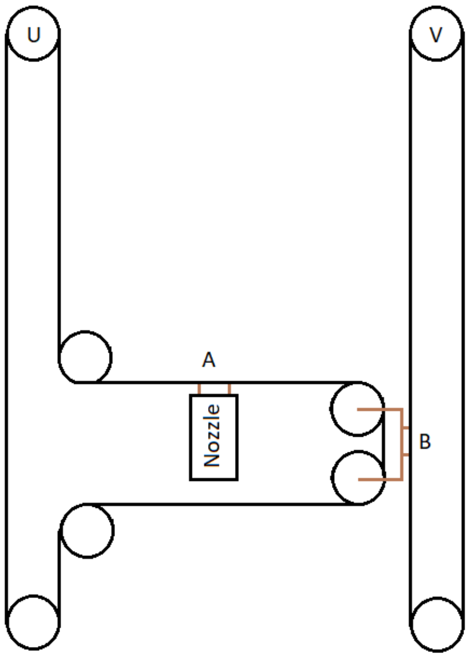


Figure 2: Unconventional cartesian plotter

As we can see this configuration overcame the cleanability problem, but as disadvantage has a greater number of pulleys than the traditional configuration. In particular, we have 8 pulleys versus 4. Furthermore, the motor-end effector transformation is not equal to the traditional one. Without going into details (is computed later), we can state that to perform a movement in the X direction we must rotate both motors in different direction (U and V) and to perform a movement in Y direction we must rotate only the U motor. Considering the mechanicals complications in exchange of a cleaner environment, we use this configuration to fulfill the cleanability problem.

The super position theorem is used to compute the end effector transformation. This theorem states that: for all linear systems, the net response caused by two or more stimuli is the sum of the responses that would have been caused by each stimulus individually. In our case, the general movement of the end effector can be decomposed in 2 orthogonal simply movement one on the X axis and the other in the Y axis. The other movements can be achieved using the combination of the simple movements. So, the transformation of our unconventional machine is obtained seeking the transformation of the movement only in the X axis and the movement only in the Y axis.

As we can see in the Figure 3 the reference system is posed in this manner: the X axis upward and the Y axis to left with the reference of the machine chassis. There is not a particular motivation, but only that the user is presupposed to be on the right side of the machine. Starting to determine the movement on the Y axis, we can see that is achieved acting an anti-clockwise rotation of motor U the block of motor V. In fact, in performing this operation, the motor U pulls the belt, and the motor U fixes the pose of the nozzle at the constant X value. Then, the nozzle's tray is pulled in Y direction. So, the simple movement on the positive Y axis is carried out by the anti-clockwise rotation:

- $\mathrm{d}_y = k \cdot \theta_U$

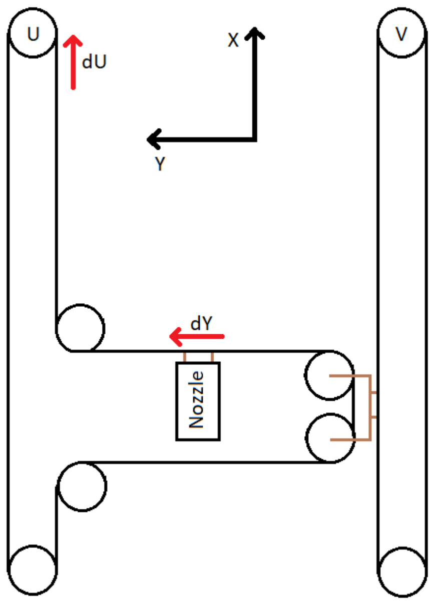


Figure 3: Determination of the transformation linear-motor movement on X axis

Considering the method used to find the transformation on the X axis, in the Figure 4 is shown how has been obtained. Starting to move the motor V in clockwise direction we cause the movement of the tray upwards, due to the fixed connection between belt and tray. But, since the tray is also fixed to the motor's U belt if we do not rotate it, the nozzle starts to move according to an oblique pattern. Finally rotating in clockwise the motor V and also, rotating the U motor in anti-clockwise we cause the movement of the nozzle's tray in X direction. So, the positive X movement is carried out by:

- $d_x = k \cdot \theta_U + h \cdot \theta_U$

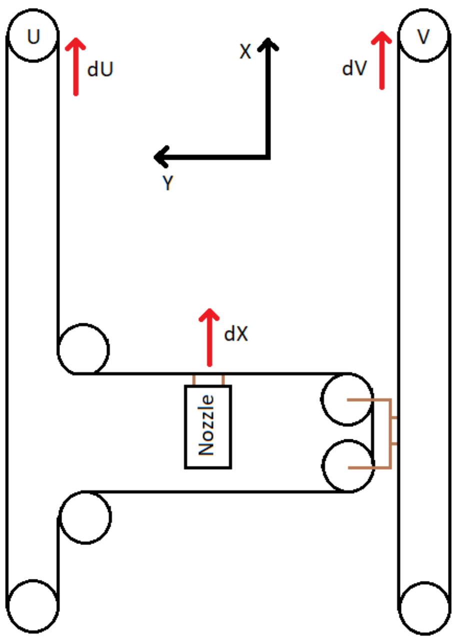


Figure 4 Determination of the transformation linear-motor movement on X axis

Finally, the superposition theorem gives us back the end effector-motor transformation:

- $d_y = k \cdot \theta_U$
- $d_x = k \cdot \theta_U + h \cdot \theta_V$

Applying basic math we compute the motor-end effector transformation, that is:

- $h \cdot \theta_V = d_x - d_y$
- $k \cdot \theta_U = d_y$

2.1.5 Double tray

To fulfill in the best manner the clean environment constraint, a double tray is used in order to collect the water jet residue from the cut. In fact, when the high pressurized water hit the target, causing the cut, if not find a proper binder, the water goes in the lower part of the machine contaminating the whole chassis. To deal with this problem we have developed a double tray configuration where, the upper tray manages the cutting nozzle and the lower tray follow the residue jet collecting it using a binder. Then, the binder is attached to a pipe for the correct disposal of the wastewater. The double tray is constructed by creating a specular tray. So, the nozzle's tray and the binder tray are equal in each part. To make stronger the structure the X movement of the two trays, they are link using a bracket. Then, the belt is attached on the bracket, and so, the movement of the belt causes the movement of both trays, as we can see in the Figure 5.

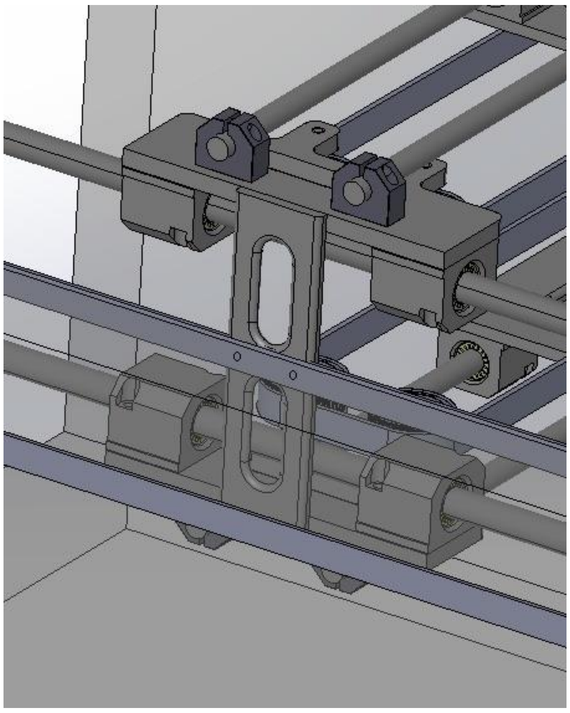


Figure 5: Close up of belt-tray coupling

Between the two trays there is a plane where the target is lying. Considering the movement on the Y axis of the trays, there are 2 belts, one for each tray since we cannot use a bracket to synchronize the motion, since in the middle of the double tray configuration lay the plane where is present the target. The view of the double tray configuration is shown in the Figure 6. For better understanding how the nozzle is bidden to the belt the target plane is not represented in the figure.

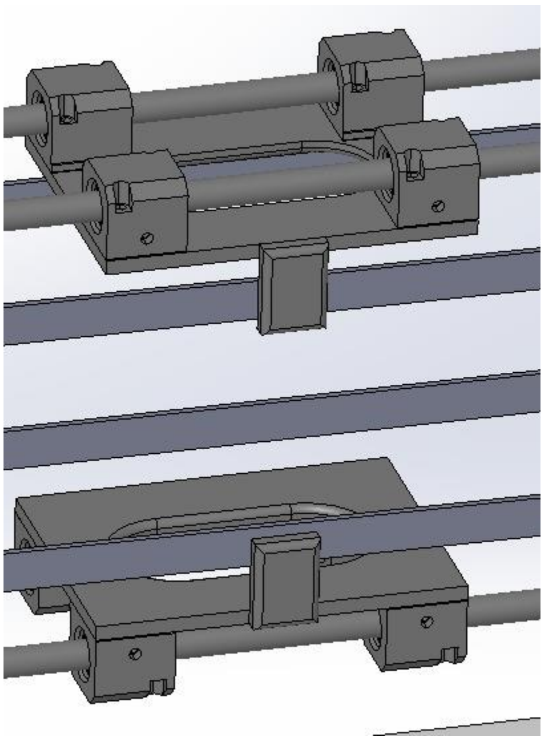


Figure 6: Close up of belt-tray coupling

In the Figure 7 is shown the front view of the machine and here we can appreciate the binding bracket of the nozzle tray to the belts. The choice of implementing a 2 trays configuration, will complicate the structure and increments its weight (more than double) making necessary more torque to obtain the willed performance. But, since the clean environment requirement cannot be negotiated to obtain a faster performance, we decided to develop a double tray configuration, giving more importance in having a clean environment.

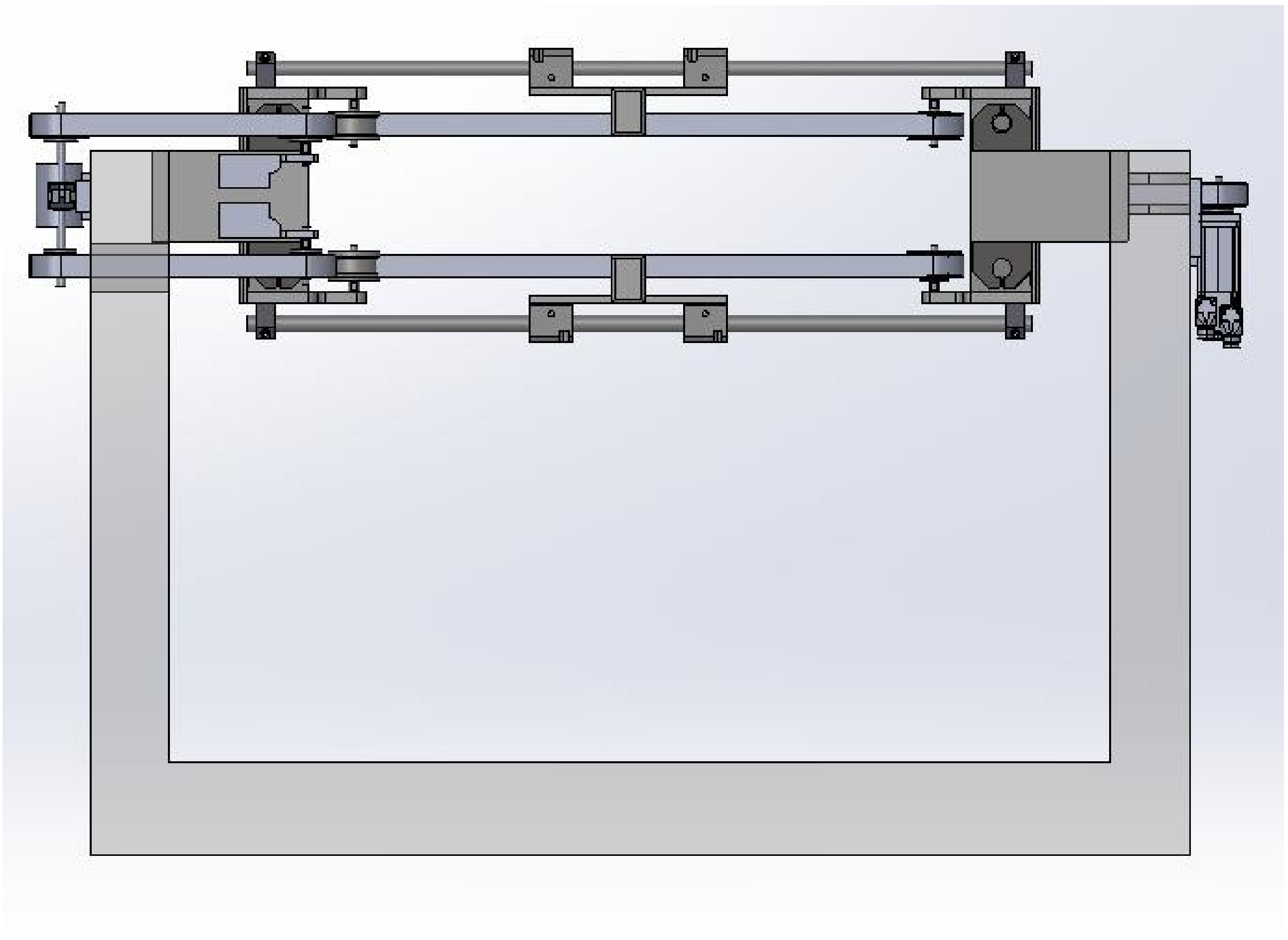


Figure 7: Front view of the trays

2.1.6 Dynamic structure analysis

Having decided the motor placement we can start to compute theoretically the dynamic and static force that are present in the structure of the machine. We choose to perform a theoretical analysis in order to extrapolate the needed formulas, that are used to compute the motor torque in function of the pulley and the tray's mass.

Firstly, we need to set the requirement in term of velocity and acceleration of the small tray and the big tray. We have taken as maximum acceleration of both trays the value of $0.5\frac{m}{s^2}$, and a value of cruise velocity of $0.5\frac{m}{s}$.

Using the Newton's second law we have obtained the torque needed to satisfy our maximum acceleration constraint:

$$\tau_i = a_{max} \cdot m \cdot r + n \cdot J_p \cdot \ddot{\theta}$$

Since we have the motor torque and the pulleys radius to set is necessary to choose a priori one and compute the other. In this project, we will select the pulleys radius and next we will compute the need motor torque to obtain the requested acceleration. This choice is justified even for the dimension of the pulley offer by the market. In fact, we cannot have a pulley with whatever radius we want but we must adapt to the dimension offers by the market. The trays weight will be obtain using the function 'measure' of the 3D cad software used in the develop of the mechanical part (Solidworks). The measure obtained is not the real one but give us an idea of what weight will have the tray. Furthermore, applying a security factor of 2 on the final torque result we make sure that we have fit the acceleration requirement. Finally, in the computation of the motor torque is considered the inertia of the pulley and the motor is considered.

2.2 Components selected

2.2.1 Linear guide and double tray

The market offers a wide range of prefabricated linear guide that are easy to use and install. But our constraints are different. In fact, most of them are rectangular shaped, that means that they can accumulate food screw and become a potential health hazard. Furthermore, the slide shaft is constructed with aluminum, a material that can be corroded by the bearing's friction or oxidated by the water and become source of contamination of the food. According to the cleanability constraints, the environment constraint and the actual law on the proper material we choose to construct our linear guide for the big and small tray, selecting the slide shaft, the bearings, and the shaft block preferably with INOX AISI 316.

2.2.1.1 Big tray

In this section there are represented the slide shaft, the slide bearings and the shaft block of the big tray linear guide. It measures 1300 mm since it contains the small tray.

2.2.1.1.1 Linear guide

We have selected a shaft with circular section of 25mm as linear guide with a length of 1300mm. According to our constraints, the material selected is the INOX AISI 316 steel. The material is suggested in wet environment since cannot be oxidated by the water. Also, the material provides a higher corrosion resistance to chemical substance and due to the high hardness, the material is resistant to the friction. The section of the shaft is shown in the Figure 8. The component selected is catalogue as R stainless steel shaft, EEWM, 1.4034 (420C).

Figure 8: Circular section shaft

2.2.1.1.2 Shaft block

Then, we have selected the element as shaft block: shaft mounting block WA 25. The shaft block is used to fix the sliding shaft to the machine's chassis. There are 3 holes on it for the screws and one hole for fix the shaft. Two of the screw holes are used to fix the piece on the machine's chassis and the other is used to tie the insert slide shaft in the proper hole. Since there any friction is not applied on the surface of it, we are less constrained in the choice. As shaft block, we have chosen a standard type made in aluminum, with a hole of 25 mm of diameter. Furthermore, to best suit the cleanability constrain we can insert 2 plugs in the screw hole. The chosen component is shown in Figure 9.

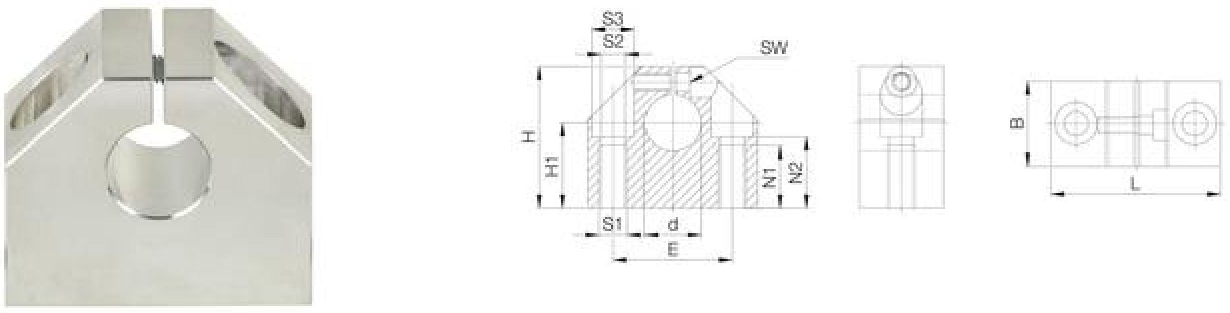


Figure 9 Shaft block

2.2.1.1.3 Slide bearings

As slide bearings we select the Bearing R pillow block RGA-04 25 mm. As we can see in Figure 10 the bearing is composed by a solid polymer is used as slide element, in order to reduce friction with the metal shaft. The housing material is aluminum and there are 2 hole to fix the element to the tray.

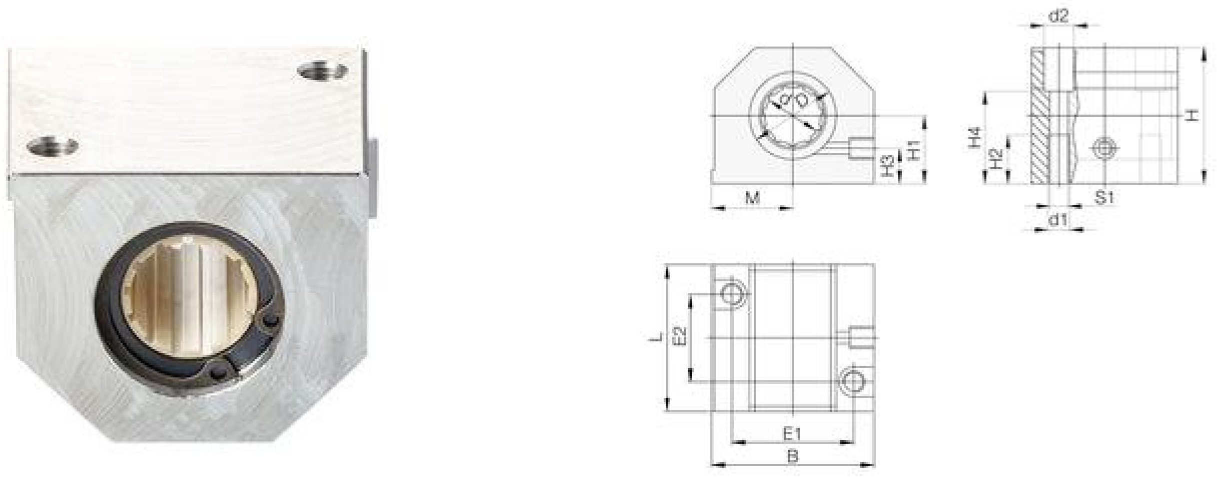


Figure 10: Slide bearings

2.2.1.1.4 Final assembly

Finally, coupling the selected component we obtain the complete linear guide shown in Figure 11

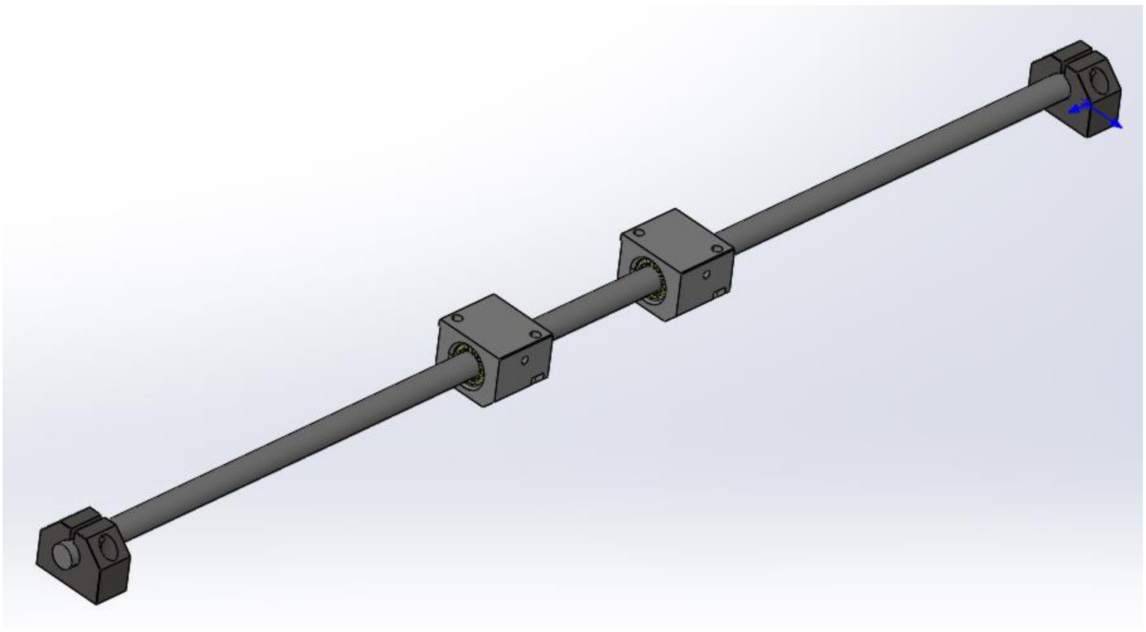


Figure 11: Final assembly of the linear guide

2.2.1.2 Small tray

The elements selected for the small tray are the same of the big tray with same characteristic but with smaller dimension. Both linear guides are similar in every component and for this reason the image of the component are not shown.

2.2.1.2.1 Linear guide

As linear guide we select a circular shaft with a diameter of 20mm and a length of 1000 mm. The material is the same of previous shaft.

2.2.1.2.2 Shaft block

As shaft block, we have chosen the WA 20, that belongs to the same series of the big tray's shaft block.

2.2.1.2.3 Slide bearings

As slide bearings, we select the component RGA-04 20 mm, that has the same characteristic of the previous used.

2.2.2 Pulley assembly and belt

2.2.2.1 Pulley

In the machine there are three types of use of the pulleys. In one case, they are placed on the motor shaft to transmit the rotation and in this case the pulley is rigidly fixed on the shaft and rotate together with the rotor. In the second case the pulleys are used to keep the belt tight and are attached to the machine chassis or to the tray using a fixed shaft (see the general view of the machine chapter 2.3). In this case must have a free rotation. So, a set of balls bearings are necessary to reduce the friction between pulley and its shaft. Also, since the pulley touches the belt in the toothed part, it must be toothed. In the third case the pully has the same role of the second one but touches the belt in the smooth part, so the pully must has not toothed but smoothed.

The toothed pulley that we have selected is made in aluminum. We have chosen the 35t054236 (code), 36T 5/42-6F (description) of the factory "Poggi trasmissioni meccaniche spa". As we can see in the Figure 12 the pulley has 42 teeth, and a medium diameter of 66mm. Also, as we can see the pulley has 2 rings around the teeth with a bigger diameter (71mm) and they are used to keep the belt inside the pulley's center in order to avoid the exiting of the belt. The pulley has an internal hole diameter of 8 mm.

Materiale	Codice	Descrizione	N° denti	Dp	De	Df	Dm	F	L	d	Peso kg
ALLUMINIO UNI 3571 TA 16	35T051036	36 T 5/10-6F	10	15,92	15,05	19,5	8	30	36	–	0,020
	35T051236	36 T 5/12-6F	12	19,10	18,25	23,0	11	30	36	–	0,030
	35T051436	36 T 5/14-6F	14	22,28	21,45	25,0	14	30	36	–	0,040
	35T051536	36 T 5/15-6F	15	23,87	23,05	28,0	16	30	36	6	0,040
	35T051636	36 T 5/16-6F	16	25,46	24,60	32,0	18	30	36	6	0,042
	35T051836	36 T 5/18-6F	18	28,65	27,80	32,0	20	30	36	6	0,060
	35T051936	36 T 5/19-6F	19	30,24	29,40	36,0	22	30	36	6	0,070
	35T052036	36 T 5/20-6F	20	31,83	31,00	36,0	23	30	36	6	0,080
	35T052236	36 T 5/22-6F	22	35,01	34,25	38,0	24	30	36	6	0,080
	35T052436	36 T 5/24-6F	24	38,20	37,40	42,0	26	30	36	8	0,110
	35T052536	36 T 5/25-6F	25	39,79	39,00	44,0	26	30	36	8	0,120
	35T052636	36 T 5/26-6F	26	41,38	40,60	44,0	26	30	36	8	0,120
	35T052736	36 T 5/27-6F	27	42,97	42,20	48,0	30	30	36	8	0,130
	35T052836	36 T 5/28-6F	28	44,56	43,75	48,0	32	30	36	8	0,130
	35T053036	36 T 5/30-6F	30	47,75	46,95	51,0	34	30	36	8	0,150
	35T053236	36 T 5/32-6F	32	50,93	50,10	54,0	38	30	36	8	0,180
	35T053636	36 T 5/36-6F	36	57,30	56,45	63,0	38	30	36	8	0,230
	35T054036	36 T 5/40-6F	40	63,66	62,85	66,0	40	30	36	8	0,280
	35T054236	36 T 5/42-6F	42	66,84	66,00	71,0	40	30	36	8	0,290
	35T054436	36 T 5/44-6	44	70,03	69,20	–	45	30	36	8	0,320
	35T054836	36 T 5/48-6	48	76,39	75,55	–	50	30	36	8	0,400
	35T056036	36 T 5/60-6	60	95,49	94,65	–	65	30	36	8	0,620

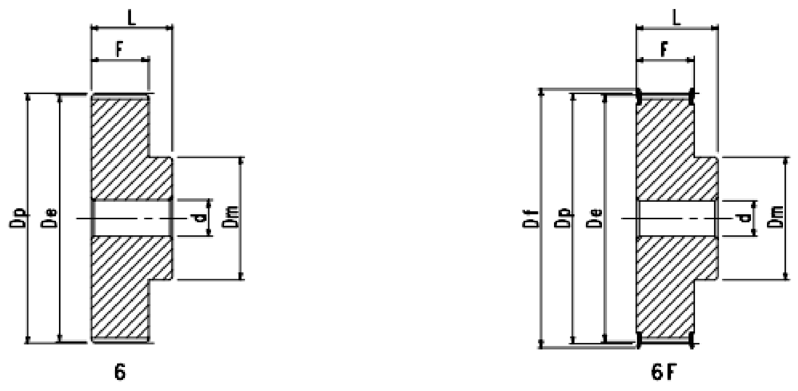


Figure 12: Pulley data

2.2.2.2 Bearings

The bearings must be chosen according to the work environment. Since the machine works prevalently in a wet environment, the bearing must provide a sealing barrier on the internal component (balls and cage) to the external factors. Also, they must satisfy the force constraint due to the acceleration of the tray. According to our requirements, we have selected the component W 6000-2RS1 produced by "RS" company. It has a basic dynamic load rating of 3.97 KN and a basic static load rating of 1.96 KN. It has an external diameter of 26mm and an internal diameter of 10mm. Also is sealed with a high resistant plastic ring.

2.2.2.3 Bearings pulley coupling

As is represented, the pulley and the bearing have a different diameter, 26mm for the bearing and 8 mm for the pulley, that do not allow the coupling. To obtain a perfect coupling we need to be machining the internal hole of the pulley until the desired measure of 26 mm to house the bearing is reached. We will not incur in structural problem of the pulley, because as we can see in the Figure 12 the pulley has a notch diameter of 40 mm so remains a 14 mm of material. Finally, the coupling is made by inserting 2 bearings for each pulley, one in the upper part and the other in the lower part. Then, a screw of 10 mm is screwed-in fixing the pulley on the machine or on the tray.

2.2.2.4 Belt

In the machine there are tree belts that we can divided into two types. One is moved by the motor U and the other by the motor V. Through Solidworks we have obtained the length of each belt. In particular, the belt related to the motor U is 5171.91 mm length and the belt related to the motor V is 2763.32 mm length. Considering the unusual belt length, we do not find a prefabricated one for our needed, so we have chosen to build our belt taking an unclosed belts with our lengths and closing it with the appropriate module. The belt has been chosen in order to consider the field of use, so we have adopted a food industry compatible type made of polyurethane produced by the "Optibelt" company.

2.2.3 Dynamic structure result

Having selected the component for the tray and its structure, we can compute the needed torque to fit the linear performance requirements of :

$v_{max} = 0.5\frac{m}{s} = v_x = v_y \qquad a_{max} = 0.5\frac{m}{s^2} = a_x = a_y$

The general formula analyzed before is:

$$\tau_i = a_{max} \cdot m_k \cdot r + n \cdot J_p \cdot \ddot{\theta}_i$$

where the indexes "i" and "k" indicates the motor (U or V) and the tray (big or small) respectively which on the formula is applied and n indicates the number of pulleys moved. Our data pulley diameter is 66.84 mm, that leads to a radius of 33,42 mm. The pulley weight = 0.29 Kg

$$\mathrm{d}_p = 66.84mm \quad r_p = 33{,}42mm \quad m_p = 0.29\ kg$$

Considering the angular acceleration and the linear acceleration relationship, we have obtained the maximum angular acceleration on the X axis. Since all the pulleys have the same radius dimension, the computed acceleration value will be the same on the Y axis:

$$\ddot{\theta}_U = \ddot{\theta}_V = \frac{a_x}{r} = \frac{a_y}{r} = 14.96\ \frac{rad}{s^2}$$

The pulley inertia is obtained approximating its shape to a disk with the same radius and weight. We will obtain a value of:

$$J_x = \frac{1}{2} \cdot m_p \cdot r^2 = 0.0001579\ Kg \cdot m^2$$

Through Solidworks we get the estimated weights of the big (m_b) and small (m_s) trays that are:

$$m_b = 43\ Kg \quad m_s = 13\ Kg$$

Entering the obtained value, we have computed the needed torque for fulfill the performance of the big tray:

$$\tau_V = a_{max} \cdot m_b \cdot r + 4 \cdot J_p \cdot \ddot{\theta}_V = 0.727\ N \cdot m$$

And for the small tray:

$$\tau_U = a_{max} \cdot m_s \cdot r + 16 \cdot J_p \cdot \ddot{\theta}_U = 0.255\ N \cdot m$$

2.2.4 Motor and encoder

In this section are shown the motor used. As analyzed in the requirement analysis we need a stepper motor with an IP protection of 65. The dynamic result indicates that the needed torque to fulfill our performance constraint is $0.727\ N \cdot m$ for the big tray and $0.255\ N \cdot m$ for the small tray. In order to unify the motor model, we have decided to use motors with equal characteristic for both trays. The choice of the model has been made according to the most stringent torque constraint, the needed torque on the big tray. Having the needed torque of $0.727\ N \cdot m$ and apply a security factor of 2 we must have a motor with $1.454\ N \cdot m$ of torque. Considering the offer on the market we select the motor P60SH86-TO0512P24C. It has a step resolution of 1.8 degree in a step (200 step in a full rotation), it guarantees an IP 65 protection, and has a torque of 1.8 $N \cdot m$ with a maximum current of 4.2 Arms. As we can appreciate in the Figure 13 the maximum torque ($1.8\ N \cdot m$) is constant in the range 0-50 rpm with a 24 Volt supply, where that tension is used in the actuation of the motors.

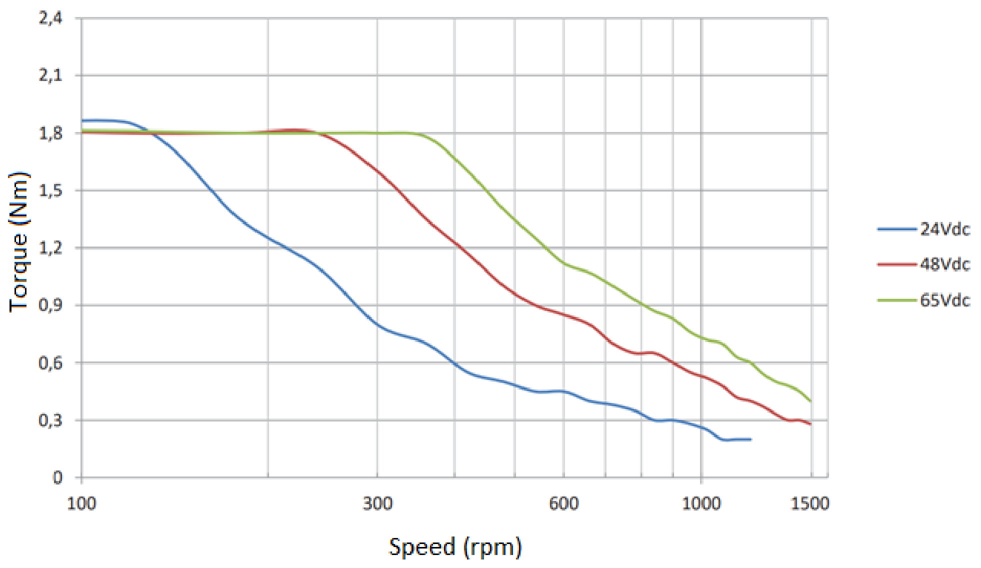


Figure 13: Torque-speed characteristic of the selected motor

The presented motor is provided by a built-in encoder with the same grade of IP protection of 65. It presents 512 pulses in a turn. The encoder has a supply voltage of 5V.

2.2.5 Limit switch

The limit switch is used as security device. It blocks the movement of the trays in the case of an impact with the machine chassis. An impact could be carried out by a general movement to an unreachable point outside the working area. Since the requirements states that the environment must be wet and during the clean process of the machine is used a pressure washer, we have decided to choose a limit switch with an IP grade of 69K, that offers the complete insulation to the fluid. The market offers a wide range of sensor types, that could be mechanical, magnetic, electric and photoelectric. To fulfill the needed IP grade protection, we have chosen a photoelectric type. The model selected is the E3FC-RN11 and is produced by "OMRON" company. As shown in the Figure 14 it has a NPN logic output (in other words it works with a positive logic output) and is pre-wired. It has the source and the receiving on the same element.

Sensor type	Sensing distance	Connection method	Model	
			NPN output	PNP output
Retro-reflective with MSR function *2	0.1 to 4 m with E39-R1S	pre-wired	E3FC-RN11 2M	E3FC-RP11 2M
		M12 connector	E3FC-RN21	E3FC-RP21

Figure 14: Limit switch sensor

Also, the sensor needs a reflector plate to bounce the generate light beam to the receiver. The reflecting element choose is the E39-R50 and is produced by the “OMRON” company shown in Figure 15. The maximum distance that the sensor reflector coupling can manage is 4 meters, so it fit our specification.

Sensing distance	Appearance	Model	Remarks
0.1 to 4 m		**E39-R50**	IP67, IP69K Ecolab tested plastic material

Figure 15: Limit switch reflector

To mount the sensor to the machine chassis we have chosen the proprietary brackets provided by the company and are shown in the Figure 16. The mounting bracket is used fix to the machine chassis and inserting the sensor in the hole with the photo element orientated to the reflector is ensure using the flush nut.

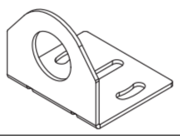
Sensor	Appearance	Model (Material)	Remarks
all types		**E39-L183** (SUS304)	Mounting bracket
		E39-EL16 (SUS316L)	M18 Flush mounting nut

Figure 16 Limit switch mounting bracket

2.2.6 Motor linear displacement transformation and encoder linear displacement.

In this section is represented the relationship between the motor angular displacement of the motors and the end effector linear displacement (nozzle). Also, there is the relationship between the number of pulse and the end effector linear displacement.

Considering the motor rotational displacement and the and effector linear displacement we know that to perform one full rotation we will need 200 steps (data of the motor). Then, we know that the pulley radius is 33.43 mm. So, every for each step of the motor correspond a linear displacement of:

$$Linear\ displacement = \frac{circumference\ of\ the\ pulley}{Number\ of\ step\ for\ one\ full\ rotation} = \frac{210}{200} = 1.05\ mm$$

Considering the number of pulse and the end effector linear displacement we know that the encoder produces 512 pulses every rotation completes by the motor. So, to compute the end effector displacement to make one encoder pulse we have to calculate:

$$Linear\ disp.for\ 1\ puls. = \frac{circumference\ of\ the\ pulley}{Number\ of\ pulse\ for\ one\ full\ rotation} = \frac{210}{512} = 0.41\ mm$$

Approximately to 1 motor's step correspond 2 encoder's pulses. The production by the encoder of 2 pulses for each motor step is necessary to have a better control of the movement. This happens because there can be no misunderstandings and in every case the movement of the motor is sensed by the encoder. Otherwise, if to one encoder pulse would correspond more motor step, we would not the knowledge of the motor motion.

Regarding to the motor-end effector displacement equations now we can compute the transformation constant. Considering the variables of the motors θ_U and θ_V as the number of steps for each one and the variable d_y and d_x as the linear displacement of the end effector in mm we compute the constant k and h as the linear displacement for one motor step. So, since we have the same pulley and the same motor configuration the constant k and h has the same value of 1.05 mm/step. So, the transformation equation become:

- $\mathrm{d}_y[mm] = 1.05 \cdot \theta_U\ [steps]$
- $\mathrm{d}_x\ [mm] = 1.05 \cdot \theta_U[steps] + 1.05 \cdot \theta_V[steps]$

Considering the maximum performance in terms of velocity of $0.5\frac{m}{s}$ in particular $500\frac{mm}{s}$ correspond to:

$$\dot{\theta}_{U\ or\ V} = \frac{v_{max}}{h\ or\ k} = 476.2\ \frac{steps}{s}$$

That lead to a step frequency of $476.2\ Hz$ and 142 rpm of the motor.

2.3 Machine view

The design of the machine has been made using the software Solidworks. It is a 3D CAD software that is used in the field of the mechanical drawing, in order to have a view of what will be the final result of the design. Many producers give the possibility to download the 3D CAD file of their component to use in the draw. In our case, after the selection of the component, we have downloaded its CAD file, then the component is coupled respecting the position constraint inside the drawing. Since the thesis aiming to the design of the linear guide, the motor and belt placement, we have drawn a symbolic chassis. As we can see in the Figure 17 the chassis is simply a U-shaped block. The drawing is 'not totally defined', that means that some component can move, and in particular the upper and lower trays can be shift according to the direction of the linear guide. This feature gives us the idea of which movement the machine can perform, or if the machine component going to crash with the others. In the Figure 17, Figure 18, and Figure19 is shown the final draw done.

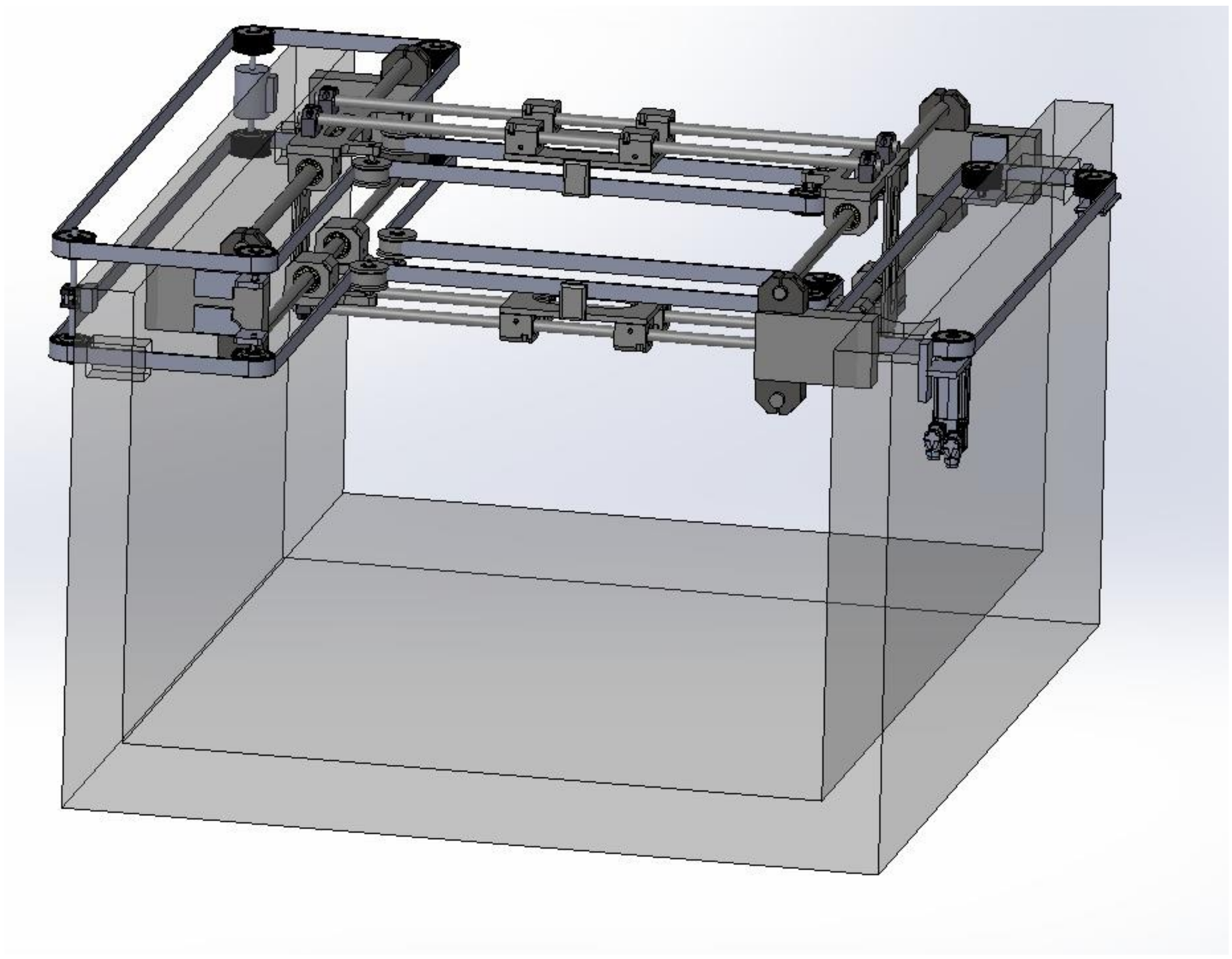


Figure 17: Machine front view

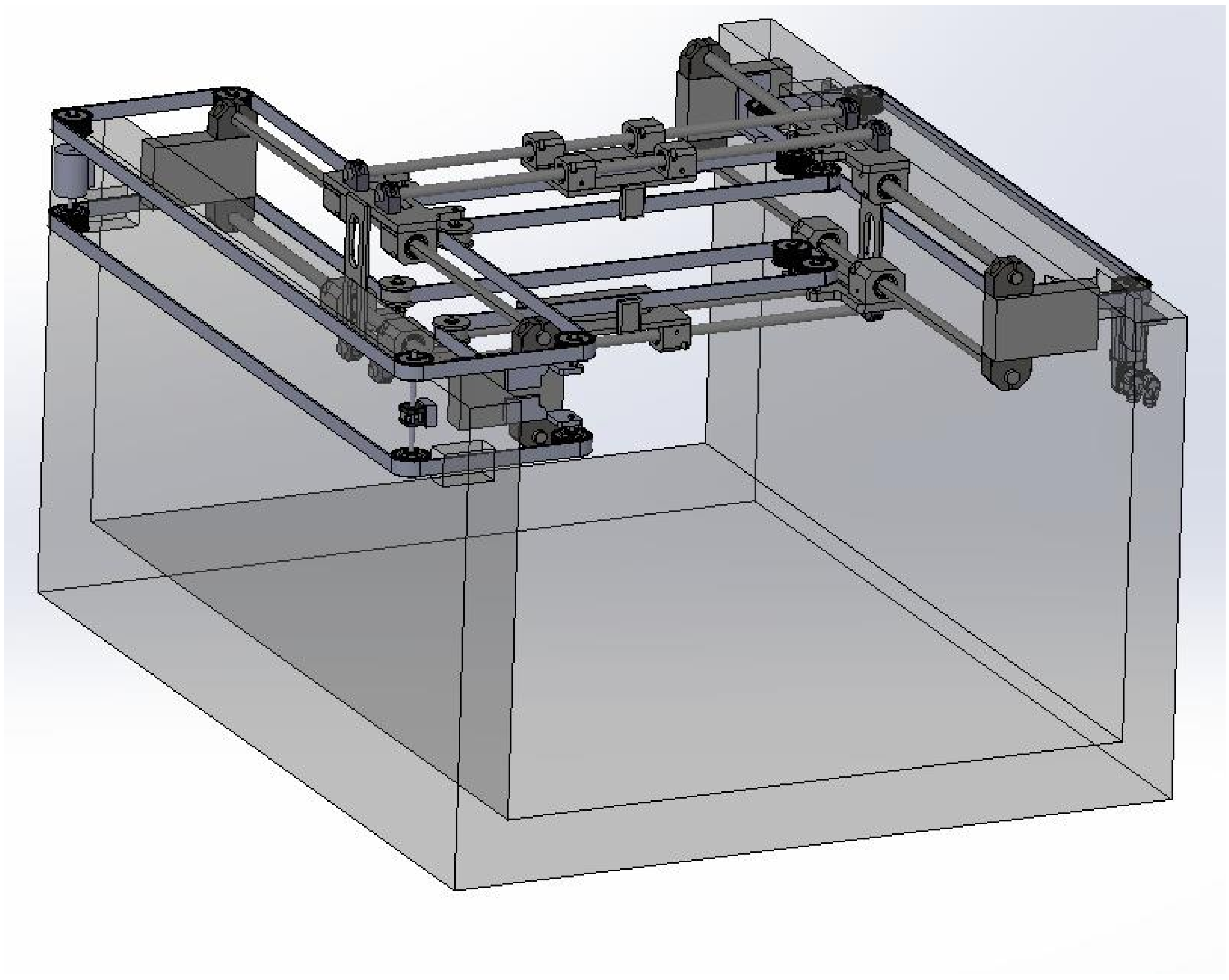


Figure 18: Machine side view

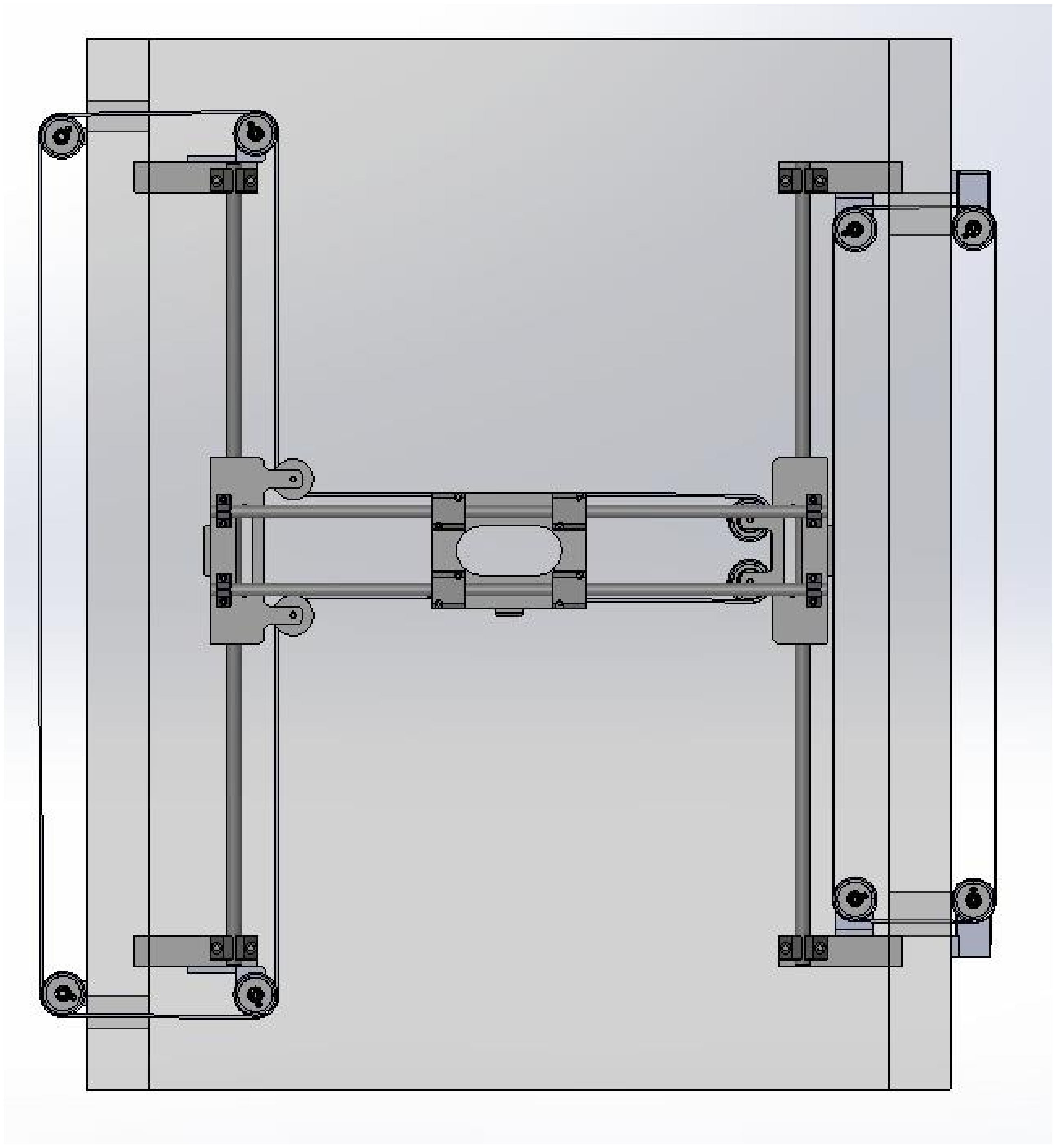


Figure 19: Machine upper view

3 Electronics

The main topic of this chapter is related to the electronic hardware used to movement and control of the end effector, and the hardware need to setup the Human-Machine-interface. Both controls systems are managed by a PC, that is the intelligent part of the control structure, where the computation is perform, and it assume the configuration of the PC as master with the slave (motor controller) using star network topology.

As we can appreciate in the Figure 20, the PC is placed between the HMI, used as input device, and the motor controller, that are specialized in control of motors. The motor drivers are connected to the motor controller in order to actuate the stepper motors with the step and direction signals. It has the function of amplify the power.

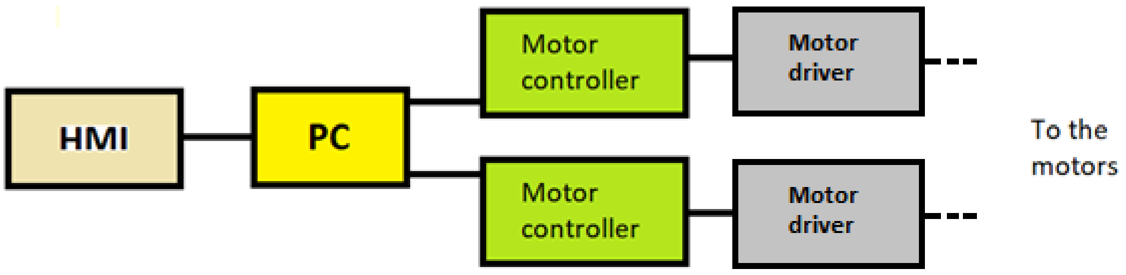


Figure 20: Electronic architecture

3.1 Requirement analysis

3.1.1 HMI

The HMI is responsible for shown the machine status, start a cut program and monitoring the correct execution. Our requirement restricts the research to a screen with a HDMI port. Future improvements could conduct to a touch-screen display.

3.1.2 PC

The PC must be enough powerful in term of power of computation to handle the HMI and the motor controller. It must has one HDMI port for display control and 4 USB port to manage two motor controller, one mouse and one keyword. The PC must have GPIO pins or a dedicated card for in/out in order to use it for read environment variables (that are the emergency, the limit switch, and the encoder). Furthermore, the choose will be oriented to small-size PC to save space in the control box.

3.1.3 Motor controller

The motors controllers should be connected in an easy way using USB standard connector provided by PC also they can support the serial communication. The motors controllers are responsible to control motor revolution and the managing of the alert signal of the emergency and the limit switches. The control is perform using step and direction command. The step is a train of 50 % PWM pulse, and each pulse of this signal causes the move of the motor by one step, so for instance a train of 50 pulse corresponds to 50 steps of the motor. The direction signal sets the direction of rotation (for example, "logic 0" for clockwise rotation and a "logic 1" for counterclockwise rotation). The controllers must be provided with a GPIO pins used to communicate to the motor driver the step and direction commands and, also, used as input for the limit switch and emergency. Furthermore, the motor controller must be enough fast in order to create a pulse train with frequency of 476.2 Hz that corresponds to the maximum displacement speed of the end effector (see chapter 2.2.6).

3.1.4 Motor driver

The motor driver is used to generate the necessary power to drive the stepper. They amplified the signal in current and voltage of the motor controller, that cannot be used due to the less power. Some type of specialized driver could be configured via software for a specific type of motor to enhance the performance. The motor driver must fit the 24V of rated tension of the stepper motor and must be able to supply 4.2 Arms of maximum current.

3.2 Component selected

3.2.1 HMI

The HMI is not defined because all screen with a HDMI interface fit our requirements.

3.2.2 PC

The PC choose is the Raspberry Pi4 B 4 Gb of ram. The choose fits the small-size and port needed requirements. Raspberry is equipped with GPIO pins, that is possible to set their status as input or output by software. The GPIO maximum input voltage is 3.3V and is considered as HIGH logic status if the tension applied on the pin is greater than 1.8V, on the contrary, is considered as LOW logic state if the tension is less than 1.8V. We will use these pins as input for the encoder, for the emergency and for the limits switch. The ARM architecture of the processor allows to save energy with sufficient performance. Furthermore the operating system is the open source Raspbian, derived from Ubuntu distro, freeing us from paid licenses and is continually updated. The operative system provides all software needed to develop the software part and for the creation of the executable application. Also, in the Figure 21, we can appreciate the small dimension.

Figure 21: Raspberry

3.2.3 Motor controller

The motor controller chosen is the Arduino UNO. Arduino series provided a wide range of controller catalogued according to the number of GPIO, dimension and speed of script execution. Arduino UNO satisfied our requirements. It is equipped with 14 GPIO where each one of them can be set as input or output in the script as our wishes. The input pins status is considered HIGH if the tension is greater that 3V and LOW if the tension is low than 3V. The Arduino is supplied by the PC (Raspberry) using the USB cable.

3.2.4 Motor driver

As motors drivers we have chosen two of DDS1148 produced by the LAM technologies company. It accepts a range of supply tensions between 20V and 50V, so considering the usual stepper tension we can select the 24V and 48V. We decide to use the 24V for supply the stepper according to the torque requirement analyzed in the chapter 2.2.3. It can manage a drive current between 2 and 10 Arms, then it fit our stepper constraint since the motors work with 4.2 Arms. The digital input range voltage for the step and direction signal is between 3 and 28V, so the output voltage of the Arduino motor controller can communicate with the driver since Arduino provide an output tension between 0 and 5 V. The chosen driver can be programmed using the proprietary interface and software in order to set the motor parameter (step in a full rotation, maximum supply current).

4 Software

The main topics of this chapter are the software requirements, the final realization of the application in Raspberry and, the control program in Arduino. The result will be an .exe application runnable in the PC. The computer application has been written in Python language [5] and, Arduino part has been written in the Arduino's proprietary language, similar to C++ [6]. Python language is interpreted by the programming console and is high-level general-purpose code. Furthermore, it is supported by a wide number of libraries for every application and offer the possibility to converting the script into an executable application with the necessary software module.

According to the Figure 22 in the Raspberry software part is implemented by the GUI in order to create the interface between the user and the machine. Then we will define the main module used to control the machine, that are:

- The graphic interface, with the aim of creating the way where the user command the machine.
- The master and slave communication, aiming to create a dialogue between Raspberry and the Arduinos controllers.
- Joystick controller, to move the end effector in the willed point of the working area.
- G-code translator and motion, with the purpose of converting the G-code language in a step and the direction format to apply to the motors
- Homing and base selection, to select the specific point in the working area that is assigned as base (home). And Homing, in order to have a procedure to calibrate the machine and move the end effector to the home position.
- Cutting map, to visualized on the screen the cut performs in the working area
- Emergency, if the emergency button pushes, it stops immediately the motor movement.

Since we must take as input the emergency, the encoders and the limit switch signal we use the GPIO of Raspberry and Arduino to provide to the intelligent part the access to the environment variables. In particular, by using the GPIO provided by Raspberry we can set the emergency, the encoders inputs and, the limits switch signals as inputs. The architecture of the software part is set by using as Raspberry as master and control Arduino as slave [7]. The most important reasons why it leads to separation are the performance limit of each component and the industrial law constraints. In fact, the pc (master) is a general-purpose device that is able to execute a large amount of data computation but cannot used to create step and direction command to control the stepper motors since is not provide enough security. On the contrary, the Arduino (slave) is a simple and reliable controller provided with a GPIO that is able to generate step and direction to control the stepper. In order to communicate with Arduino (slave) the software implements the master-slave communication using serial protocol on the universal serial bus (USB).

So, according to the Arduino part, we can see that there are present the module of:

- String decoder, with the aim of interpreting interpretate the message delivery from the raspberry.
- G-code movement, a module able to perform G-code displacement.
- Simply movement, a module linked to with the joystick in raspberry or to the homing function. Consequently, it performs the movement if one button of the joystick is pressed or if the homing function is called.
- Emergency and limit switch handling

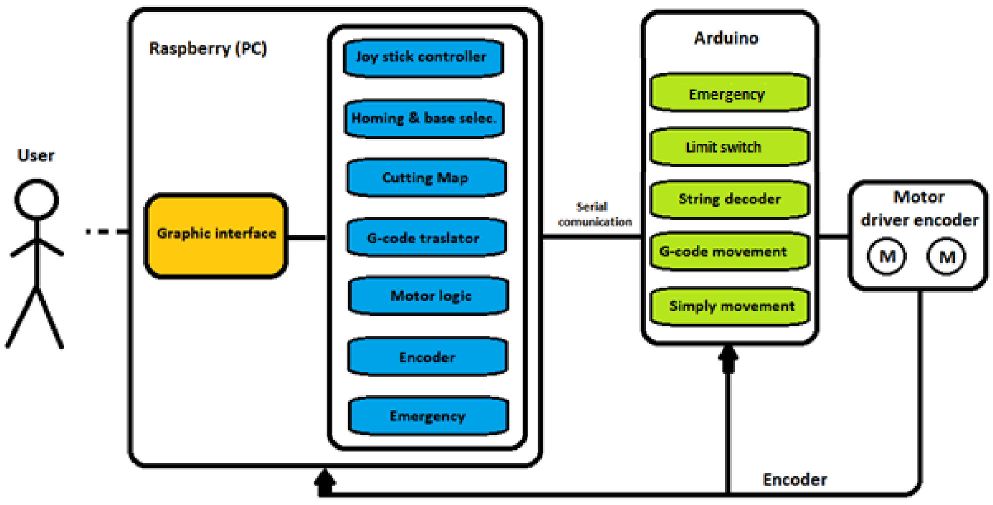


Figure 22: Software architecture

The application developed can only run on the Raspberry PC, since we use its GPIO pins and the correspond library of it. In order to have a runnable application in every PC is necessary to connect a GPIO card on it and adapt the script including the property card drivers.

4.1 Requirement analysis

This section analyses the reason of the implementation of each module and how all constraints and requirements are listed in order to respect them in the implementation.

4.1.1 Graphic interface

The graphic interface is used by the user to interact with the electronic device through graphic icon such as button, input text box, etc. These elements called "widget" that allows the interaction with the data that they hold creating a communication between Human and the PC. The functions called by acting on the widget (pushing or inserting text in the widget) are implemented and linked to the widget. The develop of the graphic interface aims to create a lightweight, reliable, and quick to learn interface. In this project the graphic interface must implement these widgets:

-One joystick to move the nozzle in the cutting area

-One widget to navigate in the pc folder to select the G-code text file to perform the cut

-One home button

-One real time image that is continuously updated to show the cut performed

-4 symbolic led to show the status of the limit switch, if are reached and enable.

Furthermore, the button must contain the name of the function called in order to be more intuitive.

4.1.2 Master slave configuration and motor control in Arduino

The master-slave is an asymmetric model of communication where one device called master controls one or more devices called slave. In this project this type of communication is used to separate the intelligent part from the actuator part, that was realized by the data exchange between raspberry and Arduino. Raspberry is responsible for the data calculation, computing the needed movement to impose to the motor, and Arduino actuate the willed movement creating the step and direction signals. The idea of the communication uses messages with unique meaning. Raspberry send to Arduino messages where each of them correspond to a particular function that Arduino must execute. So, it’s necessary to establish a defined set of messages with the correspond meaning. In just a few words, the communication works in this manner: Raspberry sent to Arduino a message (that is a string), Arduino associates it to the message (string) to one and only one function and perform the willed motion.

We should mention that, since a misunderstanding of message or a wrong delivery can conduct to a wrong cut path, ruining irremediably the piece so, the communication must be as much reliable as possible, in order to avoid the possibility of wrong sending and possibility of misinterpreting the meaning of the messages. In addition, after the receiving of the string, Arduino has to interpretate and extrapolate the information in order to actuating the desired motion. The emergency and the limit switch must be continuously check during the movement. In case of an emergency status the motor must be stop immediately, and in case of reach of a limit switch the motion has to stop and a dedicated procedure must free the limit switch.

In short, our requirements are:

- A list of strings used in the communication where correspond to the function called
- It's necessary to have a serial communication between slaves and master
- Emergency and limit switch handling

4.1.3 Joystick controller

The joystick is an input device able to control in the space (2D or 3D) the movements of the controlled object. In this project we need the joystick to move the nozzle in the machine cutting zone, where the food is positioned in order to start the cutting pattern or set the point as base. The desired joystick is made on the graphic interface by 4 widgets, that correspond to the up, down, left, and right directions. Pressing one button the software must be able to start the motion in the desired direction and at the release of the button it must stop the instantaneously the motor movement. Furthermore, in this situation it's not relevant the possibility to have a variable positioning velocity set by the operator; in the project the movement velocity is set as a constant. To recap, our requirements are:

- The control of the motion in the desired direction
- The presence of 4 button widgets on the graphic interface.

4.1.4 G-code translator and motion

The G-code translator is responsible of the extrapolation of the information from the G-code text file. The G-code translator needs a graphic widget able to navigate in the PC folder and select the desired G-code file. When the file is select, are necessary proper functions to open the G-code text file line by line in order to extrapolate the information from each line. The translator must recognize what G-code function is set (G00, G01 or G02) and compute the distance between the start point to the arrival point. Furthermore, it must be able to decodify the velocity that has been written in the G-code command (if it is present, due to it could be absent). In case of a circular path (G02), it must be able to perform a linear interpolation in order to divide the path in intermediate straight line. In order to display the time required to complete the cut we have created a progress bar, that is updated whenever a line of G-code is translated, and the desired path is perform. It is really important that the machine execution of cut, and the decoding of line goes together to have a real indication of the progress bar.

The relative distance and the velocity (if it is present) are taken as input from the motor logic function, since distance cannot be proportionally transformed in step and direction due to the unconventional pose of the stepper motor must be adapt to the correct conversion logic.

In short, our requirements are:

- The widget able to navigate into the PC folder
- Function able to open and read the G-code file with txt format
- The decisional process able to extrapolate the information
- Linear interpolation in case of G02 command (circular path)

4.1.4.1 Motor logic function

The command logic is used to create the string that must be send to Arduino. The inputs that it has to take are the relative distances in X and Y and the extrapolated velocity by using the G-Code translator. After that, it must return the strings to be sent to each Arduino. Also, the command motor logic refers to the unconventional pose of the motor, transforming the relative distances into the transformed step and direction command. Also, it has the role to guarantee the equality between end effector displacement and motor displacement. For example, if an entire rotation of one motor produces an end effector displacement of 10 cm and require 200 steps of the motor, it's quite clear that we need 20 steps to make 1 cm. So, the command motor logic must guarantee this proportionality, multiplying the transformed distances by the proportional constant. Another issue that the function must deal with is the different lengths of motor's pulse train. For example, if a motor U has to make 10 pulses and motor V has to make 5 pulses, both pulse trains must start and finish in the same instance. Doing so we obtain the starting and the finishing of rotation of both motors in the same time instance. In short, our requirements are:

- To transform the relative distances into the transformed distances according to the unconventional pose of the motors
- To guarantee the equality between the willed displacements and end effector displacements
- To guarantee the start and the end of both motor movements in the same time instance.

4.1.5 Cutting map and limit switch reader

The map is used to display the performed cut on the screen without seeing the machine during the work. It is constructed taking as input the encoder's signal and due to the unconventional pose of the motor, it is necessary to manipulate this signal. The map is implemented as widget on the graphic interface of the application and the end effector is visualized as a little square inside the widget. The displayed square correctly follows the end effector, for example if the user performs a movement in the positive X axis direction the displayed square must do the same with as less delay as possible, ideally in real time. When the cut starts, the map must visualize the moving square and in the point in which it passes the map must be colored in order to symbolize that the cut has been performed. For instance, if the previous G-command was **G01 X0 Y0,** and the executed G-command is **G01 X0 Y10** the map must visualize a straight line starting in the point X = 0 Y = 0 and finishing in the point X = 0 Y = 10. It's pretty clear that the straight line must be constructed in real time with the end effector advance.

In the construction of the map, the software must distinguish the G-Code commands, in fact the command **G00** means a rapid positioning in the set point and therefore, the path must not be colored, instead, since the commands **G01** and **G02** require a straight line and a circle with the coloration of the map. Furthermore, it's essential to have a precise map on the computer in order to increase the security because for the operator, is useless to see the machine during the work. Furthermore, we analyze the limit switch signal. The immediate stop of the movement, at the reach of one limit switch, is implemented in the motor controllers (Arduino). Then, we have to treat the communication to the user of which limit switch has been reach, for that we need a graphical indicator.

Summarizing our requirements are:

- The needed of different map construction in case of cut and in case of displacement
- A graphic widget
- A real time map
- An encoder signals condition circuit
- The G-code discrimination
- Graphical indicator of which limit switch has been reached.

4.1.6 Homing and base selection

The homing function is used to return the end effector in the home position. The home position is set by the software using a default point at the first running of the machine, but the operator can change its the position and can set it in a specific point.
These are some reasons why the implementation of this function is required:

- Firstly, due to the unclosed path of the G-code figure, since not all figures start and finish in the same point. For example, can happen that a zig-zag path has not a common starting and arrival point, so a function that returns the end effector in a specific point is necessary.
- Secondly, during the execution of the cut, the position error could accumulate cut by cut and the machine would become less precise. Therefore, a calibration procedure is required.
- Thirdly, if the end effector touches a limit switch occur a misalignment between the actual position of the end effector and the position known by the machine control software
- Fourth, the homing position is necessary in case of a sudden power-off of the PC, that leads to the loss of the position recorded by the encoders, as they are relative but not absolute.

In order to achieve an accurate calibration of the machine we need an absolute reference, and, in our case, it is given by the point where both the limit switch will activate, in one corner of the cutting area. Summarizing our requirements are:

- A precise recalibration procedure of the machine
- The possibility for the users to choose the home position

4.2 Description of the implementation

The main topic of this chapter is how the software will be implemented in relation to the constraint previously analyzed. Each module contains:

- The graphic interface: this module refers to how to set the graphic interface.
- The master and slave communication in connection with the protocol that will be used and how it must be implemented on Raspberry and Arduino. In addiction in this module is contained the script of Arduino related to the control of the motor and the handling of the emergency and limit switch status.
- The joystick: it refers to how to create the graphic widget button and the motion command in Arduino
- The G-code translator and motion: we will explain the process to use in order to extrapolate the information from the commands (G00, G01 and G02), the linear interpolation of the G02 path, and the logic to command the motors (Motor logic function),
- The homing procedure and base selection: this module show how the new base is select, the procedure does in order to obtain a correct recalibration, and the relative motion command in Arduino
- The cutting map widget and limit switch reader. The implementation of the cutting map widget and the checking of the limit switch's status

4.2.1 Graphic interface

With reference to the requirements, we must have a lightweight and reliable graphic interface with a widget as button in order to search folder and display images in the PC.
Thanks to Python, we can create a GUI called TKinter that is cross-platform and, so, the same code can work on any operating system. In connection with its visual elements, they are rendered using the native system elements, so the application will look like they belong on the platform where the application has been run.
TKinter provides the necessary widget with the aim to suit our requirements. Also, we are allowed to modify the widget in every moment and to our liking, for example, a button can be created with the desired dimension and color.

4.2.1.1 Setting of the TKinter graphic interface

In the Figure 23 we can appreciate the code used to set up the graphic interface. For first are called all the libraries that we will used below in the script (where is present the slashes). In particular the called libraries are the main module of GUI (import TKinter as tk), the searcher in the pc folder used to open G-code files (from tkinter import filedialog) and the Progressbar used to represent the progress of the cut operation (form tkinter.tkk import Prograssbar). Second are set the graphic main loop and the window characteristic. The script relative to the graphic part between these commands are run in a loop in order to update the GUI and take the input (for instance the pressing of a button). Also, we set the graphic characteristic of the window, in particular with a size of the window as 700 pixels wide and 800 pixels high, then is set the name of the application shown in the window's header as Wenji.

```
import tkinter as tk
from tkinter import filedialog
from tkinter.ttk import Progressbar

window = tk.Tk()
window.geometry("700x800")
window.title("Wenji")

/////////////////////////////////////
-------------------------------------
/////////////////////////////////////

window.mainloop()
```

Figure 23: Raspberry setting of Tkinter graphic interface

4.2.2 Master slave communication and motor control in Arduino

The communication between master (PC) and slave (Arduino) is realized through serial protocol and the communication hardware is an USB cable, that connect the Arduino to the PC. Since we have the task of implementing a safe and reliable communication between the master and the slaves, we decide to use a handshaking method. In short, the communication is divided in 6 steps.

- We have to establish the master slave connection through serial port
- In order to make sure the connection status we have to send an accomplishment message from the Arduino to the PC
- So, the next step is to send the message from the PC to the Arduino
- The interpretation of the message and actuation by Arduino
- Sending a responsive message from Arduino to PC
- Close the master slave connection

In particular, the connection is set up by using serial protocol library of python where is also set the transmit frequency at 9600 bound for second, so, that means that is sent 9600 symbols in a second. Then, in order to a have more reliable connection, Arduino will send a confirmation to the PC and puts itself in listen mode waiting for the arrive of a message. Next, if the message is received by the Raspberry, the connection is considered reliable, and the PC start to transmit the message. Then Arduino decodes the message and start the motion. At the end of the movement Arduino sends a completion message. Finally, the connection is close by the PC.

Based on what has been analyzed in the requirements section, the communication is achieved by sending messages that have a unique meaning. There are 2 types of messages:

- one character (case sensitive)
- a string made of more character (case sensitive)

The character is used to call one specific function in Arduino call from the PC or are used by Arduino to response to Raspberry. For example, pushing the right joystick button make the delivery of the letter “R” from Raspberry to Arduino, that decodifying and calling the willed function start the motion in X direction. Also, the character is used by the Arduino, to communicate the accomplishment of the movement or an emergency status. The string is used to communicate the interpretation of G-code command to each Arduino (2 different string, one for each motor controller). It contains the direction, the required step and the velocity extrapolated from each G-code command, for example, <Forward,1000,50>. Since we have 2 motors, we have also two Arduino controller, assuming the same nomenclature of the motors we have the Arduino U and Arduino V. The list of messages related to the Arduino’s controllers, is different between each controller. This is carried out by the unconventional pose of motor, for example: in case of joystick movement in the X direction we need to actuate both motor (as has been demonstrated in chapter 2.1.4) so the Arduino U must contain the X and Y movement, but the Arduino V must contain only the Y movement. In the table below are listed all messages for each Arduino controller and is shown the corresponding movement of function at which they are associated.

Considering the Table 1, we can appreciate the message-function called correspondence of the Arduino controller of the motor U

Start marker	“<”
End marker	“>”
Homing X fast	“H”
Homing X slow	“P”
Homing X find	“F”
Homing Y fast	“h”
Homing Y slow	“p”
Homing Y find	“f”
Joystick Y upward	“U”
Joystick Y downward	“D”
Joystick X leftward	“L”
Joystick X rightward	“R”
Joystick stop movement	“S”
General G-code forward	<Forward,#steps,#velocity>
General G-code backward	<Backward,#steps,#velocity>

Table 1: Table of messages relative to Arduino of the motor U

Considering the Table 2, we can appreciate the message-function called correspondence of the Arduino controller of the motor V

Start marker	“<”
End marker	“>”
Homing X fast	“H”
Homing X slow	“P”
Homing X find	“F”

Joystick X leftward	“L”
Joystick X rightward	“R”
Joystick stop movement	“S”
General G-code forward	<Forward,#steps,#velocity>
General G-code backward	<Backward,#steps,#velocity>

Table 2: Table of messages relative to Arduino of the motor V

The symbol that both Arduino controller can send to raspberry are shown in Table 3.

Emergency	“<!>”
End of motion	“<#>”
Connection successful	“<@>”

Table 3: Table of messages in common in both Arduino

4.2.2.1 Raspberry master part

This section deals with how the Raspberry send to each Arduino 2 types of strings, which are:

- One composed by the direction, the number of steps and the velocity, that for instance has the following orthography: <Forward,1000,50>
- One composed by one character, for instance: “F”

In addition, it has to ensure a reliable communication that have go through a rigid protocol of hand shacking. This last one has been implemented in this way:

- Raspberry open the serial port.
- Wait the response from Arduino.
- Send the message to Arduino.
- Raspberry sense the acknowledgment symbol from Arduino.

In Figure 23 is demonstrated how the serial port is opened using python. How we can see the software opens the serial communication with the device connected to the USB0 with a bound rate of 9600. The serial communication is saved as variable with name `serial_U`, that taking the name of the motor is the serial communication dedicated to the motor U.

```
serial_U = serial.Serial('/dev/ttyUSB0',9600)
```

Figure 24: Raspberry setting of the serial port

Then Raspberry has to wait the response from Arduino. As we can see in the Figure 25 in the function `waitForArduino` a while loop sense if the message sent by Arduino is the character “@”, if it is true we make sure that the connection was successful and can been consider reliable.

```
def waitForArduino():
    message = ""
    while message.find("@") == -1  :
      message = recvFromArduino()
```

Figure 25: Raspberry wait for Arduino function

Next, Raspberry can send the message. In Figure 26 is shown the function responsible of the delivery and, as we can see, by using the method write is sent the encoding of the string. The encoding allows to transform the alpha-numeric string into a string made up by 8-bit characters.

```
def Send_to_Arduino(String):
    waitingForReply = False
    if waitingForReply == False:
        serial_U. Write(String.encode())
        waitingForReply = True
```

Figure 26: Raspberry wait for Arduino function

At this point the string has been sent and Arduino can start the extrapolation of the data and in order to begin the motion. When the motion is complete Arduino send the completion symbol "#" in order to communicate the end of the motion. With a view to implement this function we need to put Raspberry in listen mode and sense the serial port in order to catch the message, as we can see in Figure 27. Since every message that has been sent by Arduino must be start with the symbol "<" (called start marker) and must finish with the symbol ">" (called end marker) we perform a while loop to sense if Arduino have sent the start marker. If this condition is true, so the symbol "<" has be sent by Arduino and the code continues to save all the character sent by Arduino until the end marker symbol ">" declare the finish of the message. This function is used to receive every type of messages from Arduino. In addiction the function `Receive_from_Arduino` is responsible for counting the steps made from the motor. As we can see, when the function tries to read the serial bus in seeking the start marker symbol by using a while loop, the function also calls `Encoder_U` that is responsible for the counting of the steps performed by the motor.

Since the moment after the delivery of the G-code motion string, the master puts itself in listen mode, searching the start marker and, so, we can start immediately the count of the encoder in order to avoid the loss of counting. So, we do not incur in delay between the start of motion and the start of the encoder's counting, and these 2 functions are synchronized.

Since the message in the serial communication are encrypted using the UTF – 8 coding we need to use the function “ord”. It takes as input the message read from the serial port and returns an integer corresponding to the Unicode character. The start marker and the end marker correspond to the integers 60 and 62 respectively. So, in the first while loop the brake condition is represented by the read of the start marker and in the second while loop the break condition is the read of the end marker integers. In addition, in the second while loop there is the composition of the receive message. As we can see, until is not read the end marker symbol, the message is constructing adding the message arrived.

```
def Receive_from_Arduino():
    message_reiceved = ""
    message_read = ""
    counter_U = 0
    clkLastState = GPIO.input(4)
    while  ord(message_read) != startMarker:
        Encoder_U()
        message_read = serial_U.read()

    while ord(message_read) != endMarker:
        if ord(message_read) != startMarker:
            message_reiceved = message_reiceved +
message_read.decode()

        message_read = serial_U.read()
    return(message_reiceved)
```

Figure 27: Raspberry receive from Arduino function

In the Figure 28 we can see the `Encoder_U` function. Firstly, in the function `Receive_from_Arduino` is initialized the last state of the encoder. Practically is saved the logic value of the pin number 4 as HIGH if the voltage is above 1.8V (that is the logic threshold of Raspberry) or LOW if the voltage is below 1.8V. Secondly, in the function `Encoder_U`, is check again the logic value of the pin number 4, if it is changed we are sure that the encoder has made a pulse, so the counter is updated and the last state of the encoder is set as the new. Is really important that the frequency of sense of the encoder status (frequency with the while loop is performend) is higher than the frequency of the encoder output, in order to do not loss encoder pulse. Also, in the `Encoder_U` function is checked if the emergency push button has been pressed. By using a positive logic is consider the emergency whenever the value is LOW, and we take into consideration the safety when the value is HIGH, in other words the logic zero is the HIGH level of tension and the logic one is the LOW level of tension. We have implemented this convention for the reason that, in case of malfunctioning of the push button or in case of a cut of the wire, the state will become LOW, and the emergency will be instantaneously activated. The emergency is checked every while iteration, and if it will activate the software is closed causing the stop of the machine.

```
def Encoder_U():
    if GPIO.input(14) == False:
        print("emergency system close)
        sys.exit()
    clkState = GPIO.input(4)
    if clkState != clkLastState:
        counter += 1
    clkLastState = clkState
```

Figure 28: Raspberry encoder function

In the Figure 29 we can see the function that is responsible of the delivery of the G-code string. It's noticeable that the code contains all the function presented before in order to ensure a reliable connection and communication. Then, when the motion is completed, the serial port is closed and is check is the movement perform by the machine is equal to the imposed one. In case of loss of steps, the missed steps will be applied again. This control is implemented by comparing the number of steps imposed with the steps counted by the encoder, and, in case of lack of match, the error is computed and is sent again in order to obtain the willed movement. The machine can only loss steps due to the friction or other cause. We check also is check if Arduino has sent the emergency symbol "!", that may seem superfluous in the first analysis, seeing as the emergency status is checked also in the function shown in Figure 28. But, since the emergency status must be as much reliable as possible and in case of malfunction of Raspberry PC (software or hardware, in case of cut of the emergency button cable, dedicated to its communication) it cannot be able to understand the emergency status. So, we decide to implement the recognition this important status even in both Arduino controllers. So, has been chosen to sense if both Arduino send the emergency symbol "!" and in affirmative case close immediately the application.

```
def Movement_motor_U_G_code(String,Str_U_direction,STEP_U,Speed_U):
    serial_U = serial.Serial('/dev/ttyUSB0',9600)
    waitForArduino()
    Send_to_Arduino(String)
    msg = ""
    while msg.find("#") == -1:
        msg = Receive_from_Arduino()
         if msg.find("!") == -1:
             serial_U.close()
             sys.exit()
    serial_U.close()
    if counter=!steps
        error = steps-counter
        string_residue_U =
'<'+Str_U_direction+','+str(abs(error))+','+str(abs(Speed_U))+'>'
        Movement_motor_U_G_code(string_residue_U)
```

Figure 29: Raspberry G-code movement 13

4.2.2.1 Arduino Slave part

On the basis of what we have said in the Arduino controller requirements about the connection and communication process, this part has been implemented following these steps:

- At the calling of the serial bus opening by Raspberry, Arduino must send the symbol “<@>” to ensure the correct connection
- Arduino must stay in listen mode to receive the data from the Raspberry.
- After the motion execution, it must send the completion symbol “<#>”

In addition, this part has to deal with limit switch signal, blocking the motion if we reach one of them. As we can see in the Figure 30, in the function `void setup` are initialized the limit switch signal and the emergency as input as well as the pin relative to the waiting function that is used to synchronize the motion (it will be explained below).The function `void setup` will run every time that Raspberry opens the serial bus, and, since the confirmation of connection must be sent, in the function is present the delivery of the symbol “<@>”

```
void setup() {

  pinMode(7, OUTPUT); //Waiting function output
  pinMode(6, INPUT);  //Waiting function input
  pinMode(2, INPUT);  //Limit switch Y+
  pinMode(8, INPUT);  //Limit switch Y-
  pinMode(5, INPUT);  //Limit switch X+
  pinMode(10, INPUT); //Limit switch X-
  pinMode(11, INPUT); //EMERGENCY
  Serial.begin(9600);
  Serial.println("<@>");
}
```

Figure 30: Arduino pin setting

After that, Arduino must set the pin of command of the stepper motor. The control is implemented by using he library stepper.h that provides all the methods used to set the direction of rotation and set the velocity of motor. As can see in Figure 31 we can set the pin 4 as direction, the pin 3 as step signal and 200 pulses to complete one full rotation (as has been analyzed in the selection of the motor chapter 2.2.4). The declare is made in section where all the constants are declared.

```
Stepper Right(200, 4, 3); // Dir = 4, Step = 3
```

Figure 31: Arduino stepper motor setting

Then, the control passes to the `void loop` function show in Figure 32. It is run in loop and calls firstly the function to receive the data from the Raspberry. Secondly, when the data are been decoded, the function dedicated to the motion of the motor can be performed.

```
void loop() {
  getDataFromPC()
  motor()
}
```

Figure 32: Arduino void loop function

The function `getDataFromPC` is responsible for the receiving of the data from the PC. Initially, is sense if the messages on the serial bus are available (as we can appreciate in the Figure 33) and if it is true the message are saved on the variable "char x" letter per letter.

```
void getDataFromPC() {
  if(Serial.available() > 0) {
    char x = Serial.read();
```

Figure 33: Arduino get data from Raspberry

Then we have to distinguish the G-code strings and the strings composed by only one character. In case of G-code string, we can see that the orthography is organized as a start marker "<", the message "Forward,1000,50" and the end marker ">". On the contrary in, the string composed by only one character, the orthography is just the letter, without any start or end marker. Since we read the message on the serial port letter per letter, we have decided to use the star marker "<" as discriminatory symbol. So, if in the first character there is the symbol "<" we are sure that the Raspberry is sending the string relating to a G-code movement. So, it is necessary to store the whole message in a variable until the end marker ">" is reached in order to decodify the direction, the number of steps and the velocity. The de-codification is implemented separating the G-code string in the point where there is present the comma, ",". For instance, the string <Forward,1000,50> is divided into 3 sub strings, in the first there is the direction (Forward), the second represent the number of steps (1000) and in the third is in relation the velocity (50). If the first character is a letter (Table 1 or Table 2) we are in present of imposition by a simply move (joystick move or home procedure move). The presented procedure to the recognition of the strings is implemented by using a conditional process.
When a movement requests the rotation of both motors (most of them) appears a problem, the delivery of the messages is not synchronized between them and conduct to the actuation of one motor before than the other. The problem is carried out in the execution of the code in the PC, because the time requested in the computation of the string for each Arduino controller is different and aleatory. So, the serial delivery of messages is not synchronized, and one message can be sent before the other. The solution is to implement in both Arduino controllers a synchronizing function before the actuation of the motors. As we can see in the Figure 34, when the message is received, and the proper function has been called before the actuation of the motor the function `Waiting` is called. In the function there are one output and one input pins for each Arduino and the output pin on one Arduino is connected to the input of the other and vice versa. The function sets one output pin to high status and measure the status of one input pin in each Arduino. If both output pins are being set on high status and both input pin are at high status, we have the synchronization of the Arduinos.

```
void Waiting(){
  digitalWrite(7,HIGH);
     while (digitalRead(6) == LOW){}
       }
```

Figure 34: Arduino waiting function

The following example concerns the activation of the waiting function. The output pin number 7 of the Arduino U is connected to the input pin number 6 of the Arduino V; the output pin number 7 of the Arduino V is connected to the input pin number 6 of the Arduino U. If the Arduino U receives the message before the Arduino V, Arduino U sets its pin number 7 HIGH, and the control passes to the while loop waiting for the activation of the input pin number 6. Since the Arduino V has not receive the information, it cannot set its output pin number 7 HIGH, so the Arduino U remain in the loop until the Arduino V sets the pin 7 HIGH. In case of a movement that involves only one motor, is sent a string even to the not acuate motor with the imposition of 0 step, in order to pass the `Waiting` control.
The emergency status checking has been implemented in the Arduino code and it detects at every step of the motor if the dedicated input goes at low status. If it happens, we send the symbol "!" to Raspberry and we stop immediately the movement. In addition, at every motor step, are checking the limit switch status, and in case of reach one of them the movement is block and is perform a movement in the opposite direction in order to release the switch. Considering the Figure 35 refers to G-code movement in the forward direction of motor U (function `Forward`). This function is called if in the G-code string sent by the Raspberry, is contained the direction "forward". As we can see the are set the velocity and the number of steps of the stepper as the willed constants extrapolate in the receive string. This function is also implemented for the movement in the opposite direction, the forward. In the image we can appreciate the checking of the emergency and limit switch status. In case of the detection of the emergency status the for loop is break. In case of limit switch reaching is called a proper function according on which limit switch has been activated. This function has the task of move the end effector in the opposite direction in order to release the limit switch. Even in this case the for loop is break. If do not occur violation, the for loop perform the desired movement, and at the end, call the function `Waiting` to synchronize the sending of the competition symbol "#".

```
void Forward(){
Right.setSpeed(velocity);
Waiting();
if (steps != 0){
  for(int s = 0; s < steps; s++){
  Right.step(-1);
  if(digitalRead(11) == HIGH){
    Serial.println("<!>");
    break;}
  if(digitalRead(2) == HIGH){
    Limit_switch_touch_Y_plus()
    break
    }
  if(digitalRead(8) == HIGH){
    Limit_switch_touch_Y_minus()
    break
    }
  if(digitalRead(5) == HIGH){
    Limit_switch_touch_X_plus()
    break
    }
  if(digitalRead(10) == HIGH){
    Limit_switch_touch_minus()
    break
    }
}}
steps = 0;
Waiting();
digitalWrite(7,LOW);
Serial.println("<#>");
}
```

Figure 35: Arduino motor movement

4.2.3 Implementation of the joystick

According to what has been said in the requirements analysis, the joystick is constructed by using 4 buttons that symbolize the upward, downward, leftward, and rightward direction related to the machine reference system (show in chapter 2.1.4). Pressing one button, the widget calls a function that establishes the connection to the slave and sends the moving signal (one character). The slave interprets the direction associated to that signal and start the motor moment in the set direction. The slaves remain in listen of the stop signal while the motors are running, and when occur the movement is stopped. The sending of the stop signal is delivered at the release of the button by the users on the graphic interface. Below, there is a better explanation with the implemented script.
The Figure 36 represents the creation of the button with the name `Y right`. Then, using `.bind` method is linking the press of the button to the function `Y_button_press_MOVE` and the release to the function `Y_button_press_STOP`. That function will be called at the press or the release of the button.

```
Y_button_right = tk.Button(text="Y right")
Y_button_right.place(height=50,width=80, x=200, y=50)
Y_button_right.bind('<ButtonPress>',Y_button_press_MOVE)
Y_button_right.bind('<ButtonRelease>', Y_button_press_STOP)
```

Figure 36: Raspberry joystick button setting

By using multiprocessing library, the function `Y_button_press_MOVE` calls the function `MOVE_motor_U` and the function `Y_button_press_STOP` calls the function `STOP_motor_U` as is demonstrated in the Figure 37. The library multiprocessing is used in order to create more than one process and we need it in order to execute parallelism in the application. In fact, for instance, while the function `MOVE_motor_U` is called and it is on work, is necessary also to call the function relative to the cutting map with the aim to visualize where the end effector moves. Another instance is about the movement on the X axis because it involves both motors and it is necessary to spawned 2 processes where each one control one motor. The method `.start` is used in order to start the process, and the method `.join` is used to wait the code until the process finish.

```
def Y_button_press_MOVE (event):
    Y_move = multiprocessing.Process(target = MOVE_motor_U,
args=["U"])
    Y_move.start()

def Y_button_press_STOP (event) :
    Y_STOP = multiprocessing.Process(target = STOP_motor_U,
args=[])
    Y_STOP.start()
    Y_STOP.join()
```

Figure 37: Raspberry joystick-button linked function

In the script that we can see in the Figure 38, we find the implementation of the communication. Firstly, the serial bus is opened with a speed of 9600 bound for second. Secondly, we have to wait the status of ready from Arduino, finally we will send the symbol "U" that means the rotation of the motor in clockwise direction according to the willed movement in the Y positive direction. On the contrary, the communication of the speed is not required whenever is considered as constant. In the `STOP_motor_U` function, the symbol "S" is sent in order to stop the movement. In this function is not necessary to wait the ready signal from Arduino (@) because even if the serial communication is closed, it is in listen mode for the stop signal. See the image for a clarification of this point.

```
def MOVE_motor_U (String):
    serial_U = serial.Serial('/dev/ttyUSB0',9600)
    waitForArduino()
    Send_to_Arduino(String)
    msg = ""
    serial_U.close()

def STOP_motor_U ():
    serial_U = serial.Serial('/dev/ttyUSB0',9600)
    Send_to_Arduino('S')
    serial_U.close()
```

Figure 38: Raspberry joystick functions

In the Figure 39 is represented the function called with the symbol "U" using the serial communication in Arduino. As we can see if the serial communication is available is check is the receive message that corresponds to the symbol "U" (where the symbol is saved inside the variable `U_RIGHT)` , if it is true the motor speed is set as the default constant value of 100. Then, having set the motor parameter, the program starts a while loop and at each operation is perform one step of the motor and is check is the serial communication is available. The break condition of the while loop is given by the arrival of the "S" symbol, in fact the delivery of "S" symbol from PC set the Boolean variable stop true, that breaks the loop as it is in the while break condition.

```
void getDataFromPC() {
 if(Serial.available() > 0) {
   char x = Serial.read();
   if (x == U_RIGHT) {
     Right.setSpeed(100);
     while (STOP == false){
       Right.step(1);
       if(Serial.available() > 0){
        char x = Serial.read();
        if (x == 'S'){
         STOP = true;
         }}
```

Figure 39: Arduino joystick actuation

4.2.4 G-code translator

The G-code translator has the role of interpreting the G-code text file with the aim of extrapolating the requirement steps to move the end effector in the willed point. The implementation of the G-code translator starts by creating a widget able to navigate in the PC folder and it must be select the G-code text specific file. Next, the program, reads the content in the text file by extrapolating each row as a vector of strings. The vector is composed by 3 or 4 elements where the first is the G-command header (G00 or G01 or G02). The second and the third are the destinations (for example X10 and Y8) and the last, the fourth, represents the speed. The fourth is not mandatory and in case of lack the speed is set as the previous commands. Starting with the data extrapolation process, the string is sent to a decisional process in order to choose what is the G-command header (G00 or G01 or G02). This step is essential if we want to know what is the geometrical pattern that the machine must execute (rapid positioning, straight line, or circle). Then, the second element of the vector is taken (that will be X##) and removing the letter X and converting the string is a number we obtain the X axis destination point. This last process is executed also for the third element of vector in order to extrapolate the Y axis destination point. Finally, we have the destination point that is used to compute the relative displacement by doing the subtraction between the new and the previous destination point, saved on the machine as variable. The next step is related to the check of the vector length and if it is composed by 4 elements, we are sure that the last is the velocity, and the value is computed as for the second and third elements by removing the letter F and taking the follow number. If the length is 3, the speed coefficient is not given, and we have to take the previous coefficient. At the beginning of the translation, the first row does not have a previous destination point, and so is necessary to include 2 global variables used to store the last destination point computed in the previous G-code text file. On the contrary, if the application has not made a translation (so the previous destination point does not exist) a Homing procedure is necessary in order to calibrate the machine. This is caused by the relative nature of the encoder and the volatility of the last position variable store into the PC, that are erased when the application is closed.

The graphic interface tkinter provides filedialog widget. At the press of a button widget, a navigation window will be opened in order to give the possibility to the user to choose the G-code text file. As represented in the Figure 40 the content of the file selected is save in the variable `filename.` Next, the number of lines is extrapolated by using the method readlines. This number will set the maximum slots length of the progress bar and starting from the zero slot, at each execution of the G-code command a slot is added in order to monitor the progress of the cutting pattern. As we can see, the progress bar (`pb`) number of slot is set, as 100/(number of lines - 1).

```
filename = filedialog.askopenfilename()
number_of_lines = len(open(filename).readlines(  ))
pb = Progressbar(window ,orient = HORIZONTAL,length = 200, mode =
'determinate')
pb.place( x=300, y=230)
pb['value'] += 100/abs(number_of_lines-1)
```

Figure 40: Raspberry load G-code file

In the Figure 41 is shown the opening process of the file and the final extrapolation of the G-code commands. Firstly, using `open` we open the file saved on the variable `filename` with the name `G_code_file`. After that, the Y and X destination variables and the Y and X last position variables are set as global in order to have the possibility of modifying their value in the entire script. In addition, an empty vector is created with the name "Vector_G_code_line" that will be filled with a G-code line. Then a "for" cycle scans every line of the document in order to extrapolate the G-code commands. Inside the cycle the G-code command is converted into an array of string for the manipulation. For example, the G-code command G00 X1 Y5 F40 is converter in to and array as [(G00),(X1),(Y5),(F40)]. Finally, inside the loop, the length of the vector is computed in order to check if is present the velocity. If the length is 4 elements, the last will the velocity, on the contrary if the length is 3 the velocity is not given.

```
    with open(filename, 'r') as G_code_file:

        global Y_destination
        global X_destination
        global Y_last_position
        global X_last_position

        Vector_G_code_line = []

        for line in G_code_file:

            Vector_G_code_line = line.split()
            Vector_G_code_line = np.array(Vector_G_code_line)
            length = len(Vector_G_code_line)
```

Figure 41: Raspberry setting of extrapolation

4.2.4.1 Interpretation of G00 and G01

When the G-code command is converted into an array starts the decisional process. As is shown in Figure 42 the first element of the array is compared with G00 and if it is true the machine would understand that the geometrical pattern is a straight line. So, we have the task of extrapolating the destination point and eventually the velocity but as we have said, it is mandatory a manipulation of the array since the X and Y position number are followed by the X and Y letter (i.e., X1 Y5). So, the second element of the array is saved into variable `str` and the `Residual_string` is computed by taking only the first letter of the `str`. Then it is checked if it is the letter X in order to avoid errors during the execution. Then, if the program has not incurred in problems, the `X_destination` is taken, by choosing from the second letter to the end of the array. The procedure used to extrapolate the `X_destination` is repeated also for the `Y_destination` and if the number of elements of the array is 4 it will be executed also for the `Speed`. If the array elements are 3 the speed value will be set as Boolean false, to communicate to the next function that the speed is not present in order to set the velocity as the previous. In conclusion, are computed the X and Y displacements variables (`X_displacement` and `Y_displacement`) by doing the subtraction between the destination and current position variables (`X_last_position` and `Y_last_position`). Next are updated the last position variable adding the last position on the arrival point of the movement.

```
if Vector_G_code_line[0]=='G00':
      str=Vector_G_code_line[1]
      Residual_string=str[:1]
     if Residual_string=='X':
           X_destination=str[1:]
          str=Vector_G_code_line[2]
          Residual_string=str[:1]
          if Residual_string=='Y':
                 Y_destination=str[1:]
               if lenght == 4 :
                       str=Vector_G_code_line[3]
                    Residual_string=str[:1]
                    if Residual_string=='F':
                             Speed=str[1:]
                    else :
                        Speed = False
          X_displacement = float(X_destination) - X_last_position
          Y_displacement = float(Y_destination) - Y_last_position
          X_last_position = X_displacement + X_last_position
          Y_last_position = Y_displacement + Y_last_position
```

Figure 42: Raspberry extrapolation of G00

Since the command G00 represents a rapid positioning, the machine must only perform a movement without actuating the cutting nozzle. In the Figure 43 there is the creation of the process related to the command G00. We need to create a process to parallelize the execution of the movement of the end effector and of the visualize map. In fact, when the process starts with the proper method the command is given to the function `Visualize_map`, that is used to display on screen the movement of the end effector. Next using the method `join` we will be waiting for the end of the process, so, the movement.

```
G00 = multiprocessing.Process(target = Command_motor_logic,
args=[ X_displacement, Y_displacement, Speed])
G00.start()
Visualize_map(X_displacement, Y_displacement)
G00.join()
```

Figure 43: Raspberry G00 movement

On the other hand, if the command is G01 is imposed the cut must be performed. The decisional process done in order to extrapolate the destination point and the velocity is the same of G00. As we can see in Figure 44 the difference lies in the setting of the value of a GPIO pin to control the water jet. After the creation of the relative process G01 is assigning HIGH (3.3V) is assigned to the pin number 18, and the water jet starts. At the end of the process the GPIO number 18 is reset with the value LOW (0V) and the water jet is turned off.

```
G01 = multiprocessing.Process(target = Command_motor_logic,
args=[ X_displacement, Y_displacement, Speed])
G01.start()
GPIO.output(18, GPIO.HIGH)
Visualize_map(X_displacement, Y_displacement)
G01.join()
GPIO.output(18, GPIO.LOW)
```

Figure 44 Raspberry G01 movement

4.2.4.2 Interpretation of G02

The CNC machines can perform circular path that corresponds to the command G02. The path of movement is implemented by using the linear interpolation. The circle is divided into small straight lines and applying them one by one the circular path is performed. Our code can perform only a quarter of circle. The interpretation of the command G02, that symbolizes a circle, is similar to the G00 and G01 commands. Before to show what that we have implemented, we must explain how the command is written and the meaning of each term.

A G02 command contains:

- The header, G02
- The X and Y destination point

Or the letter J or I, in case of a circle on 2D dimension

The header and the destination point have the same meaning of the commands G00 and G01. The letter J and I symbolize the offset between the starting point and the center of the circle. In particular, the letter I is the offset from the starting point to the center along the X axis and the letter J is the offset from the starting point to the center along the Y axis. For instance, if our G-code command is: G02 X-1.5 Y1.5 J1.5 F50 and the last position (that is the starting point) is X = 0 and Y = 0, the command will mean that the circle ends in point X=-1.5 Y=1.5 and its center is with an Y positive offset of 1.5 from the starting point, that means that it will be in the point X=0 Y=1.5.

Since the limitation of the ability of performing only quarter of circle, the G02 command can be composed only by one offset on one axis with positive or negative sign. We have 4 possibilities, +I, -I, +J and -J. For example, we could decodify the commands that impose a quarter of circle as:

- G02 X-1.5 Y1.5 J1.5
- G02 X0 Y5 I1.5
- G02 X4.5 Y3.5 J-1.5
- G02 X3 Y0 I-1.5

In this case there is only one letter between ‘J’ or ‘I’ and using the geometry, the offset is equal to the radius in every case. In the Figure 45 is show the desired path of the first command.

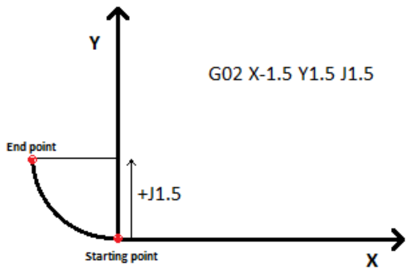


Figure 45: G02 example

On the other hand, cannot be decodify a command with offset on both axis written as:

- G02 X-0.75 Y0.75 I 0.75 J0.75

After having understood that the G-code can contain G02 commands with only one offset, we will implement the decisional process that is represented in Figure 46 in order to extrapolate on which axis the offset is, and which sign has. When the decisional process starts, firstly we have to find out what is the letter, if is ‘I’ or ‘J’. Secondly, we have to figure out which sign has the offset. According to the letter and the sign the decisional process leads to 4 different functions of interpolation. These functions are responsible for the implementation of linear interpolation by dividing the circle path in small lines and applying them one by one. These 4 functions will be explained later.

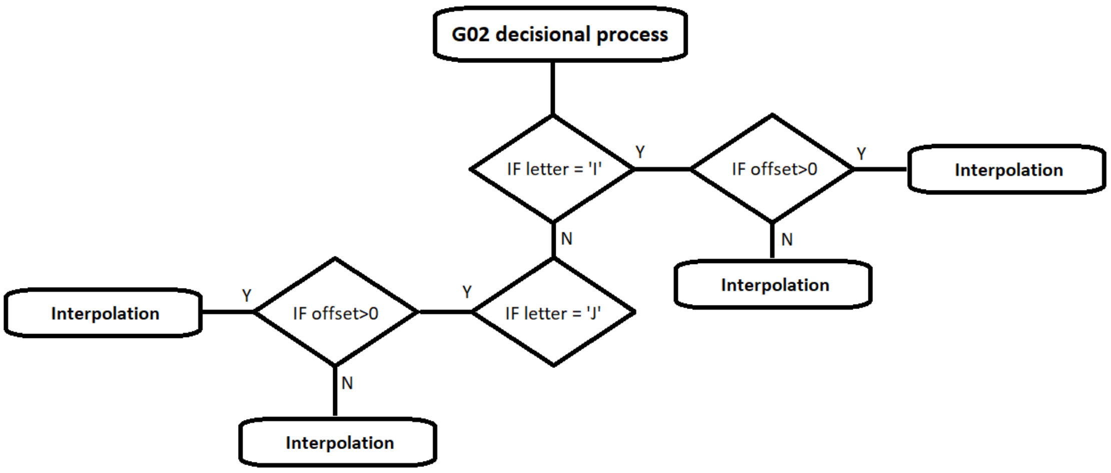


Figure 46: G02 decisional process for extrapolation

For example, in the Figure 47 represent the decisional process chosen in order to verify if the letter relative to the command G02 is the ‘I’. Clearly we have been implemented also the decisional process of the letter ‘J’. As we can see, the process is similar to the G00 and G01 ones, with the difference that is present the check of presence of the letter ‘I’. If this condition is not verified, we will be in presence of an offset on Y axis, the letter ‘J’. The radius corresponds to the numerical value that is next to the letter ‘I’ is the radius. With this code we also have extrapolated the end destination. For instance, if our command is G02 X0 Y5 I1.5, the final destination is the point X=0 and Y=5.

```
if Vector_G_code_line[0]=='G02':
      str=Vector_G_code_line[1]
      Residual_string=str[:1]
     if Residual_string=='X':
            X_destination=str[1:]
          str=Vector_G_code_line[2]
          Residual_string=str[:1]
          if Residual_string=='Y':
                 Y_destination=str[1:]
               if Residual_string == 'I':
                       Radius = str[1:]
                       if lenght == 5 :
                             str=Vector_G_code_line[3]
                             Residual_string=str[:1]
                             if Residual_string=='F':
                                   Speed=str[1:]
                             else :
                                   Speed = False
```

Figure 47: G02 extrapolation

At this point, we have extrapolated the on which axis the offset is and which sign has and the end destination. As what have been said before, there are 4 different case that are:

- +I (positive value of offset on X axis)
- -J (negative value of offset on X axis)
- +J (positive value of offset on Y axis)
- -J (negative value of offset on Y axis)

Every case leads to a different interpolation function. The interpolation function figures out where the center is and generates 4 intermediate points that divide the circle into 5 straight lines. Then, the function, performs 5 movement in order to execute the desired path. Hereafter there is an overview on the linear interpolation of each function has been implemented and in particular, is explained how has been develop the function relative to the offset +J.

4.2.4.2.1 Positive offset on Y axis (+J)

Considering a positive offset on the Y axis, that means a +J on the G-code command the imposed quarter of circle is oriented as is shown in Figure 48. This piece of circle is the translation of the follow G-code command: G02 X-1.5 Y1.5 J1.5, where X = -1.5 Y = 1.5 is the end point and J1.5 means that it has a radius of 1.5.

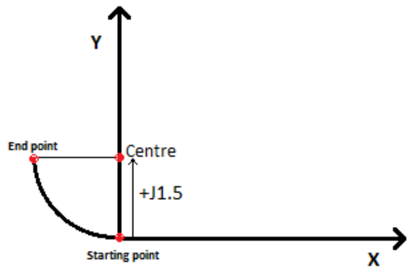


Figure 48: G02 positive offset on Y axis

Since the geometry, the coordinate of the center is obtained as:

- For the X coordinate is the same since there is no offset.
- For the Y coordinate adding the last position to the radius

Then, a for loop is used in order to create the interpolate lines as we can see in the Figure 49. Firstly, the sine and cosine vectors are set used to compute the intermediate points. These vectors are composed by 4 elements, that are the cosine and sine of the angle 18, 36, 54 and 72 degrees. These values are used to create 4 intermediate points that lead to 5 lines; starting from the last position point passing through those 4 points and getting the point number 4 (see Figure 49). In short, the for loop computes the point 1, 2, 3 and 4 and performs the straight displacement from the starting point to the point 1 and so on in order to get to point number 4. Until now, we perform the 1, 2, 3 and 4 lines, we are in the point number 4, and we have not completed the path since it finishes in the end point. In order to complete the path, when the for loop is ended, the displacement will be performed from the point number 4 to the end point

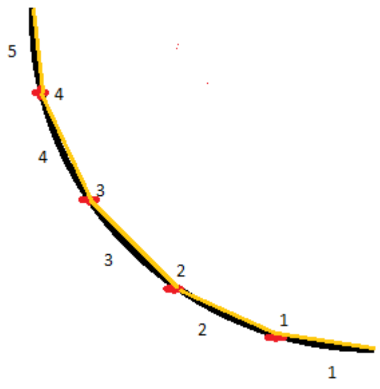


Figure 49: Example of linear interpolation

As we can see in Figure 50, the fictitious destination (the intermediate point) on the X axis, is the subtraction between the X coordinate of the center and the multiplying of the radius by the value of cosine selected by the iteration counter. On the contrary, the fictitious destination on the Y axis is the sum between the Y coordinate of the center and the multiplying of the radius by the value of sine selected by the iteration counter. The fictitious point, that is used to compute the displacement that the machine has to perform, is computed at every iteration. The for loop, runs the code inside 4 times by computing 4 intermediate points. Finally, at each iteration, the last positions variables are updated, and the motor logic function is called to perform the straight movements.

```
if float(Radius) > 0:
      CENTRE_X = X_last_position
     CENTRE_Y = Y_last_position + float(Radius)
      COSINE = [0.95,0.81,0.59,0.31]
      SINE = [0.31,0.59,0.81,0.95]
      for i in range (4) :
      FICTITIOUS_X = CENTRE_X - float(Radius)* COSINE[i]
      FICTITIOUS_Y = CENTRE_Y - float(Radius)* SINE[i]

      X_displacement = FICTITIOUS_X - X_last_position
      Y_displacement = FICTITIOUS_Y - Y_last_position

      X_last_position = X_displacement + X_last_position
      Y_last_position = Y_displacement + Y_last_position

            G02 = multiprocessing.Process(target =
            Command_motor_logic,args=[X_displacement,Y_displacement
            ,Speed])

            G02.start()

            Visualize_map(X_displacement,Y_displacement)
      G02.join()
```

Figure 50: G02 positive offset on Y axis, linear interpolation

When the for loop is ended, we are in the point number 4, and we need to perform the last displacement in order to finish the circle path. Then, the control is given to the code (show in Figure 51) that computes the displacement between the end position of the circle and the last position (the point number 4). Calling the function Command motor logic, the last displacement is performed completing the quarter of circle.

```
X_displacement = float(X_destination) - X_last_position
Y_displacement = float(Y_destination) - Y_last_position

X_last_position = X_displacement + X_last_position
Y_last_position = Y_displacement + Y_last_position

G02 = multiprocessing.Process(target = Command_motor_logic,
args=[X_displacement, Y_displacement, Speed])
G02.start()

Visualize_map(X_displacement,Y_displacement)

G02.join()
```

Figure 51: G02 positive offset on Y axis, last movement

4.2.4.2.2 Negative offset on Y axis (-J)

A negative offset on the Y axis is carried out by a quarter of circle oriented in the way that is shown in Figure 52.

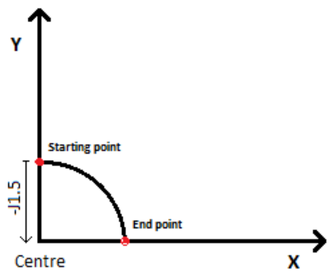


Figure 52: G02 negative offset on Y axis

In the Figure 53 is shown the code used to compute the intermediate point and for the execution of the displacements (the linear interpolation). The radius has the negative sign, and the center is computed:

- On X considering the center coordinate as the X last position.
- On Y adding the radius (that is negative) to the Y last position.

Then, inside the for loop, the coordinate of the intermediate point are computed at each iteration as:

- The fictitious coordinate on the X axis is the center minus the multiplying of the radius (that is negative) times by the element of sine vector in the position equal to the counter of the for loop.
- The fictitious coordinate on the Y axis is the center minus the multiplying of the radius times the element of cosine vector with the same logic that was mentioned before.

```
if float(Radius) < 0:
      CENTRE_X = X_last_position
     CENTRE_Y = Y_last_position + float(Radius)
      COSINE = [0.95,0.81,0.59,0.31]
      SINE = [0.31,0.59,0.81,0.95]
      for i in range (4) :
      FICTITIOUS_X = CENTRE_X - float(Radius)* COSINE[i]
      FICTITIOUS_Y = CENTRE_Y - float(Radius)* SINE[i]

      X_displacement = FICTITIOUS_X - X_last_position
      Y_displacement = FICTITIOUS_Y - Y_last_position

      X_last_position = X_displacement + X_last_position
      Y_last_position = Y_displacement + Y_last_position

            G02 = multiprocessing.Process(target =
            Command_motor_logic,args=[X_displacement,Y_displacemen
            t,Speed])

            G02.start()

            Visualize_map(X_displacement,Y_displacement)
      G02.join()
```

Figure 53: G02 negative offset on Y axis, linear interpolation

Finally, is perform the last displacement as for the letter +J

4.2.4.2.3 Positive offset on X axis (+I)

A positive offset on the X axis is carried out by a quarter of circle oriented in the way that is shown in the Figure 54.

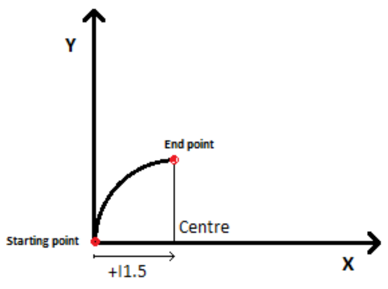


Figure 54: G02 positive offset on X axis

In the Figure 55 is shown the code used to compute the intermediate point and to execute the displacement. The center is computed:

- On X computing the center coordinate adding the radius to the X last position.
- On Y considering the center coordinate as the Y last position.

Then, inside the for loop the coordinate of the intermediate point are computed at each iteration.

- The fictitious coordinate on the X axis is the center minus the multiplying of the radius times the element of cosine vector
- The fictitious coordinate on the Y axis is the center plus the multiplying of the radius times the element of sine vector.

```
if float(Radius) > 0:
      CENTRE_X = X_last_position + float(Radius)
     CENTRE_Y = Y_last_position
      COSINE = [0.95,0.81,0.59,0.31]
      SINE = [0.31,0.59,0.81,0.95]
      for i in range (4) :
      FICTITIOUS_X = CENTRE_X - float(Radius)* COSINE[i]
      FICTITIOUS_Y = CENTRE_Y + float(Radius)* SINE[i]

      X_displacement = FICTITIOUS_X - X_last_position
      Y_displacement = FICTITIOUS_Y - Y_last_position

      X_last_position = X_displacement + X_last_position
      Y_last_position = Y_displacement + Y_last_position

            G02 = multiprocessing.Process(target =
            Command_motor_logic,args=[X_displacement,Y_displacement
            ,Speed])

            G02.start()

            Visualize_map(X_displacement,Y_displacement)
      G02.join()
```

Figure 55: G02 positive offset on X axis, linear interpolation

Finally, the last displacement, as for the letter +J is performed.

4.2.4.2.4 Negative offset on X axis (-I)

A negative offset on the X axis is carried out by a quarter of circle oriented in the way that is shown in the Figure 56.

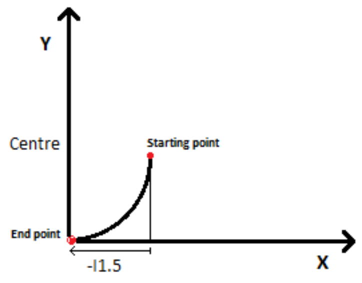


Figure 56: G02 negative offset on X axis

In the Figure 57 is shown the code used to compute the intermediate point and to the execution of the displacement. The center is computed:

- On X computing the center coordinate adding the radius (that is negative) to the X last position.
- On Y considering the center coordinate as the Y last position.

Then, inside the for loop, at each iteration, are computed the coordinate of the intermediate point.

- The fictitious coordinate on the X axis is the center minus the multiplying of the radius (that is negative) times the element of cosine vector

- The fictitious coordinate on the Y axis is the center plus the multiplying of the radius (that is negative) times the element of sine vector.

```
if float(Radius) < 0:
      CENTRE_X = X_last_position + float(Radius)
     CENTRE_Y = Y_last_position
      COSINE = [0.95,0.81,0.59,0.31]
      SINE = [0.31,0.59,0.81,0.95]
      for i in range (4) :
      FICTITIOUS_X = CENTRE_X - float(Radius)* COSINE[i]
      FICTITIOUS_Y = CENTRE_Y + float(Radius)* SINE[i]

      X_displacement = FICTITIOUS_X - X_last_position
      Y_displacement = FICTITIOUS_Y - Y_last_position

      X_last_position = X_displacement + X_last_position
      Y_last_position = Y_displacement + Y_last_position

            G02 = multiprocessing.Process(target =
            Command_motor_logic,args=[X_displacement,Y_displacemen
            t,Speed])

            G02.start()

            Visualize_map(X_displacement,Y_displacement)
      G02.join()
```

Figure 57: G02 negative offset on X axis, linear interpolation

Finally, is perform the last displacement as for the letter +J

4.2.4.3 Command motor logic

The command motor logic function takes as input the `X_displacement`, Y_ `Y_displacement` and the speed. It gives as output two strings that each one of them is sent to the proper controller (Arduino) to actuate the motors. The string are constructed using the correct conversion logic imposed by the unconventional pose of the motors. The inputs are used to compute the necessary step and direction on both motors in order to achieve the imposed destination point with the desired velocity. The string is constructed summing 3 sub-strings that are:

- The direction: forward or backward
- The absolute number of steps
- The velocity

These 3 sub-strings are computed in the in the function and are merged in order to be sent to the Arduino controller

In the Figure 58 is shown the check of the presence of the velocity and the computation of the motors direction through analyzing the sign of the compute steps. To check if the velocity is given, a control of what contain the variable `Set_speed` is performed. According to what has been developed in the G-code translator section, in the variable Set_speed is stored the value of velocity that is extrapolated extrapolate from the G-code text file. If the value is a number, a check is performed in order to establish if it is less than the maximum velocity that the machine can achieve (500 mm/s). If is greater, the velocity is set as the maximum value and is stored in the variable Speed , else the given value is saved in the previous variable. Unless if the value was set as Boolean false, that means that is not given and the value Speed is set as the previous velocity recorder. Then, by using the motor steps-displacement relation and the relation between the linear movement and the angular movement (chapter 2.1.4) is computed the number of steps of each motor and is adapted in order to fit the G-code displacement to the real displacement multiplying be the constant Constant_of_proportionality. According to what we have demonstrated in the chapter 2.2.6 the step-displacement relation is 1.05mm/step. That means that the constant Constant_of_proportionality has a value of 0.952 step/mm. The signs of the imposed steps are used to compute the motors directions. So, we assign the direction analyzing the sign of the motor steps. If the variables are positive the forward direction will be assigned, on the contrary, if they are the backward direction negative are assigned.

```
def Command_motor_logic (X_displacement, Y_displacement,
Set_speed):

      if Set_speed == False:
            Speed = Previus_speed
      if Set_speed < 500
            Speed = Set_speed
            Previus_speed = Set_speed
      else:
            Speed = 500
            Previus_speed = 500


      STEP_V = Constant_of_proportionality*(Y_displacement +
      X_displacement)

      STEP_U = Constant_of_proportionality*(X_displacement)

      if STEP_U > 0:
            Str_U_direction='forward'
      if STEP_U < 0:
            Str_U_direction='backward'
      if STEP_V > 0:
            Str_V_direction='forward'
      if STEP_V < 0:
            Str_V_direction='backward'
```

Figure 58: Raspberry command motor logic

At this point we might incur in some problem. If we have to perform a movement where both motors are involved, applying a different number of steps to each motor with the same velocity will occur the end of the movement before in one motor rather than in the other. For instance, if our motor logic function imposes with the same direction of 100 steps to the motor U and 1000 steps to motor V with the same velocity of 100 steps in a second and the movement starts at the same time instant will occur that the motor U will conclude the movement after 1 second and the motor V after 10 seconds. This wrong behavior of the motors will lead to wrong path. In fact, if we imposed a G01 movements (a straight line), the final movement will be two lines with different angular coefficient and the change of inclination occurs in the point where one motor stops. To overcome the problem, we need to weight the velocity according to the different number of steps, giving more velocity to the longer movement (that means more steps) and less velocity to the shorter movement (that means less steps), to have the synchrony of the start and the stop of motors motion. In short, we have that the velocity `Set_speed` (velocity imposed by the G-code file) is the magnitude velocity that is composed by Vx and Vy as we can see in the Figure 59.

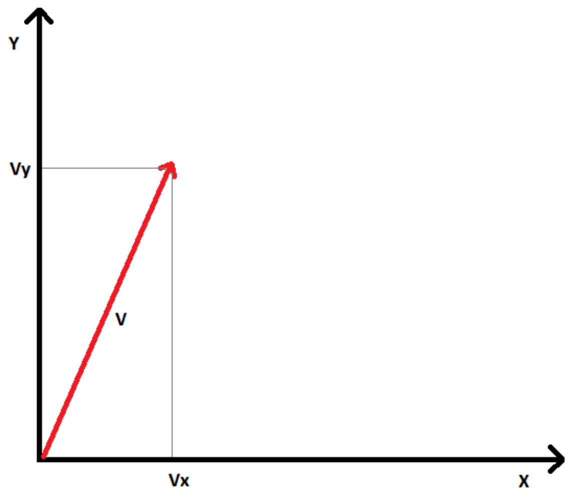


Figure 59: Velocity magnitude

So, the velocity of each motor is computed as follow. Our constraints are:

- The competition time of movement on both axes must be the SAME.
- The magnitude of the velocities of both axes must be EQUAL to the velocity set by G-code

These requirements are translated in equation. The first leads to:

$$\frac{\mathrm{d}_y}{v_y} = \frac{\mathrm{d}_x}{v_x}$$

Doing this computation, we set as equal competition time of movements. The second lead to:

$$v_{max} = \sqrt{v_x^2 + v_y^2}$$

Making these computations, we make sure that the magnitude of Vx and Vy is equal to velocity imposed by the G-code.

These two equations, lead to a two variable system that the unknown Vx and Vy are solved, and are:

- $v_x = \sqrt{\frac{v_{max}}{1+\left(\frac{d_y}{d_x}\right)^2}}$
- $v_y = \sqrt{\frac{v_{max}}{1+\left(\frac{d_y}{d_x}\right)^2}} \cdot \left(\frac{d_y}{d_x}\right)$

Finally, applying the derivate of step-displacement relation that is remember:

- $h \cdot \theta_V = \mathrm{d}_x - \mathrm{d}_y$
- $k \cdot \theta_U = \mathrm{d}_y$

And deriving we obtain:

- $h \cdot \dot{\theta}_V = v_x - v_y$
- $k \cdot \dot{\theta}_U = v_y$

So, in conclusion the angular velocity for each motor, that has been implemented in the code, is:

- $\dot{\theta}_V = \frac{\sqrt{\frac{v_{max}}{1+\left(\frac{\mathrm{d}_y}{\mathrm{d}_x}\right)^2}} - \sqrt{\frac{v_{max}}{1+\left(\frac{\mathrm{d}_y}{\mathrm{d}_x}\right)^2}} \cdot \left(\frac{\mathrm{d}_y}{\mathrm{d}_x}\right)}{h}$

- $\dot{\theta}_U = \frac{\sqrt{\frac{v_{max}}{1+\left(\frac{\mathrm{d}_y}{\mathrm{d}_x}\right)^2}} \cdot \left(\frac{\mathrm{d}_y}{\mathrm{d}_x}\right)}{k}$

Having compute the weight velocity for each motor, we have to analyze the limit cases, when is request a movement only on the x axis or y axis. In these two cases, the computation done by the presented formula leads to a wrong result. In fact, in case of dx equal to 0 the ratio dy/dx goes to infinity, then the result will be invalidated. In order to achieve the correct velocity value, an analysis of the displacement must be made. In case of dy = 0, a movement only in the x axis, are actuate both motors with the same velocity, so is necessary to set their velocity equal to the G-code ones. In case of dx = 0, a movement only in the y axis, is actuate only the motor U so in necessary to set its velocity to extrapolate ones.

In case of a movement more in one axis then in the other, for instance X = 100 and Y = 0.1 no problem arises, since the internal variable inside the formula goes to high value but rest bounded and the computer can manage the numbers.

In the Figure 60 is shown the computation of the velocity in the general case and the decisional process in the limit cases.

```
dy=Y_displacement
dx=X_displacement

if dy == 0:
      Speed_V = - Speed
      Speed_U = Speed

if dx == 0:
      Speed_V = 0
      Speed_U = Speed

Speed_V = (sqrt(Speed/(1+(dy/dx)**2)) -
sqrt(Speed/(1+(dy/dx)**2))*(dy/dx))
Speed_U = (sqrt(Speed/(1+(dy/dx)**2))*(dy/dx))
```

Figure 60: Raspberry velocity computation

We have computed the direction, the needed step and the velocity. Then the string is constructed by appending the value in this order: direction, steps, and velocity as it is shown in Figure 61. Then, the strings are sent to the function that is able to create the connection with Arduino, using the library multiprocessing to parallelize the execution.

```
string_tot_V =
'<'+Str_V_direction+','+str(abs(STEP_V))+','+str(abs(Speed_V))+'>'
string_tot_U =
'<'+Str_U_direction+','+str(abs(STEP_U))+','+str(abs(Speed_U))+'>'

M_U = multiprocessing.Process(target = Movement_motor_U_G_code
,args=[string_tot_U,Str_U_direction,STEP_U,Speed_U])
M_V = multiprocessing.Process(target = Movement_motor_V_G_code
,args=[string_tot_V,Str_V_direction,STEP_V,Speed_V])

M_U.start()
M_V.start()
```

Figure 61: Raspberry string construction

4.2.4.4 Movement motor U and movement motor V

After having seen in the previous chapter (4.2.2) which function we need to have a reliable communication between Raspberry and Arduino, we can talk in this chapter about how the string is sent Arduino.
Considering the string used to communicate a straight-line movement from G-code (for instance <Forward,1000,50), the dedicated function is shown in the Figure 62. We can appreciate how the function follow the presented hand shacking protocol. Firstly, the serial port is opened, then we will be waiting for the connection signal from Arduino. Next, the communication is considered reliable, and we can send the string in order to allow Arduino to start the motion. In the end Arduino will send the end motion symbol "#" in order to ensure the competition of the motion. Finally, the serial port will be close.

```
def Movement_motor_U_G_code(String,Str_U_direction,STEP_U,Speed_U):
    serial_U = serial.Serial('/dev/ttyUSB0',9600)
    waitForArduino()
    Send_to_Arduino(String)
    msg = ""
    while msg.find("#") == -1:
        msg = Receive_from_Arduino()
    serial_U.close()
    if counter=!steps
        error = steps-counter
        string_residue_U =
'<'+Str_U_direction+','+str(abs(error))+','+str(abs(Speed_U))+'>'
        Movement_motor_U_G_code(string_residue_U)
```

Figure 62: Raspberry G-Code motor U movement

4.2.5 Homing and base selection

4.2.5.1 Base selection

The home position is the point into the operative space of the machine that is set as starting point (X=0 and Y=0). Every G-code file refers to it in order to perform the cut, in other words is the absolute reference. In the application, the user can set the home position at his discretion pressing the button created in the Figure 63 with the name `Select point as base`. Pressing this button, the function `Select_as_base` is called, and the home position is set as the current position. In fact, the variable X and Y are the current position of the end effector and the variables X_home and Y_home are the home coordinates so make the equality of these two variables we set the base as the current position. So, by pushing the button, the home variables are set as the current position. The procedure that must done to set the base is extremely easy: the user has to move the end effector with the joystick in the desired position and he has to push the button to set the new base. The machine has its own home point set as default, but to increase the flexibility we have implemented the presented function to allow the user to set the home position as he wishes.

```
def Select_as_base():
    global X_home
    global Y_home
    global X
    global Y
    X_home = X
    Y_home = Y

Base_selection = tk.Button(text="Select point as base",command=
Select_as_base, wraplength = 100 )

Base_selection.place(height=100,width=100, x=475, y=25)
```

Figure 63: Raspberry base selection

4.2.5.2 Homing procedure

Having regard to the precision required in performing the homing function and the possibility to have a positional error on the end frame, the homing procedure is implemented using the absolute position given by the limit switch. Furthermore, due to the errors, the application position variable, might not be much so, the limit switches absolute reference is required. The absolute reference is used by moving the end effector in a corner, where we have the activation of both limit switches. The procedure is programmed calibrating one axis at a time, zeroing the position. Firstly, the calibration on the X axis is done, later we will have the calibration on the Y axis, and finally a movement from the corner to the home position is perform. Doing this procedure one axis at time extends the time required but increase the precision. The calibration procedure for both axes consist in moving the end effector with high velocity along one axis until the dedicated limit switch is reached. Then we perform a constant distance displacement in a direction away from the limit switch in order to release it. So, with a low speed we reach again the limit switch. Finally, making few steps away the limit switch we release it (to not operate the machine with an alert message on the limit switch insert). This procedure leads to a perfect calibration of the axes, and it will be repeated for the X and Y.

As we can see in the Figure 64 and Figure 65 are shown the procedure of calibration for the X axis in (Figure 64) and the Y axes (Figure 65). Firstly, in the Figure 64 is created the button widget linked to the function "`homing_procedure`" is created. Pressing the above-mentioned button, we can start the homing procedure, that can be stopped only for emergency pressing the dedicated external button. Then, using the library multiprocessing to parallelize the calling of function, the functions relative to the movement "`Move_motor_U_G_code`" and "`Move_motor_V_G_code`" will be called. These functions, perform a movement in the X direction until the limit switches indicate the reach of them. After, using the method .joint, the script will wait the termination of both processes. Then, the movement related to the release of the limit switches is performed and reaching again with extremely low velocity. Finally, a movement for the release of them with the same velocity of the previous movement is performed.

The symbol H, P, and Fare sent to Arduino, where H corresponds to the high velocity limit switches meet function, P corresponds to low velocity that releases a reach again limit switches and F corresponds to low velocity reach limit switches. These symbols are sent to both Arduino controllers since the movement on the X axis is carried out by the rotations of both motors. In fact, seeing the symbols listed in Table 1 and Table 2 in the chapter 4.2.2 the homing procedure symbols on the X axis are set on both Arduino.

In the Figure 65 the same procedure for the calibration of Y axis is implemented. The symbol h, p and f are sent to Arduino, in order to avoid repetition, corresponds to the same function of the X axis. Finally, are set the internal variable of position as the home position constant and, are set the counters variables for the map as linear combination of the home position constants considering the poses of the motors, and the movement from the corner to the home position is performed.

```
HOMING = tk.Button(text="Homing", command = homing_procedure)
HOMING.place(height=100,width=100, x=50, y=25)

def homing_procedure():
    global X_last_position
    global Y_last_position
    global X
    global Y
    global counter_u
    global counter_v

    homing_X_U1 = multiprocessing.Process(target =
Move_motor_U_G_code, args=["H"])
    homing_X_V1 = multiprocessing.Process(target =
Move_motor_V_G_code, args=["H"])
    homing_X_U1.start()
    homing_X_V1.start()
    homing_X_U1.join()
    homing_X_V1.join()

    homing_X_U2 = multiprocessing.Process(target =
Move_motor_U_G_code, args=['P'])
    homing_X_V2 = multiprocessing.Process(target =
Move_motor_V_G_code, args=['P'])
    homing_X_U2.start()
    homing_X_V2.start()
    homing_X_U2.join()
    homing_X_V2.join()

    homing_X_U3 = multiprocessing.Process(target =
Move_motor_U_G_code, args=['F'])
    homing_X_V3 = multiprocessing.Process(target =
Move_motor_V_G_code, args=['F'])
    homing_X_U3.start()
    homing_X_V3.start()
    homing_X_U3.join()
    homing_X_V3.join()
```

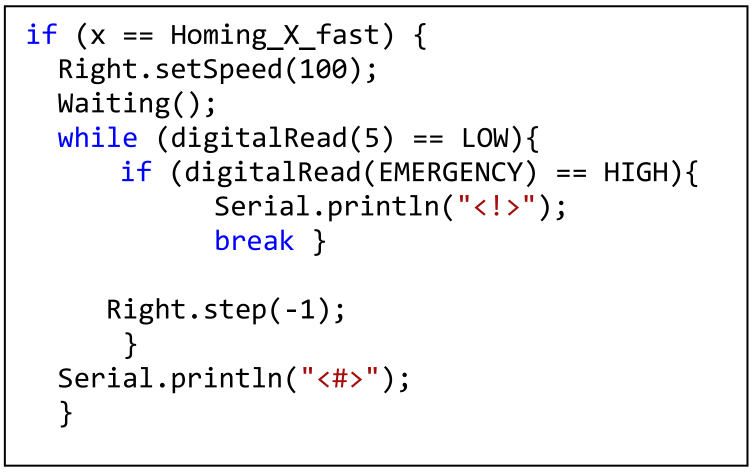
Figure 64: Raspberry X axis homing procedure

```
homing_U4 = multiprocessing.Process(target = Move_motor_U_G_code,
args=['h'])
homing_U4.start()
homing_U4.join()

homing_U5 = multiprocessing.Process(target = Move_motor_V_G_code,
args=['p'])
homing_U5.start()
homing_U5.join()

homing_U6 = multiprocessing.Process(target = Move_motor_V_G_code,
args=['f'])
homing_U5.start()
homing_U5.join()

X = X_home
Y = Y_home

counter_u = X_home + Y_home
counter_v = Y_home

homing_procedure_movement = multiprocessing.Process(target =
Command_motor_logic,args=[X_home, Y_home, 50])

homing_procedure_movement.start()

Visualize_map(X_home , Y_home)

homing_procedure_movement.join()
```

Figure 65: Raspberry Y axis homing procedure

In the Figure 66, Figure 67, and Figure 68 we can see the code implemented in the Arduino controller of the motor U in order to calibrate the axis X and this code is also implemented in the motor controller of the motor V. The code related to the axis Y is neglected, due to the similarity to the code regarding the calibration of the X axis and the unique difference is that the code for is implemented only in the controller of the motor U since the movement of the Y axis is carried out only by its movement.

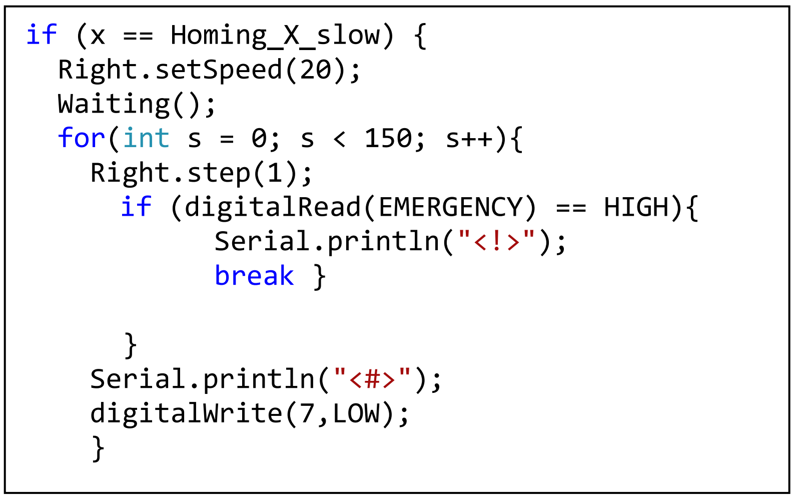
In the image Figure 66 we can see the procedure to meet the limit switch on the X axis at high velocity. In particular the symbol H in Arduino is sent and the velocity at 100 steps in a second is set. Then, the waiting function used to synchronize the starting of both motors is called, since the serial communication is made one by one, could happen that one Arduino receives the symbol before and start before the other. Next, a while loop is performed until the achievement of the limit switch. At each iteration the status of the emergency push button is checked, and a motor step is performed. In case of an emergency status the while loop is break and the symbol “!” is sent to Raspberry and means the stop of the application. When the limit switch is reached, and we have the breaking of the while loop and the symbol “#” is sent from Arduino to PC that means the end of the function and the PC can proceed with next command.



```
if (x == Homing_X_fast) {
  Right.setSpeed(100);
  Waiting();
  while (digitalRead(5) == LOW){
      if (digitalRead(EMERGENCY) == HIGH){
            Serial.println("<!>");
            break }

     Right.step(-1);
      }
  Serial.println("<#>");
  }
```

Figure 66: Arduino fast homing movement

Then, as shown in the Figure 67, we will move slowly away from the limit switch, freeing it. The velocity is set as 20 and a "for" cycle of 150 steps is performed.



```
if (x == Homing_X_slow) {
  Right.setSpeed(20);
  Waiting();
  for(int s = 0; s < 150; s++){
    Right.step(1);
      if (digitalRead(EMERGENCY) == HIGH){
            Serial.println("<!>");
            break }

      }
    Serial.println("<#>");
    digitalWrite(7,LOW);
    }
```

Figure 67: Arduino slow homing movement

Then, we reach the limit switch again with extremely low velocity (10 steps in a second) to enhance the precision of calibration with a while loop with break statement of the limit switch activation. Finally, a for loop to move away from the limit switch is performed freeing it, having the calibration of the axis.

```
if (x == Homing_X_find) {
  Right.setSpeed(10);
  Waiting();
  digitalWrite(7,LOW);
  while (digitalRead(5) == LOW){
      if (digitalRead(EMERGENCY) == HIGH){
            Serial.println("<!>");
            break }

    Right.step(-1);
      }
  Waiting();
  for(int s = 0; s < 10; s++){
      if (digitalRead(EMERGENCY) == HIGH){
            Serial.println("<!>");
            break }

     Right.step(1);
         }
  Serial.println("<#>");
  digitalWrite(7,LOW);
     }
```

Figure 68: Arduino find limit switch movement

4.2.6 Cutting map and limit switch reader

As analyzed in the requirement chapter, the map must be in real time (with less delay as possible) and it must have as input the encoder signal and the limit switch signal. Also, the map must be displayed on the graphic interface with a higher refresh rate as possible. For the construction of the map, a matrix with a dimension proportional to the working area have been used. The matrix is assumed as a square and has the dimension of 400x400 elements, is composed by all zero and is called map matrix. Then, in order to represent the end effector, we use another matrix composed by all ones elements with dimension of 5x5, that is named end effector matrix. With the aim of achieving the representation of the position are mesh these 2 matrices. The end effector matrix is positioned inside the map matrix where the end effector is. The position of placement is obtained using the encoder signal. Finally, we have a matrix that is composed with all zeros except a square of ones where the end effector is. The obtained matrix is updated cyclically using a while loop, moving the end effector matrix according to the encoder signals at each iteration.

The developing has taken into consideration if a cut has been performed or a simple displacement has been made (joystick or rapid positioning of G-code, G00). As it has been explained in the requirements analysis, if is performed a cut the map must be colored in the path of the end effector and symbolizes the cut on the target. On the contrary, in case of a simple movement, the end effector must slide on the map without coloring the map. This constraint has been fulfilled:

- In case of a cut movement, the updated map matrix is saved at each while iteration and is reupdated in the next iteration. So, the map matrix with inside the end effector matrix is saved and when the encoder signal state to move the end effector matrix, this last is repositioning in the matrix before. As we can see in the Figure 69 the path is created the with the move of the end effector. The map matrix is reset only at the end of a G-code program

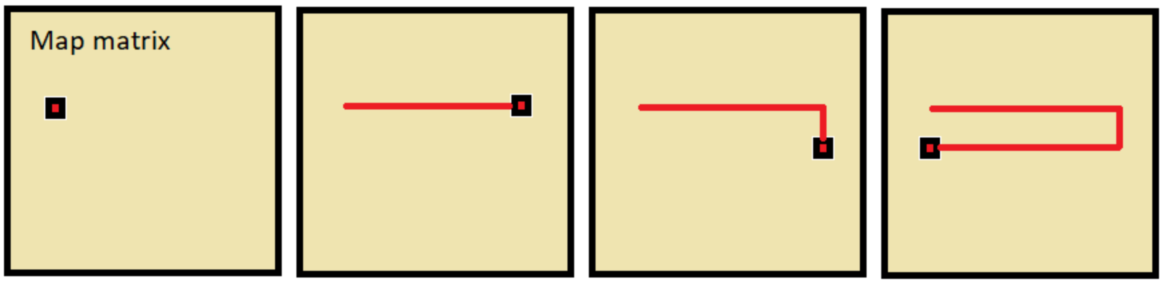


Figure 69: Example of cutting map construction

- In case of a simple movement, the map matrix is reset at each while iteration. So the end effector matrix is moved inside the map matrix and we can see how it work in Figure 70

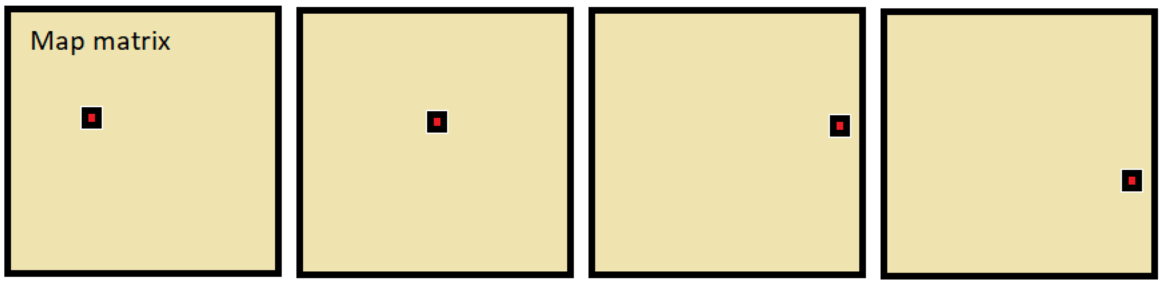


Figure 70: Example of moving map construction

Considering the limit switch signals, they are used to sense if they have been reached. Near the map its status will be displayed, with a symbolic led that assumes the green color if it will be free or the red color if it will be reached. Also, in case of reach of the limit switch, a message will be displayed in order to suggest the perform of the homing procedure, since the misalignment of the real position and the position known by the software. A positive logic is adopted for security reasons in such a way that, the limit switch is considered pushed if it has a low logic level. In case of free limit switch, the output will be a high logic level. The security carried out implementing this configuration is in case of a limit switch fault. In fact, a malfunction on the limit switch occurs, it gives as output the low logic status blocking the machine.

Since the stop of the movement due to the reach of the limit switch is made in the Arduino controller and not in the raspberry PC and since the limit switches checked take a huge computation power, its status is checked every 10 iterations of the while loop. Proceeding in this way we do not incur is security problem since the motor movement is arrested by the Arduino and not by the raspberry. Raspberry uses the limit switch signal only for see its status. As we can see we give more importance to the map construction than the limit switch status.

In the Figure 71 is shown the map's, encoder's pins and variable settings. As illustrated, starting from the GPIO pins setting, we have the pins number 26, 20, 21, 19 that are set as input and correspond respectively to the limit switch of X, -Y, Y, -X axis. The method `.PUD_DOWN` is used to set the internal pull-down resistance of the raspberry, in order to avoid false signals (might carried out by signal fluctuation). In addiction are set the encoder's pins (4 and 10) as input with same pull-down method of the limit switch input. Finally, the map matrices will be set. The `End_effector` matrix is set as 5x5 of ones elements and the `Map` as 400x400 of zero elements.

```
GPIO.setup(26,GPIO.IN, pull_up_down=GPIO.PUD_DOWN)  #X
GPIO.setup(20,GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #-Y
GPIO.setup(21,GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #Y
GPIO.setup(19,GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #-X

GPIO.setup(4,GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #Encoder U
GPIO.setup(10,GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #Encoder V

End_effector = numpy.ones((5,5), dtype=int)
Map = numpy.zeros((400,400), dtype=int)
```

Figure 71: Raspberry pin setting

After having set the needed variables, we must show the function `Visualize_map`. As demonstrated in the Figure 72 the function `Visualize_map` it takes as input `X_displacement` and `Y_displacement` used to compute the direction of movement of the `End_effector`. Next, the following variable are set as global: the matrices of the `Map` and the end effector the `X` and `Y`, that are the position of the end effector known by the machine software, and `counter_v` and `counter_u` are the counter of number of steps that the encoders do. Finally, to count the encoders steps, is saved their actual value in the variable `Encoder_Last_State` (of encoder relative to motor U and V). During the script will be check if this value change, to count the step.

```python
def Visualize_map(X_displacement,Y_displacement):

    global Map
    global End_effector
    global X
    global Y
    global counter_v
    global counter_u

    Encoder_Last_State_U = GPIO.input(4)
    Encoder_Last_State_V = GPIO.input(10)
```

Figure 72: Raspberry map function

Next the while loop that will run until the brake condition of the variable `Finish_movement` is implemented. This brake condition is necessary, in fact when a G-code displacement is called using multiprocessing library, it also called the `Visualize_map` function to visualize the moving end effector. The condition that, when the G-code command ends have to ends also the map, is implemented setting at true logical value the before mentioned variable.

Then, the for loop is implemented in order to give major priority to the map than to the limit switch. As we can see at one iteration of the while loop correspond 10 iterations of the for loop. At the beginning, the state of the encoders is sensed, and they are saved in the variable `Encoder_State` (of encoders U and V). Next, will be check if the variables `Encoder_State` and `Encoder_Last_State` of encoder U are different. If it is true, it is quite clear that the encoder has made a step, since the last is different from the new state, so it changes the relative GPIO input pin.

Then, due to the unconventional pose of motors, the encoders steps do not correspond to a move in a specific axis, but a move in a specific axis correspond to a merge of encoder steps. In fact, as was mentioned in the chapter 2.1.4 between the motors rotations and the end effector displacement exist these relations:

- $h \cdot \theta_V = \mathrm{d}_x - \mathrm{d}_y$
- $k \cdot \theta_U = \mathrm{d}_y$

where θ_V and θ_U are the motors angular displacements, k and h are proportional constant and d_x and d_y are the linear displacements. Since the encoder pins will be not provide a direction signal but only an amount of step proportional to the displacements, we will need a conditional process in order to establish the direction of movement of the end effector. Considering the encoder relative to the motor V, the angular displacement to linear displacement relation is $h \cdot \theta_V = \mathrm{d}_x - \mathrm{d}_y$, and so, checking the sign of the $\mathrm{d}_x - \mathrm{d}_y$ the direction of rotation of motor U will be established. If $\mathrm{d}_x - \mathrm{d}_y$ is positive, we will set the rotation clockwise and the counter of motor U is incremented. If it is negative, the rotation is anti-clockwise, and the counter of motor U is decreased. Considering the encoder V, the angular displacement to linear displacement relation is $k \cdot \theta_U = \mathrm{d}_y$ and conducts to determine the motor V direction checking the sing of d_y. If it positive the motor spin clockwise and the counter of motor V is incremented. If it is negative the motor spin is anti-clockwise, and counter of motor U is decremented. If $d_x - d_y$ or d_y relation give as result the value 0 we do not have the imposition of movement on the relative motor, so the appropriate encoder does not move, and the relative encoder variable is not increase or decrease.

This procedure, shown in Figure 73, is done at every change of the encoder state, then is obtain the changing of the variable `counter_v` and `counter_u` according to the movement of the end effector.

```
    while Finish_movement == False:
        for i in range(10):
            Encoder_State_U = GPIO.input(4)
            Encoder_State_V = GPIO.input(10)

            if Encoder_State_V != Encoder_Last_State_V:
                if Y_displacement - X_displacement > 0:
                    counter_v += 1
                if Y_displacement - X_displacement < 0:
                    counter_v -= 1
                if Y_displacement - X_displacement == 0:
                    counter_v = counter_v

            if Encoder_State_U != Encoder_Last_State_U:
                if X_displacement > 0:
                    counter_u += 1
                if X_displacement < 0:
                    counter_u -= 1
                if X_displacement == 0:
                    counter_u = counter_u
```

Figure 73: Raspberry find direction and encoder pulse, map function

Then the checked status of both encoders is saved in the last state, in order to sense if at the next iteration the state change as is shown in Figure 74. Having determine the variable `counter_v` and `counter_`, has been computed the end effector position using the relation relative to the linear displacement to angular displacement, that are:

- $d_y = k \cdot \theta_U$
- $d_x = k \cdot \theta_U + h \cdot \theta_V$

```
Encoder_Last_State_U = Encoder_State_U
Encoder_Last_State_V = Encoder_State_V
canvas = tk.Canvas(window, width=400,height=400)
canvas.place(x=0,y=320)
Y = abs((counter_v)*Constant_of_proportionality)
X = abs((counter_u-counter_v)*Constant_of_proportionality)
```

Figure 74: Raspberry map placement

Finally, having computed the position of the end effector (X and Y) we merge the `End_effector` matrix inside the `Map` matrix in the calculated point. So, we obtain a matrix with all zeros except in the inside matrix's position of the end effector, in which we have a square of dimension 5x5 of ones. Then, we enhance the square multiplying by 256. This operation is done because the next command converts the matrix into an image composed by a scale of grey where to the elements in the matrix that have zero value is assign black color and to the elements in the matrix that have 256 value is assign white color. So, we have obtained an image that is all black except in the point where the end effector is, there is a white square. This procedure is represented in Figure 75.

```
Map[X:X+5,Y:Y+5]=End_effector
Map=Map*256
img =  ImageTk.PhotoImage(image=Image.fromarray(Map_coloured))
canvas.create_image(20,20, anchor="nw", image=img)
```

Figure 75: Raspberry map creation

After 10 iterations of the for loop, the control is given to the limit switch control. As we can in the Figure 76 is checked if at least one limit switch is reach. If this condition is realized, that means that one of the input goes to low logic level, a graphic message is permanently shown near the map (at the right side of the map). The message suggests the perform of the homing procedure since by touching a limit switch, a misalignment of the real position and the position known by software could occur.

```
if GPIO.input(20) == False or GPIO.input(21) == False or GPIO.input(26)
== False or GPIO.input(19) == False :

      limit_switch_touch = tk.Label( text="WARNING, A LIMIT SWITCH IS
      REACHED, IS SUGGESTED PERFORM HOME PROCEDURE", wraplength =
      200,bg="yellow")
      limit_switch_touch.place(height=120,width=200, x=420, y=360)
```

Figure 76: Raspberry general limit swich checking status

After the general control, the check on each limit switch will be implemented. In the Figure 77 is shown the control of the limit switch positioned on the positive X. Using an if statement, is sense the GPIO 26 and if is false (if has a low logic value) we will have the touch of the switch, according to the security logic set. So, a red label symbolizes the led the shown near the map. On the contrary, if the limit switch is free a green label symbolizing that it has not be reached.

```
if GPIO.input(26) == False:
       led_X_positive = tk.Label( text="  ", bg="red")
       led_X_positive.place(height=10,width=10, x=420, y=350)
else:
       led_X_positive = tk.Label( text="  ", bg="green")
       led_X_positive.place(height=10,width=10, x=420, y=350)
```

Figure 77: Raspberry limit switch checking status

5 Circuit diagram

The main topic of this chapter is the electrical wiring between the components selected. The results present will be used to construct the electrical panel, where the hardware is contained. We can divide the electric scheme in 5 main groups that are:

- The supply
- Raspberry device
- Arduino devices
- Driver and motor
- Limit switch and emergency push button.

5.1 Supplies

Considering the supply part, we need to connect to the electric power all our devices. Raspberry has a supply voltage of 5 V with a maximum current consumption of 3 A. Arduino is supplied by the Raspberry with the USB cable used also for the serial communication. The driver has a supply according to the rating voltage and current of the motors that are 24V and 4A for each motor. The limit switch and the emergency push button work with 24 V with a total current consumption of less than 1 A. Considering the 24 V supply we need at least 10 A of current capability. Also, to protect the supply and the hardware, a fuse is used with the aim of breaking the circuit if a fault will occurs. The fuse has a cut current of 10 A. Furthermore, is insert the automatic circuit breaker. The 24V and 5V tensions are marked with the C1 and C2 header on the cable. The scheme related to the supply is shown in the Figure 78.

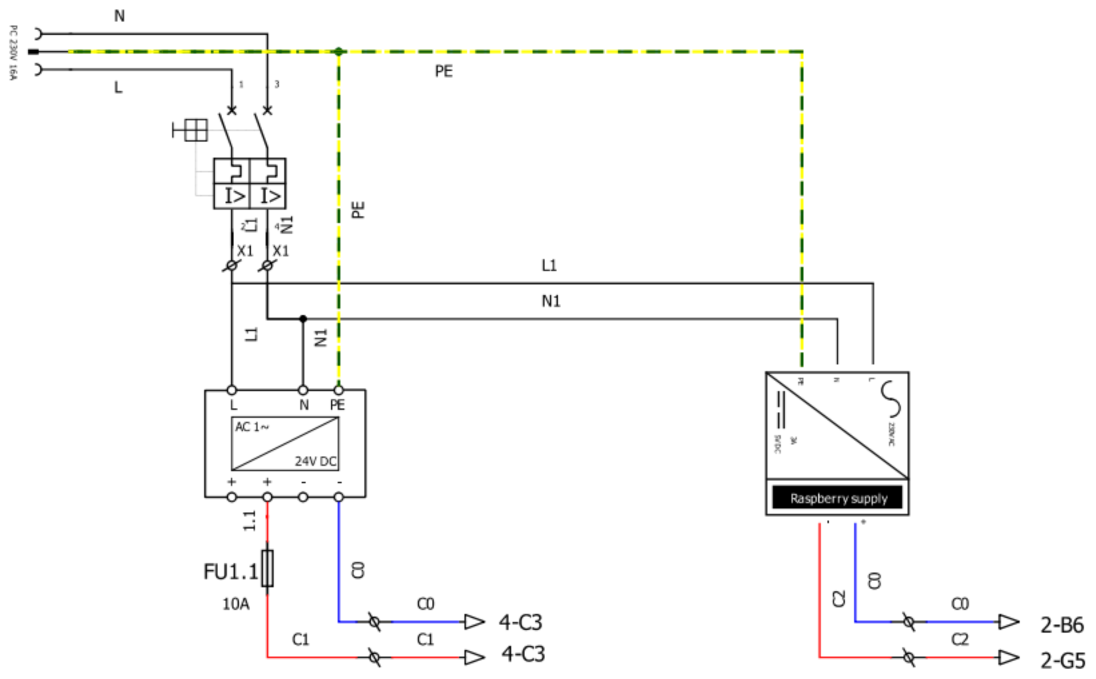


Figure 78: Wiring diagram of supplies

5.2 Raspberry

In the Figure 79 we can appreciate the raspberry wire connection. It is supply with 5V (C2). Also, using its port are attached: the USB cable to the Arduino U and V, the HDMI to the display, and the keyboard and the mouse on the USB input. Regarding to the GPIO input output, the encoder U and V (cables 2.3 and, 2.4), the limit switch signal (cables 2.5, 2.6, 2.7 and, 2.8) and the emergency push button (cable 2.9) are connected. The normally closed switch of the limit switch and the emergency, are controlled by relays, acting as separator since the different supply voltages. The limit switch and emergency are supplied with 24 V and the GPIO accept 3.3V, due to the lower voltage a step-down is used to reduce the voltage from 5V to the 3.3V.

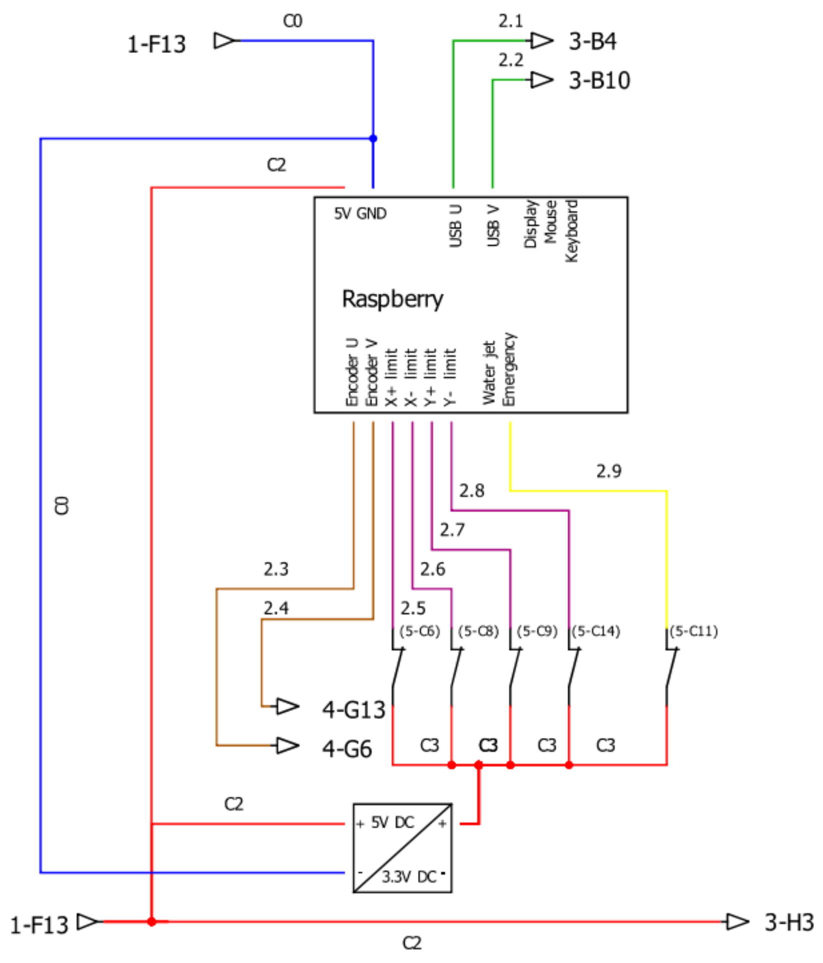


Figure 79: Wiring diagram of Raspberry

5.3 Arduino

The electric scheme of the wiring of the Arduinos is represented in the Figure 80. As we can see, both Arduino are connected through the USB to the Raspberry (cables 2.1 and, 2.2). Using the GPIO, the limit switch and the emergency status are sense (cables 3.1, 3.2, 3.3, 3,4 and, 3.5). Also, the wires to synchronize the motion are connected (Waiting function, cables 3.6 and 3.7). The generate Step and Direction signal of both Arduino are connected to the proper motor driver (cables 3.8, 3.9, 3.10 and, 3.11).

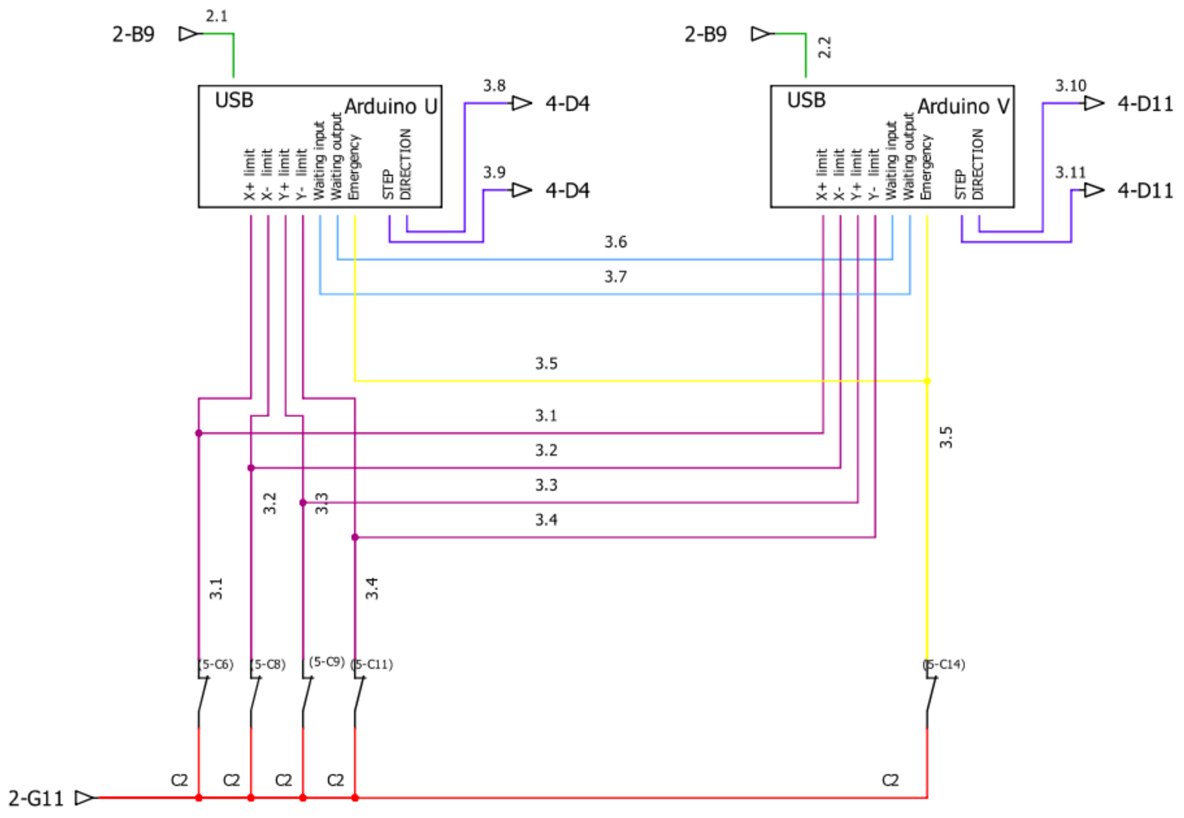


Figure 80: Wiring diagram of Arduino

5.4 Drivers and motors

In the Figure 81 we can appreciate the drivers and the motors connection. The drivers are supplied with 24V and take as input the step and direction signal (cables 3.8, 3.9, 3.10 and, 3.11) coming from the proper Arduino individually. The driver actuates the motors with 4 wires for each one, where control the 2 inside coils (cables 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, and 4,8). On both motor is present the encoder dedicates output signal (cables 2.3 and, 2.4) that is manage by Raspberry with the aim to create the map.

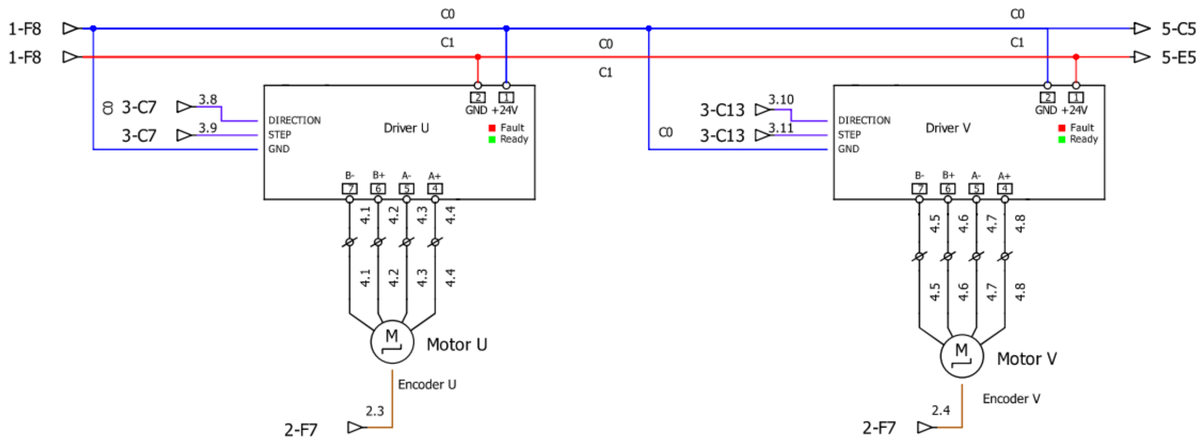


Figure 81: Wiring diagram of drivers and motors

5.5 Limit switches and emergency

Finally in the Figure 82 we can appreciate the limit switch and emergency wiring. The photosensors and the emergency push button are supplied with 24V and are normally closed type according to the security logic adopted. The outputs are connected to the relays in order to separate the circuit, since they are taken as input from Raspberry and Arduino must accept 3.3V and 5V respectively. The relays must control 2 outputs.

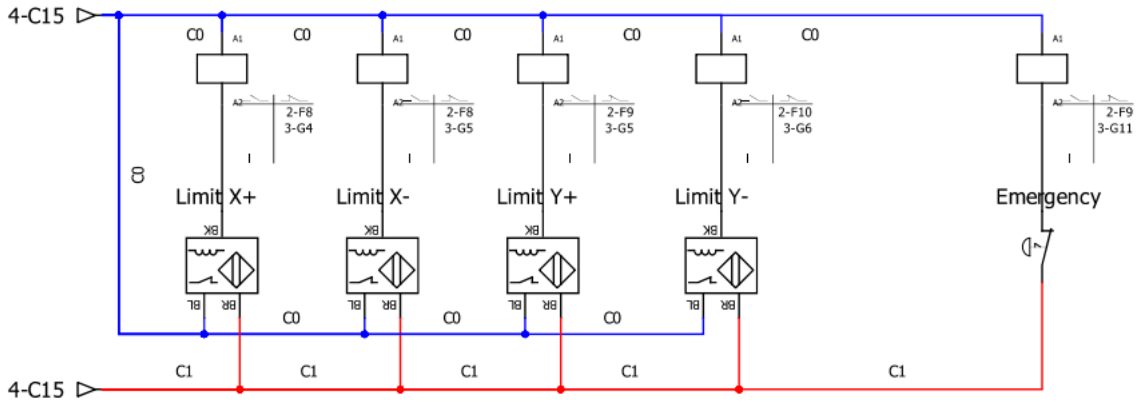


Figure 82: Wiring diagram of the limit switches and emergency

6 Results

The main topic of this chapter is to show the implementation of the machine and the test of some G-code path. The machine has not been constructed as the 3D CAD draw (chapter 2, mechanic) using the component selected, but a scale prototype has been built to verify the correct mechanical, electrical and software operation. The main material for the construction of the chassis is wood, the linear guide has been made using a circular shaped rod in metal (INOX), for the slide bearings we use a holed cylinder in Teflon and for the belt and pulleys we use a plastic wire and reduced scale pulleys respectively. The motor placement is the unconventional type developed. Considering the motors, we decided to mount the stepper with an encoder M1233062S8 with coupling NEMA 23 and 1.8 N*m of torque at 4.2 Arms produced by the LAM company. The electronic infrastructure remained the same with Raspberry as master, and one Arduino controller (as slave) and one motor driver for each stepper. The software has remained the usual that we developed. In the Figure 83 refers to the view from above of the prototype, we can appreciate the placement of both motors on the machine chassis.

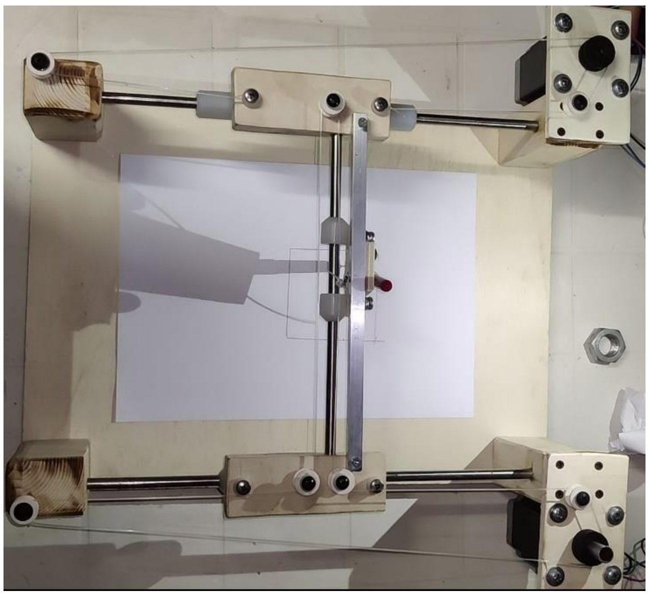


Figure 83: Prototype in scale

The Figure 84 shown a close up of the belt configuration on the small tray (the tray where the end effector is placed). We can see that a pen has been used as end effector (to draw the movements) that is attached on the belt using a bracket. The only degree of freedom that the end effector has, is the linear movement on the linear guide, the Y axis. The movement on the X axis is performed by the couple of linear guides placed on the sides of the image.

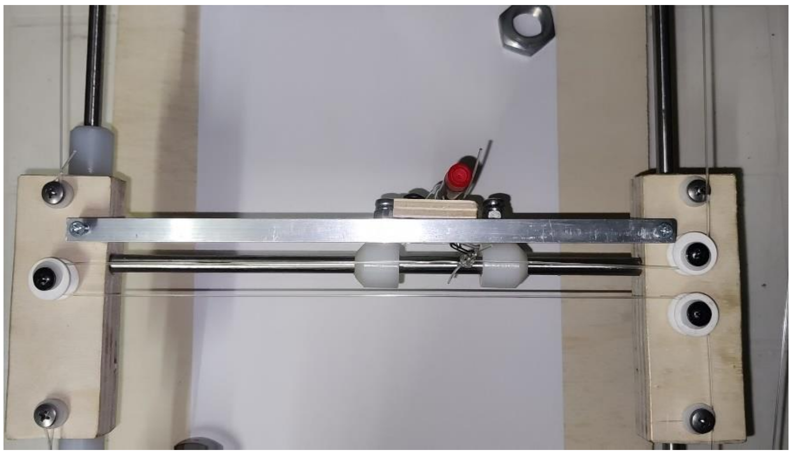


Figure 84: Prototype close up

The Figure 85 refers to the implemented graphical interface that comes out by opening the executable application. As we can appreciate, the window is rendered as the native system graphic. In the header of the window, we have the presence of the application name, Wenji. The implemented buttons are present and link to their functions, also in left bottom part of the image is present the map represented by the black square, where the position of the end effector is placed inside (the white square). Referring to the right bottom part of the image, we can appreciate the limit switches status indicator. In this case all of them are disabled and a green square represents their status (since the end effector is in the central part of the cutting area and no limit switch is reached).

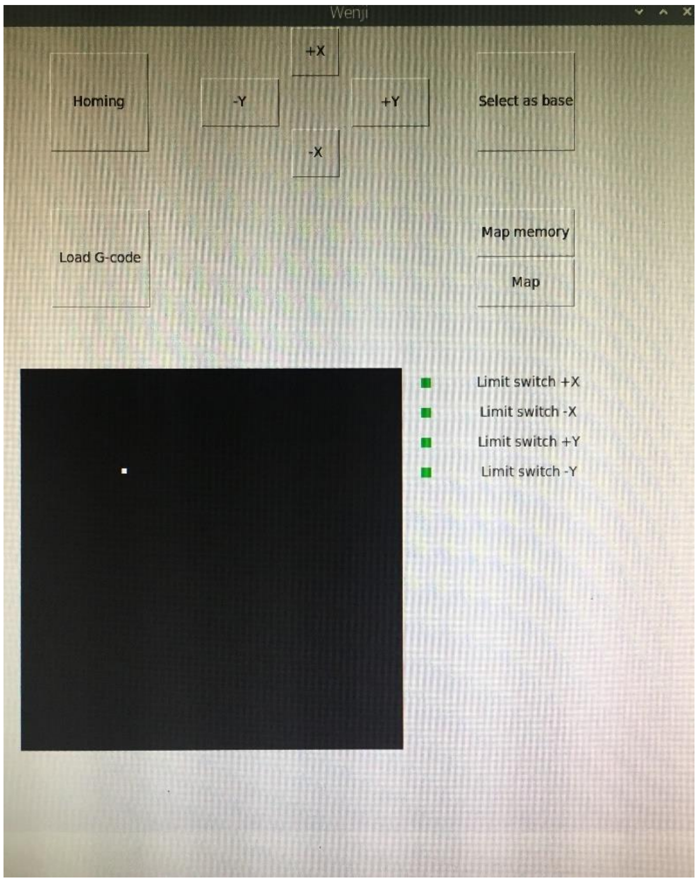
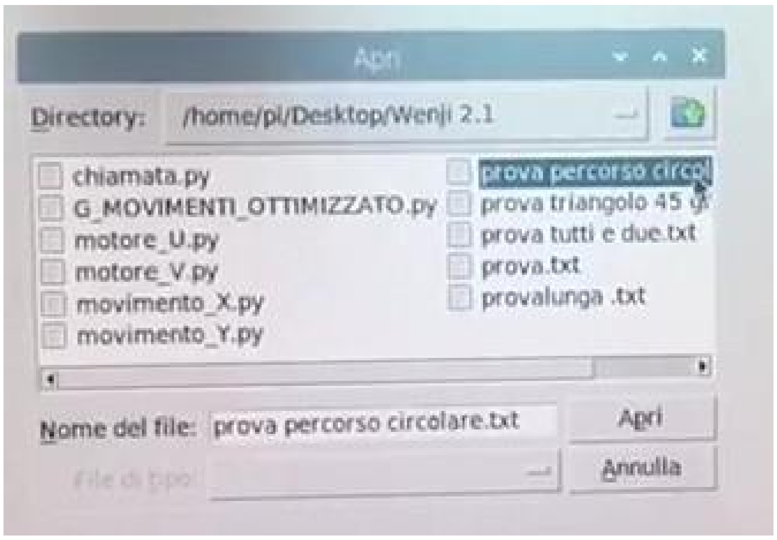


Figure 85: HMI interface

Pressing on the button “Load G-Code” a dialog window appears, giving the possibility to load a G-code text file. In the code is implemented the discrimination of the files displayed restrict only on the format .txt and .py. This is done in order to avoid the possibility caused by the inattention of the users in the selection of the correct format. In Figure 86 is represented the navigation window.


Apri

Directory: /home/pi/Desktop/Wenji 2.1

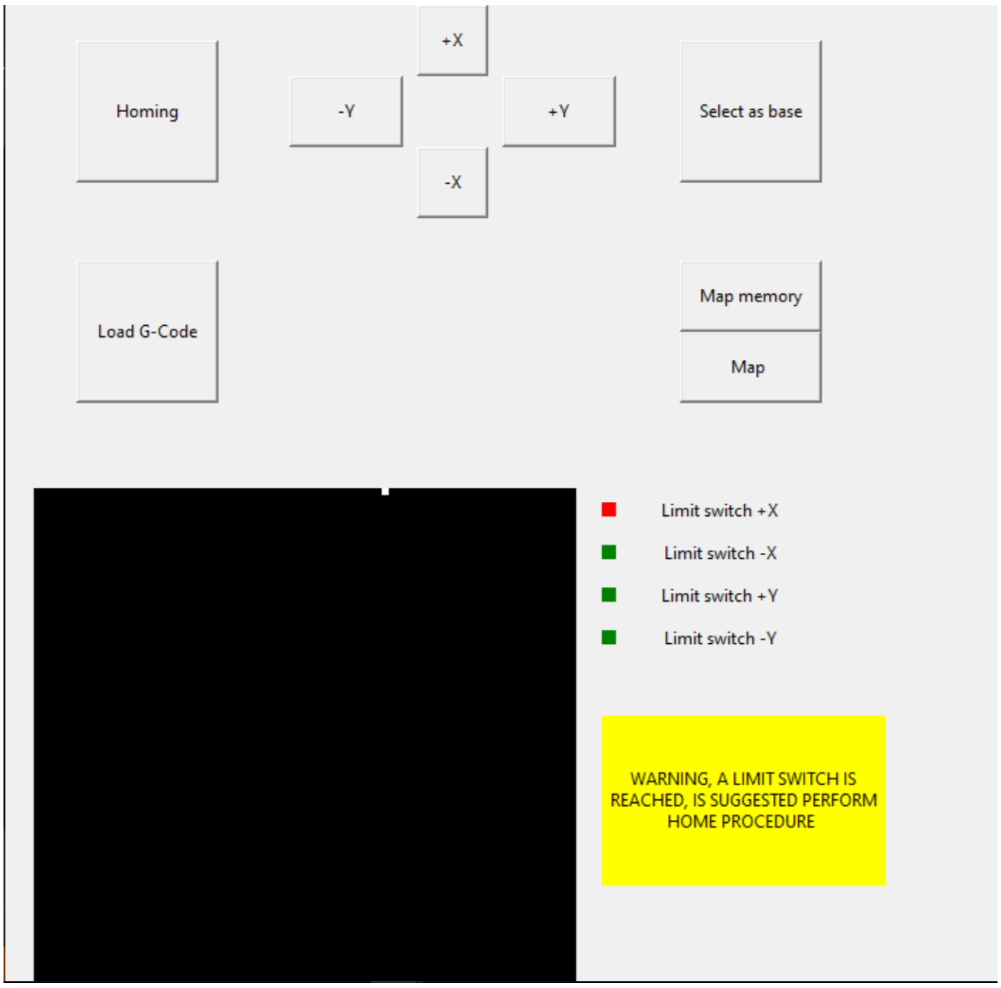
chiamata.py
G_MOVIMENTI_OTTIMIZZATO.py
motore_U.py
motore_V.py
movimento_X.py
movimento_Y.py
prova percorso circol
prova triangolo 45 g
prova tutti e due.txt
prova.txt
provalunga .txt

Nome del file: prova percorso circolare.txt

File di tipo:

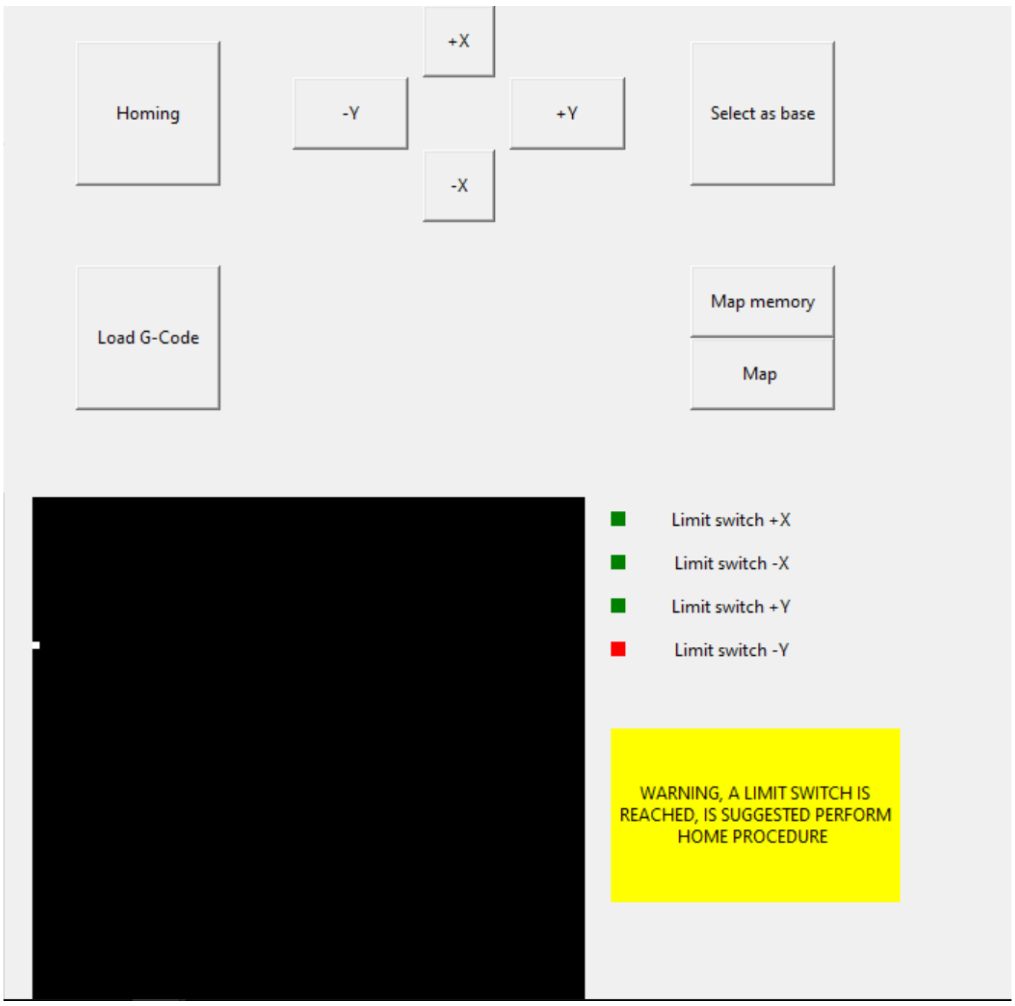
Figure 86: Navigation window

Considering the limit switch checking status, an example of activation is shown in the figure 87 and 88. In the first figure the limit switch relative to the positive X axis is reached. Then the switch will produce a low signal, according to the positive logic presented in the chapter 4.2.2.1 means the activation of it. So the signal is take as input from the GPIO pin number 26 and the software recognize the activation of the limit switch displaying a symbolizing red led to the reached switch on the HMI. In the Figure 88 is represented the reaching of the limit switch placed on the negative Y axis. According to what described in the chapter 4.2.6, a message on the HMI reminds to perform the homing procedure in order to calibrate the machine, since could happed a misalignment of the real position of the end effector with the position known by the machine. As we can see, in the figure is presented the button with the name “Map memory” and “Map”. These tools are used to not clean the map between G-code file execution. For instance, if is active button “Map memory” the map will be store until is push the button “Map”. On the contrary, if the button “Map” is active the map will be initialized as new at every loading of G-Code file.



WARNING, A LIMIT SWITCH IS REACHED, IS SUGGESTED PERFORM HOME PROCEDURE

Figure 87: Reach positive X limit switch



WARNING, A LIMIT SWITCH IS REACHED, IS SUGGESTED PERFORM HOME PROCEDURE

Figure 88: Reach negative Y limit switch

In the Figure 89 are shown some cutting path visualized of the map. We can appreciate:

- A rectangle
- A rectangle with circular corners
- A zig-zag path with an oblique line

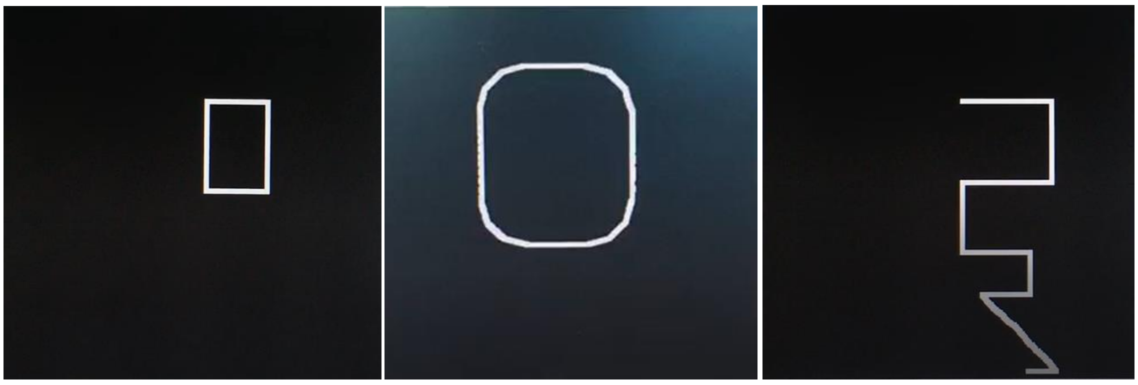


Figure 89: Map path visual example

The Figure 90 refigure a cutting path performed by the prototype. The corner of the path is not orthogonal but are curved due to the mechanical flexibility of the chassis, which would not occur in a rigid structure.

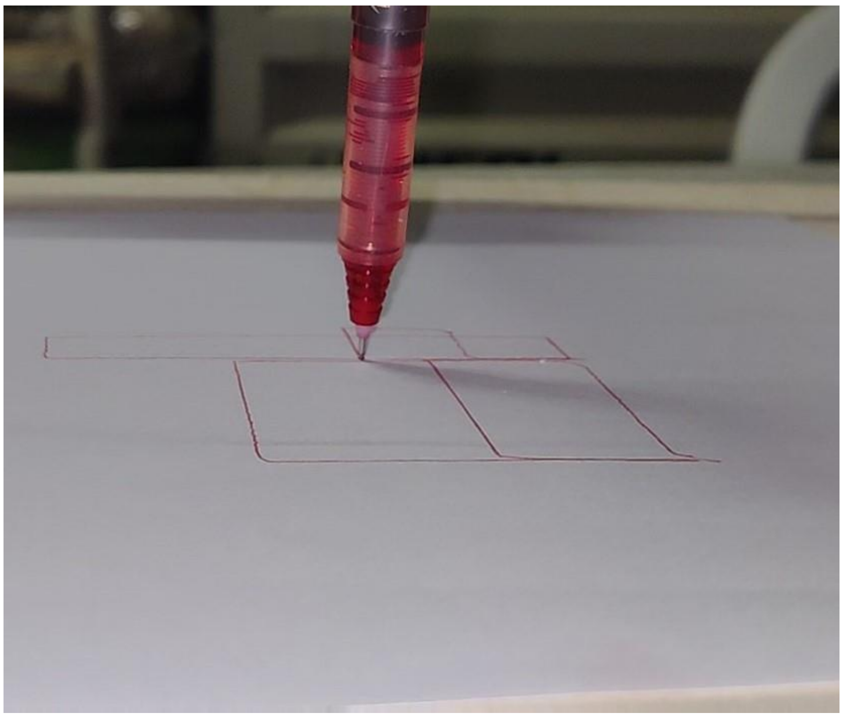


Figure 90: Path draw

6 Conclusion

The aim of this thesis is to develop a planar plotter for food cutting using an unconventional pose of motor. We can divided the project in 4 main subsections: the mechanical, the electronics, the software, and the wiring diagram sections.

Considering the mechanical section, an analysis of the working environment has been done and gives as result the constraint that we must fulfil in order to have a reliable machine. Secondly, we have followed the FDA, HACCP, and the current European regulation in the selection of materials used to build the mechanical part of the machine. Thirdly, an analysis of the dynamic of the machine has been done in order to compute the necessary torque to fulfill the performance requested. The results of this section are the unconventional pose of the motors, that have allowed the fulfillment of the cleanability constraint, the mechanical design of the machine, and the choice of the components in terms of stepper motors, linear guides slide bearings, and pulleys. Finally, a drawing of the machine has been done using the 3D CAD software in order to build the machine.

In the electrical section, we have performed the selection and the choice of the needed electronic hardware in order to control the machine. In particular, the HMI, the PC, the motor controller, and motor drivers has been selected.

The software section refers to the implementation of the HMI and the code to control the motors. Also, we have chosen the topology of the control with the PC as master and the motor controllers as slave with a star network topology. Firstly, the master-slave communication has been developed using a handshake protocol to ensure and increase the reliability. Secondly, we have analyzed which graphical widgets must have a CNC machine, since the plotter belong to this type of machines. The result achieved are the implementation on the PC of the joystick, the G-Code load files and translator, the homing procedure and base selection, and the map of the cutting area. So, we proceeded ahead with the implementation of the mentioned features. Then, the control program contained in the slaves has been develop and includes, the creation of the step and direction signals, and the limit switches and emergency handling.

Considering the wiring diagram section, the electrical diagrams that must be followed in the wiring of the electrical panel have been designed.

The final result is a fully working machine, where the cutting path can be set using G-Code file and in particular can perform the G-code header as G00, G01, and G02. Also, on the HMI of the machine, the performing cut is displayed on a map. The homing procedure has been implemented and has the role to calibrating the machine and move the end effector in the base point. The base point can be set as the user wishes using the joystick in order to reach it and then pushing the dedicated button. Also, for security reasons, the limit switches and the emergency status have been implemented, blocking the movement in case of activation. The unconventional pose of the motors allows the placement of them on the machine chassis. So, is avoided the movement of one of them and the proliferation of bacteria nest on it since both motors are out of the cutting area and the water and food splashes cannot reach them.

That is a problem that could occurs on the conventional cartesian plotter. In addition, the double tray configuration implemented, allows to manage the nozzle with the upper tray and with the bottom tray, to follow the residue water jet in order to collect the water using a pipe.

During the development we have incurred in the performance limitation of the PC. This problem was found in the map construction. In fact, as has been analyzed in the chapter 2.2.6, the step frequency corresponding to the maximum velocity is $476.2\ Hz$ with 200 pulse in a full rotation so, considering that the encoder creates 512 pulses in a full rotation it means that at the maximum speed the encoder will produce a signal with $1219\ Hz$. This frequency is above the computational capabilities of the PC and could occur that some encoder pulses are lost. This this lack affects the construction of the map, an in particular, the represented end effector does not follow correctly the imposed path even if the cut will be successful performed.

During the development of the machine to me and my tutor came up with the idea of equipping the computer vision on the machine. The idea refers to the application of the developed machine to the leather cut, where using the intelligence, we have left the task to the PC to choose the best path in order to minimize the waste material. Another improvements that can be implemented, is the recycle of the used water coming out from the cut. Since our world is heading towards a more efficient use of resources, we could develop a closed loop system of the water, filtering it after its use.

Bibliography

[1] Carreño-Olejua, René, Werner C. Hofacker, and Oliver Hensel. "High-pressure water-jet technology as a method of improving the quality of post-harvest processing." Food and bioprocess technology 3.6 (2010): 853-860.

[2] Regulation 27 October 2004 No:1935/2004 “On materials and articles intended to come into contact with food and repealing Directives 80/590/EEC and 89/109/EEC”

[3] Schmidt, Ronald H. Basic elements of equipment cleaning and sanitizing in food processing and handling operations. University of Florida Cooperative Extension Service, Institute of Food and Agriculture Sciences, EDIS, 1997

[4] Timperley, D. A., R. H. Thorpe, and J. T. Holah. "Implications of engineering design in food industry hygiene." Biofilms—Science and Technology. Springer, Dordrecht, 1992. 379-393.

[5] VanRossum, Guido, and Fred L. Drake. The python language reference. Amsterdam, Netherlands: Python Software Foundation, 2010.

[6] Badamasi, Yusuf Abdullahi. "The working principle of an Arduino." 2014 11th international conference on electronics, computer and computation (ICECCO). IEEE, 2014.

[7] Cicolani, Jeff. "Raspberry Pi GPIO." Beginning Robotics with Raspberry Pi and Arduino. Apress, Berkeley, CA, 2018. 103-128.