School of Industrial and Information Engineering Department

of Mechanical Engineering

DESIGN AND DEVELOPMENT OF A CONTINUOUS ROTATION VSA

Relator: Prof. Giberti Hermes

Correlator: Ing. Castelli Kevin

Master Thesis by:

Marco Gavioli 858385

Academic year 2019/2020

“Happiness can be found, even in the darkest of times,
if one only remembers to turn on the light”
Albus P.W.B. Dumbledore

J.K.Rowling - Harry Potter and the Prisoner of Azkaban

Sommario

Negli ultimi decenni la presenza di robot in ambito industriale è aumentata fino a diventare quasi indispensabile. Si sono evoluti rapidamente in ogni ambito fino a diventare in grado di svolgere ogni genere di compito. Le caratteristiche principali dei robot industriali sono la precisione e la ripetibilità, qualità che possono essere raggiunte grazie all'utilizzo di un attuatore rigido e che sono essenziali in uso industriale dove la presenza umana non è contemplata. Per poter entrare a far parte della quotidianità umana ed essere in grado di lavorare fianco a fianco con le persone, il focus dei robot si è spostato dalla precisione alla sicurezza e dalla ripetibilità alla adattabilità. Ciò ha portato alla nascita dei Robot Collaborativi (Cobot), i quali sono in grado di assistere l'uomo e di collaborare con lui.

Al giorno d'oggi i cobot sono una realtà in espansione e le loro caratteristiche essenziali sono garantite da una versione modificata dell'attuatore rigido tipico dei robot, chiamato Attuatore ad Impedenza Variabile (VIA). I VIA sono dispositivi caratterizzati da una routine o un meccanismo fisico che permette ai cobot di interagire con l'uomo, senza essere un rischio per la sua sicurezza.

Tra i vari modelli di VIA, quello in cui l'impedenza è rappresentata da una molla meccanica è chiamato Attuatore a Rigidezza Variabile (VSA).

L'obiettivo della tesi è lo sviluppo, sia meccanico che a livello di software, e la realizzazione di un VSA a basso costo capace di adattarsi a più situazioni possibili.

Partendo da uno studio matematico della situazione ideale, il prototipo è analizzato diviso in moduli per permettere di assemblare facilmente le parti di ricambio o future implementazioni. La progettazione meccanica inizia con il gruppo del motore e continua attraverso l'attuatore, il sistema di molle e il gruppo cinghia. I dispositivi che compongono l'hardware sono scelti e analizzati nel dettaglio per sviluppare l'azione di controllo.

Infine, il prototipo è stato realizzato attraverso la stampa 3D e il suo comportamento è stato confrontato con i risultati dell'analisi matematica.

Abstract

In the past decade robots have increased their presence in industrial environments until becoming almost essential/indispensable. They have evolved fast and widely and, nowadays, can perform every kind of task. The main characteristics of industrial robots have been precision and repeatability, features which can be achieved thanks to a rigid actuator and which are essential for industrial use where human presence is not contemplated. In order to join human everyday life and to be able to work side by side with people, robot focus has been shifted from precision to safety and from repeatability to adaptability. This led to the birth of Collaborative Robots (Cobots), which are able to assist humans and to collaborate with them.

Nowadays Cobots are an expanding reality and their core is represented by a modified version of a robot rigid actuator called Variable Impedance Actuator. VIAs are devices characterized by a routine or a physical mechanism which allows Cobots to interact with humans without exposing them to risk/ without putting them at risk.

Among all VIAs models, the one in which the impedance is represented by a mechanical spring is called Variable Stiffness Actuator.

The aim of this thesis is the design, both mechanical and software, and the realization of a low cost VSA able to adapt to as many circumstances as possible.

Starting from a mathematical dissection of the ideal situation, the prototype is analysed divided into modules in order to allow easy replacements and further re-designs. The mechanical design starts from the motor group and continues through the actuator, the springs system and, finally, the belt group. The hardware devices are chosen and deeply analysed in order to implement the control action.

Finally, the prototype is realized through 3D printing and its behaviour is compared with the initial results of the mathematical analysis.

Index of Contents

List of Figures

List of Codes

Introduction

In recent years the demand of machines able to work side by side with humans in everyday life has been increased. The market answer has been Collaborative Robots (Cobots) and their diffusion has been possible thanks to the introduction of new technological producing methods, such as 3D printing, and affordable microcontrollers.

Traditional robot main features are positioning precision and repeatability, characteristics needed to maintain a constant work efficiency but not suitable for the presence of humans inside their working area: this is the reason why Cobots have been conceived.

Human safety is the crucial point of Cobots development; external unpredictable behaviours are the most challenging tasks they have to deal with and, therefore, they must answer to human actions in the safest possible way.

These characteristics are intrinsic in Cobots definition and must not depend on control strategies or, more in particular, on how external sensors may react to the environment; for these reasons, the solution has been sought in a mechanical design that directly allows to be aware of the position and the applied force at the same time. The core of this behaviour is a new type of actuator, called Variable Impedance Actuator.

Among VIAs, there is a particular type whose impedance is represented by an elastic element; they are called Variable Stiffness Actuator (VSA).

The aim of this dissertation is to design, both mechanically and software, a VSA applicable to a Collaborative Robot.

The VSA under study must be an easy-to-use device that can be positioned in place of traditional actuators.

The main features, the VSA has to satisfy, are defined by contemplating the widest possible applicability; for this reason unrestricted rotation in both directions is the main aim of the mechanical design, followed by the simplicity of assembling and disassembling and by the low production cost.

The thesis features the mechanical design and the hardware/software design but, first of all, the theoretical model of the spring system is analysed through a mathematical point of view. The results of this analysis have been validated by the characterization of a simple prototype. After the design, the prototype has been produced using a 3D printer and its behaviour has been tested.

1 State of art

In order to analyse a VSA, a brief dissection about its more general design, called VIA, is needed. A Variable Impedance Actuator is a device that changes its equilibrium position in relation to applied external loads and to its own mechanical features as defined by Vanderborght et al. [1]. Their behaviour simulates the one of human joints in which the muscles regulate the stiffness of movements. The controls of a cobot reaction to an external interaction can be managed in plenty different ways but, for Vanderborght et al., they can be categorized in four different families.

First of all, it is possible to select a group of VIAs whose impedance variation is regulated by a software control and not by a physical component. This category comprehends all the machines which become aware of the external environment through sensors such as force or torque ones. The compliance is simulated by the software which controls the movement.

The second family is called “Inherent damping” and includes devices in which the damping is physically controlled and operates on the joints.

The third category uses an inertial mass to store or release energy and, for this reason, is called “Inertial”.

The last family is the most significant for this dissertation and it is formed by mechanisms in which the compliance is modified by a compliant element. It is called “Inherent compliance”. Within this family it is possible to highlight two sub-groups identified by the mechanical properties of the complaint mechanism.

The compliance can be fixed as for Serial Elastic Actuators, SEAs, which are the first attempted to apply a flexible element in series to a traditional gear as shown by Pratt and Williamson [2].

The properties of the compliant element can be adaptable, and the energy is stored by and elastic element as happens for the Variable Stiffness Actuators (VSA).

While the first group is interesting due to its simplicity and adaptability to one precise task, the VSA category is more suitable to machines with a wide range of possible activities such as the cobots are.

1.1 Variable Stiffness Actuators

The devices called VSA have, as distinctive characteristic, the possibility to change the compliance and, more in particular, the stiffness to adapt their response to the circumstances. It is straightforward to understand that this modification requires a second control and, thus, a second actuation.

These double actuation units are analysed by Tagliamonte et al. [3] and categorised in relation to the working configuration. It has been decided to divide this kind of devices into three groups.

The first one includes double actuation units in which the displacements of the two motors are summed to generate the total displacement of the output. They are identified as “Serial configuration” devices and, while they can generate high displacements, the maximum torque is limited by the smaller motor. These devices can control the displacement and the impedance, and they can be described as an implementation of the SEA. Figure 1.1 shows a general scheme of this configuration.

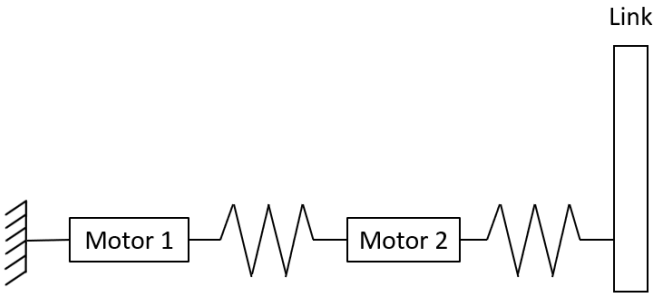
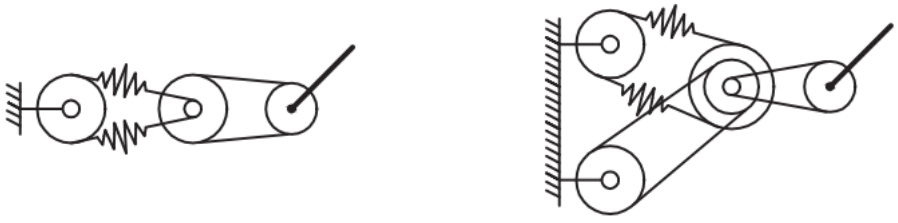


Figure 1.1 Serial Variable Stiffness Actuator Scheme

Within this group many combinations are possible, and they are characterized by the relative position of the motors and the elastic elements.

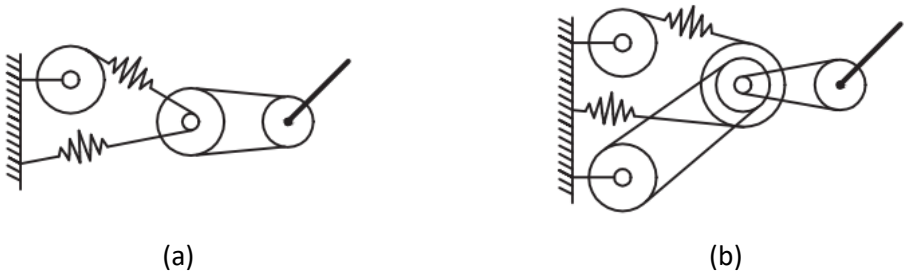
If the elastic elements are both before the second motor, they are called “ordinary” (Figure 1.2a) while, if one motor controls the torque and one the stiffness, they are called “differential” (Figure 1.2b).



(a) (b)

Figure 1.2 Purely Serial Elastic configuration

Furthermore, the two motors can work in a quasi-antagonistic configuration both ordinary (Figure 1.3a) and differential (Figure 1.3b).



(a) (b)

Figure 1.3 Quasi-antagonistic Serial Elastic configuration

The second group is formed by devices with two actuators whose displacements are not summed up. They present a "Parallel" configuration and the total generated torque is equal to the sum of each actuator ones. Figure 1.4 shows how a parallel VSA can be schematized.

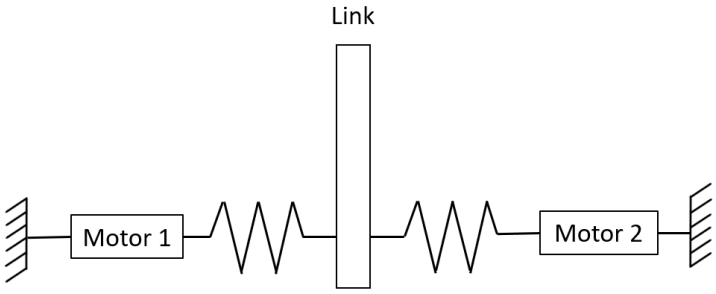


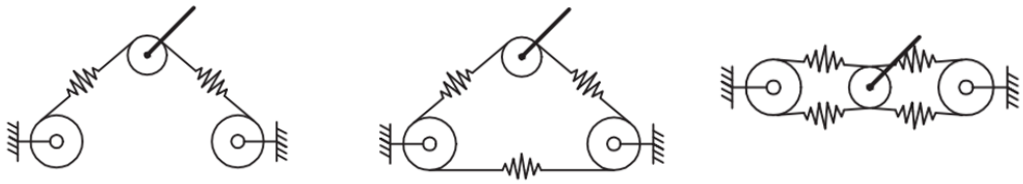
Figure 1.4 Parallel Variable Stiffness Actuator Scheme

Purely parallel actuators can be configured as "collocated" (Figure 1.5a) or "distributed" (Figure 1.5b).

(a) (b)

Figure 1.5 Purely Parallel Elastic configuration

It is possible to state the parallel configuration as the most similar to human junction way of working because each joint has always at least two muscles which work is to contrast the other one increasing or decreasing its stiffness. The actuator or muscle, which work as just described, are called agonist/antagonist and the configuration can be “simple” (Figure 1.6a), “cross coupled” (Figure 1.6b) or “bidirectional” (Figure 1.6c).



(a) (b) (c)

Figure 1.6 Agonistic/Antagonistic Parallel Elastic configuration

This dissection shows only a brief summary of all the configurations because they are deeply investigated in [3].

The third and last group, in which the category of the double actuation units can be divided, is called “Physically controllable impedance” and includes all the configuration which cannot be considered neither serial nor parallel. One of the motors controls the elastic properties and the other one the motion.

In accordance with Van Ham et al [4], there can be a further distinction among the devices included in the “physically controllable impedance”. They consider the division between the ones designed to adjust their natural dynamics and the ones developed for human-robot interaction. The former use the compliance modification to reach a desired motion and to reduce the energy consumption. The latter, instead, are used to increase the safety and the possibility of co-working between humans and machines.

1.2 Existing VSA

At the present day, some models of VSA are already present on the market and they have been already categorized and analysed by a consortium called Viactors [5].

Its aim is the creation of a solid theoretical base in order to understand and to further develop the technology behind a variable stiffness actuator.

Carloni et al [6], members of the consortium, analyse the variable stiffness actuators in relation to the power flow and proposed a port-based model. Regarding the present dissertation, it is interesting because they group the VSA models in three categories in relation to their working principle.

Some models act on the transmission ratio between the internal spring and the actuator output modifying the length of a lever. This is the case of a VSA called AwAS [7] in which the springs have constant pretension while their fixation points change.

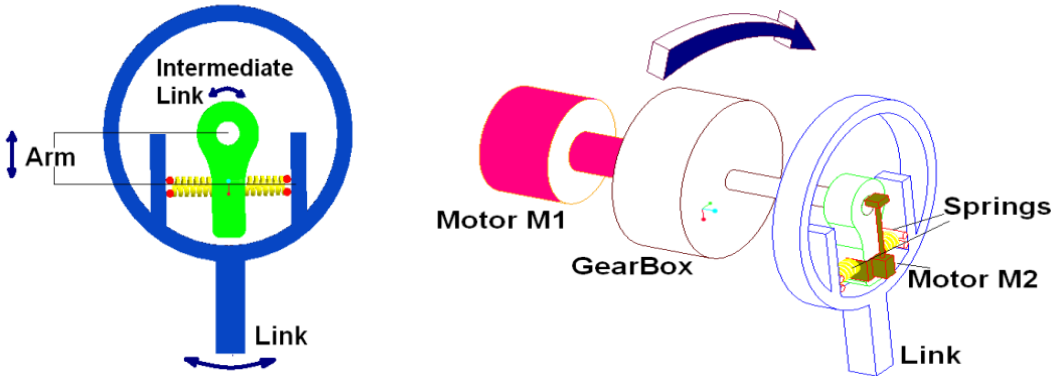


Figure 1.7 AwAS schematics

Figure 1.7 shows a scheme of the mechanism which regulates the stiffness. The first motor regulates the rotation of the output and the second one changes the dimension of the arm. This configuration ensures that, ideally, no energy is needed to change the stiffness of the device. The maximum stiffness in related to the length of the effective arm and to the spring rate.

A second and improved version of the AwAS is called AwAS-II [8]. The mechanism has been changed fixing the position of the springs and the force application point while the pivot point changes. The following Figure 1.8 shows the mechanism of this device:

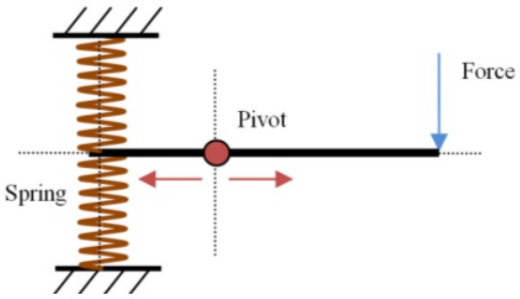


Figure 1.8 AwAS-II mechanism

If the pivot point is closed to the springs the stiffness goes to zero while, if the pivot is closed to the force application point, the device is nearly rigid.

Other VSA models are based on an antagonistic spring setup in which nonlinear springs act in opposite directions. VSA [9] and VSA-II [10] are characterized by this kind of mechanism.

The former, shown in Figure 1.9, is a device in which the transmission is carried out by a belt moved by two motors. The belt is pretensioned by three springs with the same elastic constants. If the rotations of the motors are equal, the stiffness does not change. If motor 1 rotates counter-clockwise and motor 2 rotates clockwise, the stiffness increases while, if the opposite happens, it decreases.

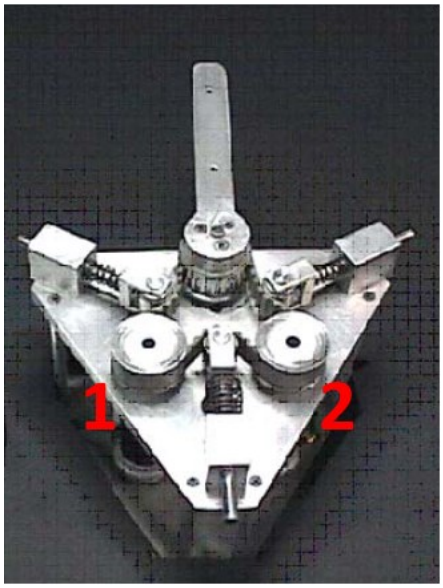


Figure 1.9 VSA

The latter is an evolution of the VSA, and it is called VSA-II. Actually, the mechanism is completely new because it is based on two four-bar linkage transmissions whose input bars

are connected to linear springs. The output bars of the linkage mechanisms are connected to the output of the whole device. it is shown in Figure 1.10.

Figure 1.10 VSA-II

Another example of the antagonistic spring setup is the VSA-Cube [11], shown in Figure 1.11, which is also a perfect example of low cost variable stiffness actuator.

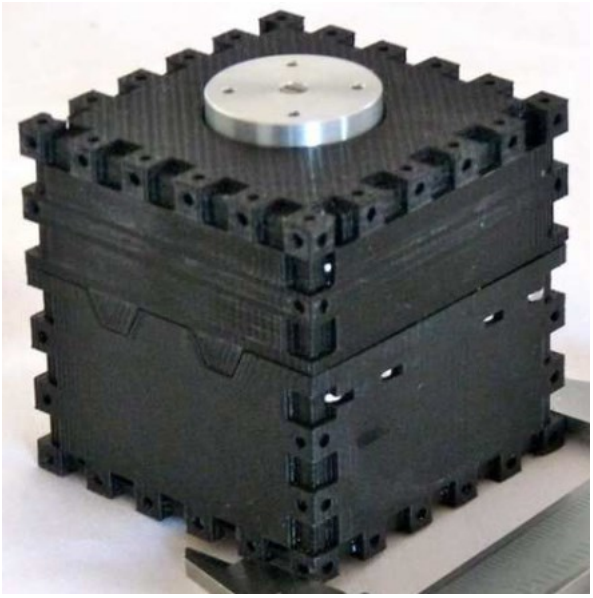


Figure 1.11 VSA-Cube

The last setup analysed can be seen in [12] where the VS-Joint is presented. It has a design based on a mechanical decoupling which means that the displacement and the stiffness control are completely separated on a mechanical point of view. One motor controls the joint position and the other one controls the stiffness value. Figure 1.12 shows a brief schematic of the device.

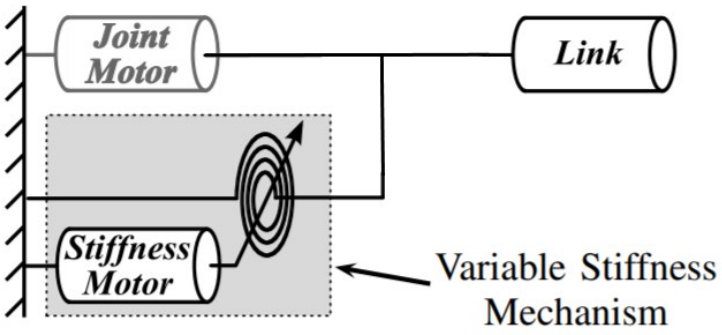


Figure 1.12 VS-Joint Schematic

This approach is also adopted by the OCRJ analysed in [13] in which the compliance element are four leaf springs with variable length. The length modification affects the deflection of the polypropylene foils. The joint motor controls the displacement of the output shaft while the second motor is responsible for the motion of the slider.

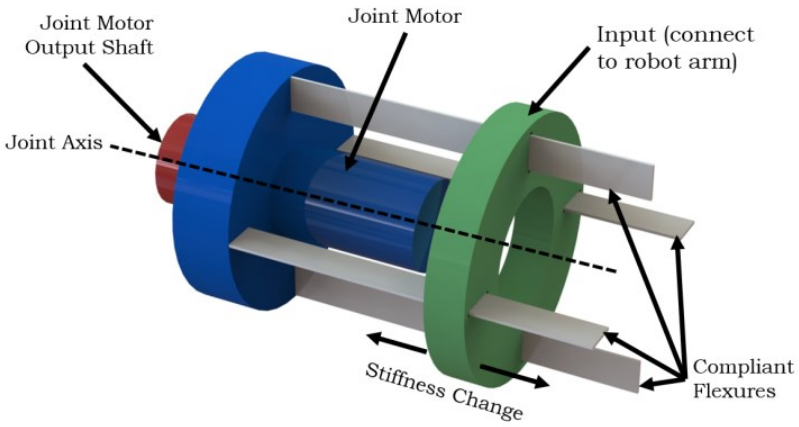


Figure 1.13 OCRJ

2 Mechanical design and dynamic analysis

The mechanical design of the VSA actuator, object of the study, followed the guidelines defined by the possible chosen application. The purpose was the use of the VSA in collaborative machines and, more precisely, in collaborative robotic arms.

This kind of applications can be useful in a wide range of situations, from industrial assembly lines and interactive toys for kids, to every operation in which a machine working side by side with a human is needed. The crucial point of the design was the safety: a collaborative machine must never be a risk for human health. It was decided to design a fully "independent from external environment" VSA actuator. It means that, thanks to a general control algorithm, the VSA lays the basis for a cobot able to avoid destructive impacts if it collides with unplanned obstacles and, thus, to modify its trajectory if needed.

The wide range of applications, in which the actuator can be used, defined one of the crucial features, which was the continuous rotation in both directions, to allow large movement without mechanical limitations.

Regarding the stiffness variator, the main points of the design were the choice of the spring typology, the application of the spring and the method to vary the stiffness.

2.1 Adopted solution

Six different hypotheses were considered:

1. The first idea was based on a lever mechanism which had the point of application of the force that could be moved to modify the torque. The system was constituted by a single lever linked to the output axis by four springs. The force application point could slide within the lever to vary the ratio between input and output force which operates on the springs. The movement was managed by a conjugated cam which generated a linear path. The motion of the force application point simulated the stiffness variation. The main issue of this idea was the management of the cam motion, it needed at least two motor: one for each cam. In addition, the synchronization of the cams motion demanded the presence of one encoder for each motor and one to read the position of the output shaft. Furthermore, the stiffness range of variation was limited by the length of the lever and a completely rigid actuator was difficult to achieve.
2. The second hypothesis was still based on a lever mechanism, but the stiffness variation was guaranteed by the motion of the pivot. The system was similar to a

traditional crank-rod system in which the extremal point of the crank was linked to the output shaft through two springs. The rod was fixed on the input shaft and the crank, leaned on the pivot, transported the rotation to the output shaft. The pivot translation modified the ratio between input and output force. To allow both the directions of rotation, the mechanism would had been doubled and it would become more complex.

3. The third idea considered the opportunity to couple two epicyclic gears with fixed external crown. Only one motor was used to control the rotation of the sun gear, the planetary gears of the two systems were connected through a non-rigid link and transferred the motion to the output shaft. This system underwent no further investigations because it was not possible to separate the rotational motion and the mechanism to vary the stiffness without the use of at least three motor.
4. The fourth idea was completely different from the previous ones because the connection between the input and output shaft was not mechanical but the consequence of the attractive force of two magnets and the stiffness was regulated by the distance between them. The system was mechanically simple, but the control would had been really complex.
5. The other hypotheses were all based on flexible structures and the stiffness was directly related to the number of active unit cells. The auxetic structures were considered but the material composition should had been investigated and this was not among the project objectives.
6. The sixth system was constituted by two disks, linked respectively to the input and output axes, with a spring in between. Originally the spring should had been a torsional compliant mechanism but, for simplicity, it had been substituted by harmonic steel foils used as leaf springs. This application was similar to the OCRJ.

The pro and cons analysis resulted in the development of the hypothesis number six. The simplicity of the idea and the possibility to choose in a second moment the mechanism for the modification of the stiffness were the main motivations of the choice. Figure 2.1 shows a very simple schematic of the device and the possible movements of the output. The device can be categorized as a VSA with a decoupling between the displacement and the stiffness regulation control.

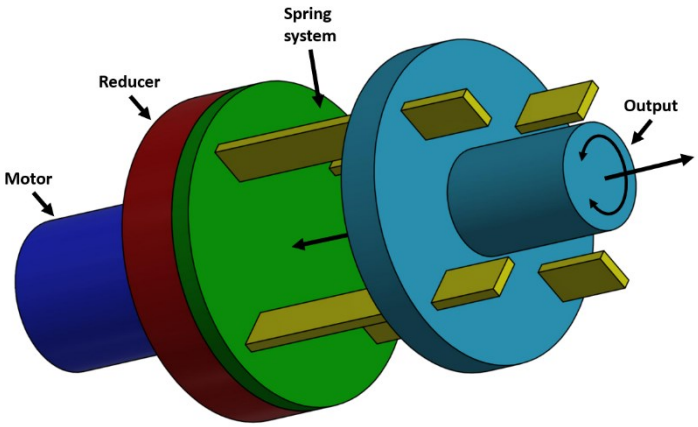


Figure 2.1 Idea n°6 concept

To complete the system, it had been decided to add a compact reducer to decrease the motor speed rotation and increase the torque before entering the VSA module. The needed features reduced the choice among only three kind of actuator: epicyclic actuator, harmonic drive and cycloidal actuator. This was not a fundamental part of the VSA because it could be replaced in further development, but a wrong choice would had brought to problems not directly related to the VSA device itself. Even if an epicyclic gear would had been printable with a 3D printer, the needed accuracy would not be reached and , furthermore, the printing of the flexible spline of the harmonic drive needs a study on the printer itself; these were the reasons why the cycloidal reducer had been chosen.

2.2 Leaf spring characterization

The development of the prototype started with a preliminary study of the ideal VSA system in order to analyse how the steel foil deforms in relation to the input force.
The ideal system was designed as shown in Figure 2.2 where the A-end is fixed and the B-end is a slider. These two constraints make the system hyperstatic and allow to carry out the analysis through the Displacement Method.

Figure 2.2 Undeformed system

Applying in point B a force perpendicular to the beam, the system deforms (Figure 2.3) and it is possible to use the "u" displacement as the variable of the study.

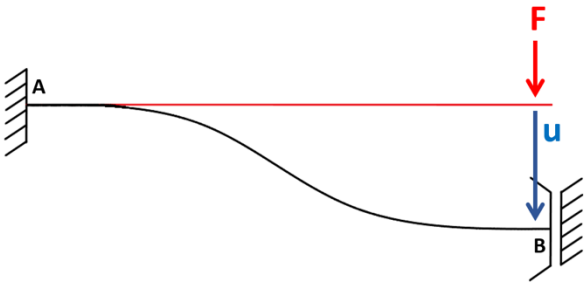


Figure 2.3 Deformed system

The reaction forces generated by the displacement are shown in Figure 2.4 and calculated in equations 2-1 and 2-2:

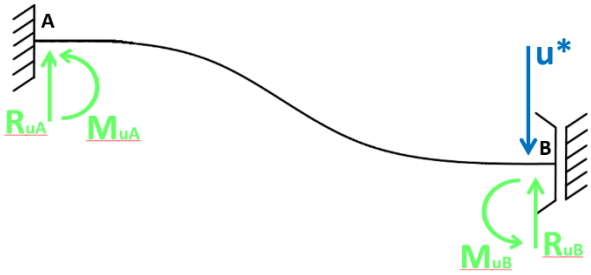


Figure 2.4 Reaction

$$R_{uA} = R_{uB} = \frac{12EJ}{L^3} \quad 2\text{-}1$$

$$M_{uA} = M_{uB} = \frac{6EJ}{L^2} \quad 2\text{-}2$$

The vertical force in B must be equal to 0:

$$F = -u^* \times R_{uB} = -\frac{12EJ}{L^3} \times u^* \quad 2\text{-}3$$

From equation 2-3 it is possible to calculate the particular displacement u* generated by the application of force F:

$$u^* = -\frac{FL^3}{12EJ} \quad 2\text{-}4$$

Knowing the displacement, it is possible to calculate deformation and internal actions with the elastic line differential equation:

$$EJu = a\frac{x^3}{6} + b\frac{x^2}{2} + cx + d \quad 2\text{-}5$$

Where a, b, c and d are coefficients which describe the system and must be calculated imposing boundary conditions:

$$u'(0) = 0 \qquad d = 0 \quad 2\text{-}6$$

$$u(0) = 0 \qquad c = 0 \quad 2\text{-}7$$

$$u'(L) = 0 \qquad a\frac{L^2}{2} + bL = 0$$

$$b = -a\frac{L}{2} \quad 2\text{-}8$$

$$u(L) = u^* \qquad EJu^* = a\frac{L^3}{6} + b\frac{L^2}{2}$$

$$EJu^* = a\frac{L^3}{6} - a\frac{L^3}{4} \quad 2\text{-}9$$

Substituting equation 2-4 into equation 2-9, it is possible to find the value of coefficients a and b:

$$-EJ\frac{FL^3}{12EJ} = -a\frac{L^3}{12}$$

$$a = F \qquad 2\text{-}10$$

$$b = -\frac{FL}{2} \qquad 2\text{-}11$$

The equation which describes how the beam deforms is the following one (2-12):

$$u(x) = F\frac{x^3}{6EJ} - \frac{FLx^2}{4EJ} \qquad 2\text{-}12$$

$$u(x) = \left(\frac{x^3}{6EJ} - \frac{Lx^2}{4EJ}\right) \times F = \frac{1}{K_{foil}}F = C_{foil}F \qquad 2\text{-}13$$

Equation 2-13 shows how displacement u(x) and force F are related through the value of C_{foil}, which is the compliance of the steel foil. The stiffness K_{foil} is the inverse of the compliance and function of the foil length. It can be seen that if the length is near 0 mm, the stiffness value is almost infinite (rigid configuration) and the force needed to generate even a small displacement is very high; furthermore, both stiffness and force decrease with the increasing of the length. The connection between force and displacement, generated by the application of a spring, is essential to the development of the whole device under study.

As initial step, the ideal system calculation was developed in Matlab to understand if the results were feasible. Figure 2.5 shows the results:

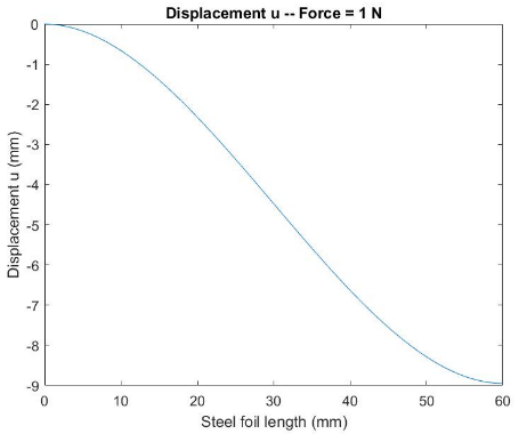


Figure 2.5 Ideal displacement

To validate the calculation, a simple system was designed in order to have a characterization of the behaviour of the steel foil. The system is shown in Figure 2.6 and it is composed by the axes (1), the upper base (2) which can slide along the axes but its rotation is integral with the axes one, the lower base (3) whose position is fixed but it can rotate freely, the encoder (4) and the steel foil (5).

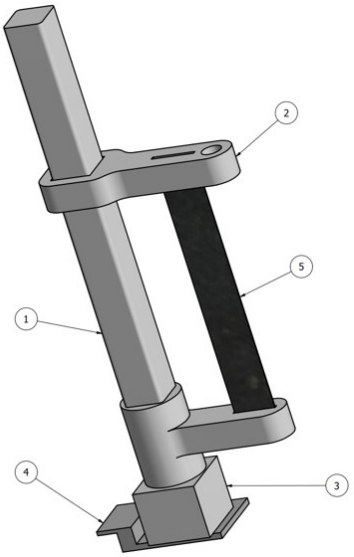


Figure 2.6 Ideal model

In Figure 2.7 it is possible to see that the deformation of the steel foil is slightly different from the predicted one because, in addition to the bending action already analysed, it is present also a torsion. Its effect is very low if the rotation is small and, for this reason, it was neglected from the study.

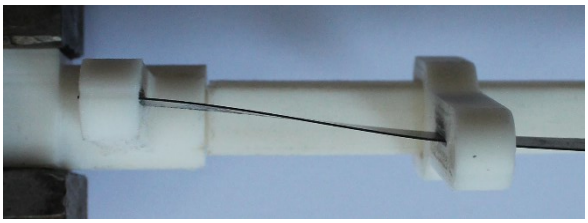


Figure 2.7 Real deformation

In order to measure the rotation of the system it was used a very simple hardware consisting of a control board (Arduino Uno), a rotational encoder with 60 counts per rotation and a dynamometer to measure the applied force.

In the following table are compared the results of the measurements and of the calculation with the application of four different weights:

Weight	Measured rotation	Calculated rotation	Error %
0,055 kg	12°	10,93°	9,79%
0,085 kg	18°	16,61°	8,37%
0,130 kg	24°	24,53°	2,16%
0,220 kg	36°	37,68°	4,46%

More precise measurements would have been possible with a reduction between the encoder and the axis but, even in this situation where the minimum measurable angle of rotation is 6°, the errors are still within ±10% and, for this reason, the deformation hypothesis were stated as acceptable.

2.3 Prototype design

After the identification of the principle which allows the stiffness variation, the development of the actual prototype started. The idea was to design a VSA that can be used in a wide range of situations thanks to a modular system design with interchangeable parts to allow easy replacements.

The prototype, shown in Figure 2.8, is bigger than the needed one in order to allow a more accurate analysis of the mechanisms and of the spring system. It consists of four main parts: the motor group, the actuator, the spring system and the belt group.

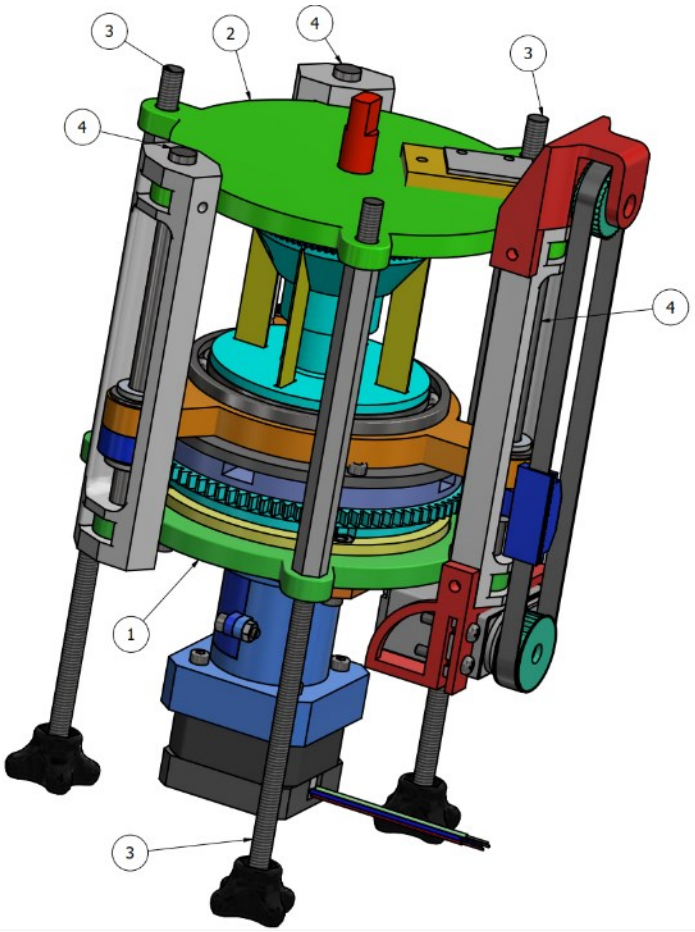


Figure 2.8 Prototype

The four modules are connected by three threaded bars (3) and three rods (4) which link together the lower (1) and upper (2) fixed case base. Rods of 8mm of diameter are needed because the inner mechanism must move, and linear bearings slide over them. Moreover, threaded bars M6 are useful to tight the whole device and to be used as feet of the structure.

Moreover, this kind of connection allows to have a clear view of the working mechanism and it is fast and easy to assemble and disassemble.

As written in the introducing chapter, all the system was designed to be printable by a 3D printer and this is the reason why there is a high number of pieces which compose the mechanism: increasing the parts number means having easy parts to be printed. These parts could have been jointed together with ABS glue, a moisture of ABS scraps and acetone, which avoids the use of bolts. In this case different parts would have become a single entity and, since this is a prototype, it was chosen not to use glue to leave the possibility to change one part without re-printing all the other ones.

The following Figure 2.9 shows that the device was designed to allow to discharge axial reactions generated by motor and output axes on the structure. The colours indicate how each axial reaction is transferred to the structural elements (1) and (2), shown in Figure 2.8. Red reaction is generated by the output axis and it can be discharged in both directions on the upper or lower base; blue reaction can be generated by the cam and it is transferred to the lower base; green reaction comes from the motor axis, goes through the Oldham joint and arrives to the lower base without affecting the input shaft of the actuator. This is possible because the Oldham directly touches the inner cage of the ball bearing and allows an axial relative movement of the shaft.

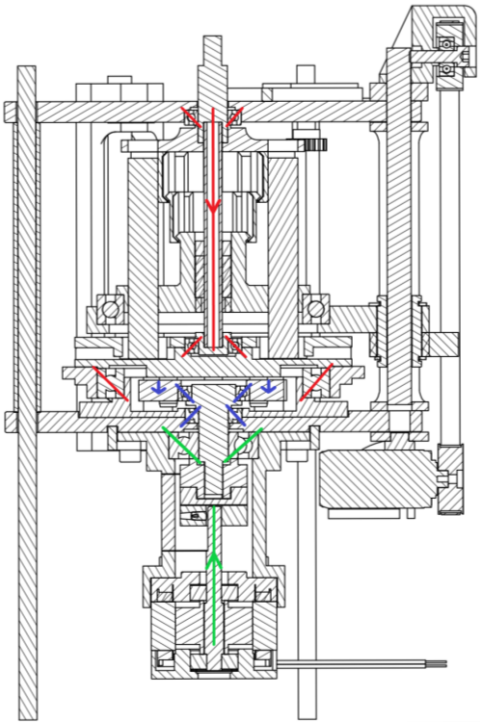


Figure 2.9 Bearings reactions

It can also be seen that each ball bearing is pre-loaded thanks to its housing. Pre-loading a ball bearing is fundamental to avoid noise and vibrations generated by its balls hitting against the cages.

The dissertation started analysing the motor group and proceeded with the actuator, the spring system and, finally, the belt group.

2.3.1 Motor group

The motor group is shown in Figure 2.10. it is designed not to affect the actuator if there is need to change the motor (1). The case (3) is simple and easy to assemble; it has a cover (4) to have enough space to regulate the internal parts.

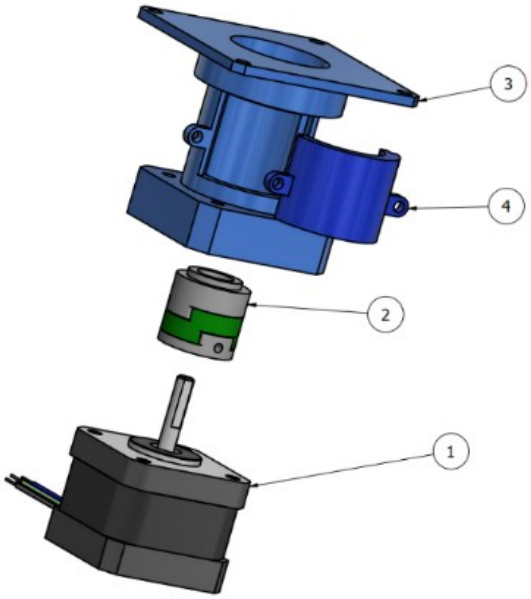


Figure 2.10 Motor group

For the study purpose a motor with high controllability and precision is needed and this is the reason why a stepper motor (1) was chosen.

The connection between the case and the actuator was designed to allow the use of different cases; the motor and the actuator axes are linked together by an Oldham joint (2) which allows a not-perfectly axial motion transfer thanks to its design divided into three parts. Figure 2.11 shows how it was designed: the lower part has a cylindrical hole to be coupling with the motor shaft fixed by a grub screw, the middle part is free to slide allowing to correct small axiality problem and the upper part has to be connected to the actuator input shaft which has a square section but it allows an axial relative shift.

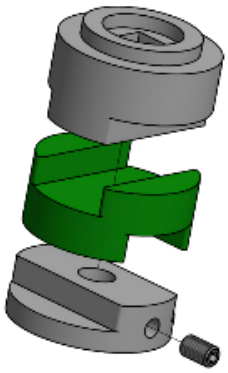


Figure 2.11 Oldham joint

2.3.2 Actuator

The chosen actuator was a cycloidal gear because it allows a high reduction ratio with a compact size. In Figure 2.12 it is possible to see all the elements of the actuator and how they are linked together.

The 1st analysed element was the lower external case (1), which is the whole system base but not directly an actuator part, and it is linked to the upper external case through three threaded bars and three cylindrical rods. Then there are three elements rigidly connected through bolts (16), to the lower case: the adapting case (10), the base (2) and the encoder housing (11). The first one allows to fix the motor case reducing possible movements generated by the rotation of the motor. The base is the housing of the cam (4) and allows its rotation thanks to ten rods which are built-in the base internal profile. It is also the element which aligns the axis of the gear (7) through a ball bearing (12) to the input shaft one. The rotation of the rigid part of the device needs to be measured through an encoder (6) which is held in place by the encoder housing (11). To reduce the space taken by the bolt joints, every nut is hidden inside a hexagonal hole properly designed in relation to the bolt size. The gear (7) forms a rigid group with the rotating base (8) and the closing plate (9), which transfers the rotation generated by the cam (4) to the springs system and, consequently, to the output shaft of the device. The design of the gear has been done using the “gear generator” of a CAD program choosing a reducing ratio equal to 50/10,5. The cam (4) is the core of the actuator and transfers its rotation through six steel rods (5) which are rigidly connected to the rotating base.

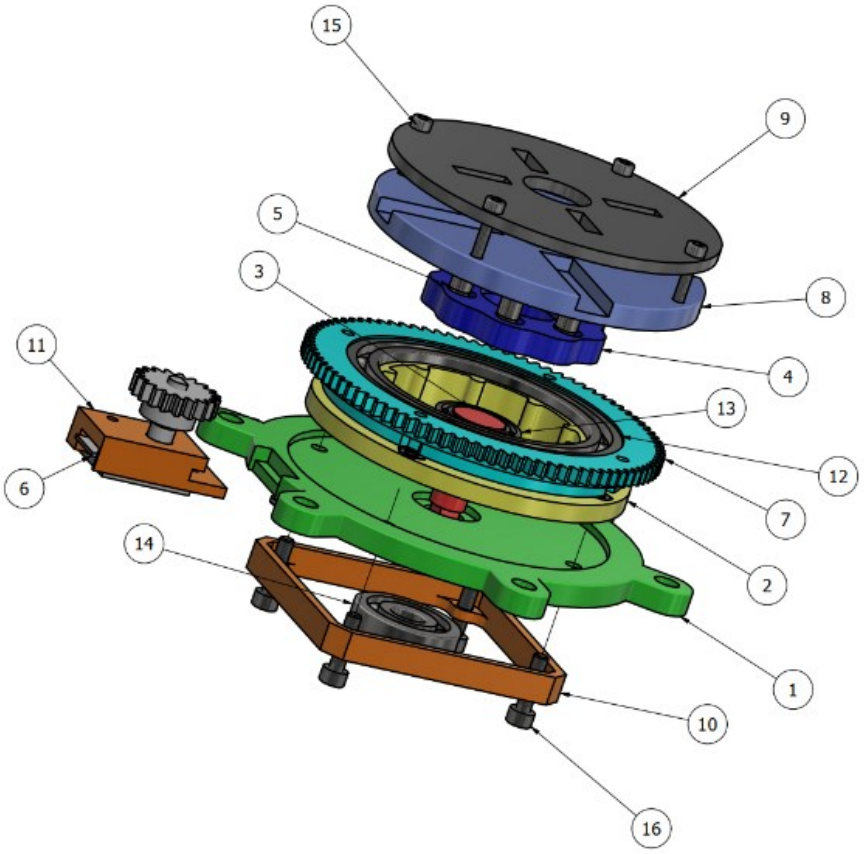


Figure 2.12 Actuator

Before analysing how the cycloidal drive was designed, a brief explanation about how this kind of actuator works is needed.

The rotational motion starts from the input shaft (3) which is mounted eccentrically to a ball bearing (13), causing the cam (4) to move in a circle. It will independently rotate around the bearing as it is pushed against the inner surface of the base where the rods are positioned. The reduction ratio is related to the number of the rods (R) and of the lobes (L) of the cam profile.

$$r = \frac{R - L}{L} \qquad 2\text{-}14$$

Input axis is shown more in detail in Figure 2.13: the motion is transferred from the motor to the input axis through an Oldham joint (Figure 2.11) to permit a slightly misalignment between the two rotation axes. The square section of the input axis is inserted into the Oldham joint, but a slight sliding is allowed.

In Figure 2.14 it is displaced the eccentricity of the final section of the input shaft equal to 1 mm. It is also possible to see how the dimensions have been adapted to the 3d printing tolerances.

Figure 2.13 Input shaft

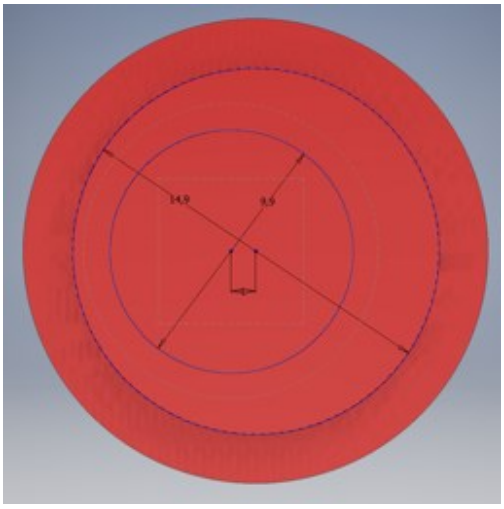


Figure 2.14 Eccentricity

The design of the cam profile was the most challenging task of the reducer study. It was chosen to use a set of equations to describe the cam profile and to implement it directly in a sketch of the 3D modeller.

The needed parameters are the following ones:

D = inner diameter

d = oscillation diameter

R = D/d → reduction ratio

t = 0 : 360/(2*π) → angle in radiant

The set of generic equations is:

$$x(t) = (D + d) \times \cos(2\pi t) + \cos\left(\frac{2\pi t \times (D + d)}{d}\right) \qquad 2\text{-}15$$

$$y(t) = (D + d) \times \sin(2\pi t) + \sin\left(\frac{2\pi t \times (D + d)}{d}\right) \qquad 2\text{-}16$$

The reduction ratio was set equal to 1/9 and, for this reason, the cam has 9 lobes and the rods on the fixed base are 10. The base diameter is equal to 53 mm.

The equations become as follow:

$$x(t) = (23{,}850 + 2{,}650) \times \cos(2\pi t) + \cos\left(\frac{2\pi t \times (23{,}850 + 2{,}650)}{2{,}650}\right) \quad \textit{2-17}$$

$$y(t) = (23{,}850 + 2{,}650) \times \sin(2\pi t) + \sin\left(\frac{2\pi t \times (23{,}850 + 2{,}650)}{2{,}650}\right) \quad \textit{2-18}$$

The design cam is shown in Figure 2.16. The centre of the cam was designed to house a ball bearing while, on a diameter of 35 mm, six holes were needed to transfer the motion to the output shaft.

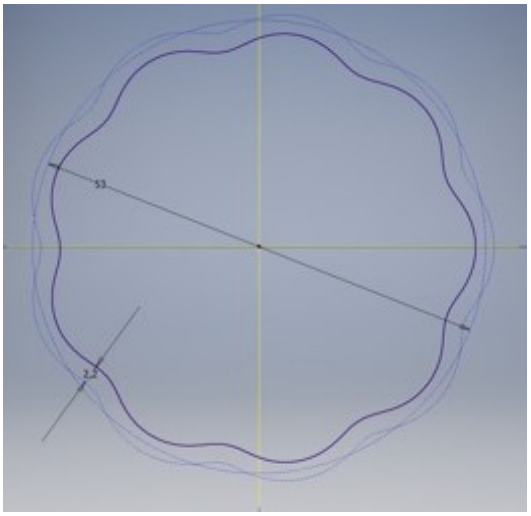


Figure 2.15 Cam profile

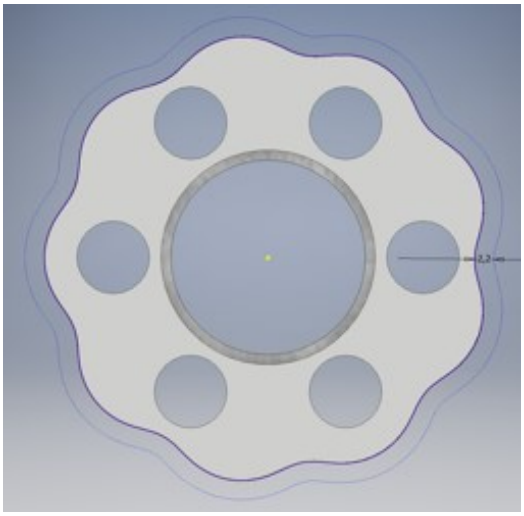


Figure 2.16 Cam

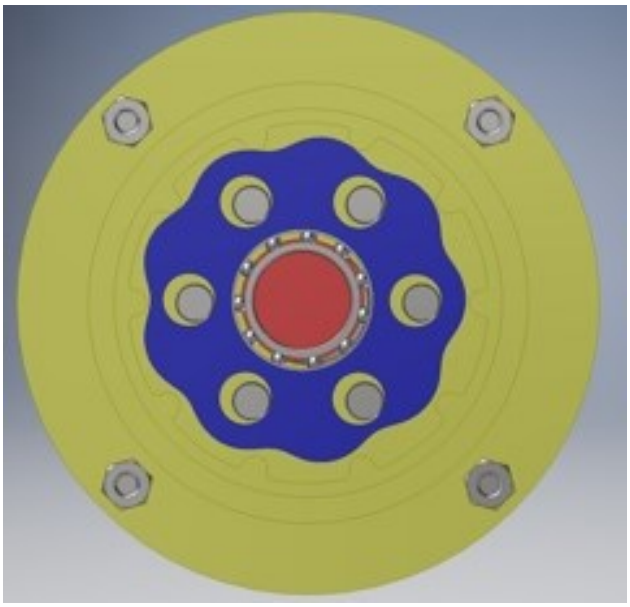


Figure 2.17 Cam view

2.3.3 Springs system

The 3rd main part of the VSA is the springs system, based on the principle analysed in previous chapters. It transfers the rotation from the actuator to the output shaft and it can be rigid or not. If the configuration is set to “rigid” the angular position of the actuator and of the output shaft are equal and the output is precise; on the other hand, if the configuration is “flexible” the output angular position can be different from the one of the actuator and the system can accept it without breaking.

The system was developed to house four leaf springs to increase the minimum stiffness up to a value that allows to reach a good deformation without incurring in torsion phenomenon described in Paragraph 2.2. Each leaf spring has the following characteristics:

H = 47 mm

L = 10 mm

s = 0,45 mm

By using the calculation of the ideal model (Paragraph 2.2), it was possible to identify the stiffness value of the single leaf spring:

$$K_{foil} = \frac{-1}{\frac{x^3}{6EJ} - \frac{Lx^2}{4EJ}} \qquad 2\text{-}19$$

$$K_{foil\_minimum} = \left(\frac{12EJ}{L^3}\right) = 1{,}8\ \mathrm{N/mm} \qquad 2\text{-}20$$

The minimum stiffness of the whole system can be calculated multiplying the just calculated value by the number of leaf springs because the springs work in parallel:

$$K_{tot\_minimum} = 4 \times \left(\frac{12EJ}{L^3}\right) = 7{,}23\ \mathrm{N/mm} \qquad 2\text{-}21$$

As already known, system stiffness upper limit can be approximated with infinite (rigid configuration); its trend is shown in Figure 2.18 in logarithmic scale.

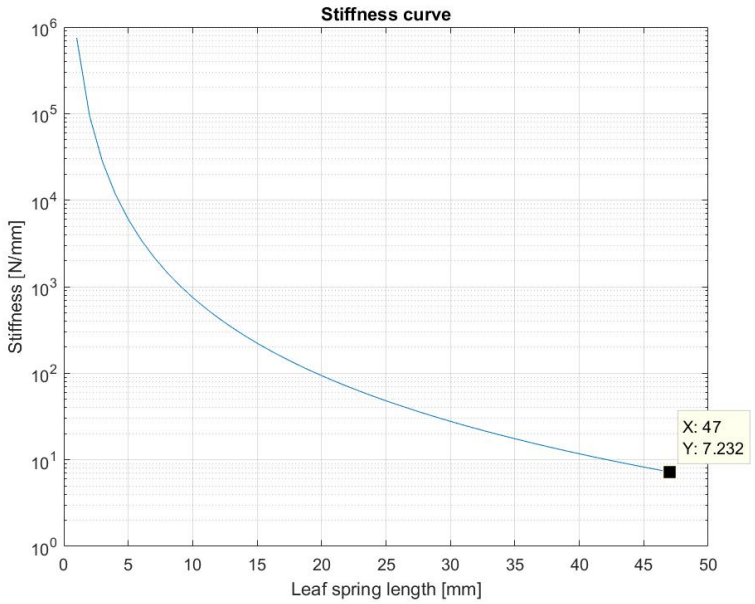


Figure 2.18 Stiffness plot

The calculation of the applied torque can be directly derived from Equation 2-13 in which values of displacement and stiffness are combined. The following step, to be taken, is the calculation of the displacement from the measured angle of rotation φ.

$$u(x) = R_a \tan \varphi \qquad 2\text{-}22$$

Ra is the distance between the rotation axis of the shaft and half the width of the steel foil as shown in Figure 2.19:

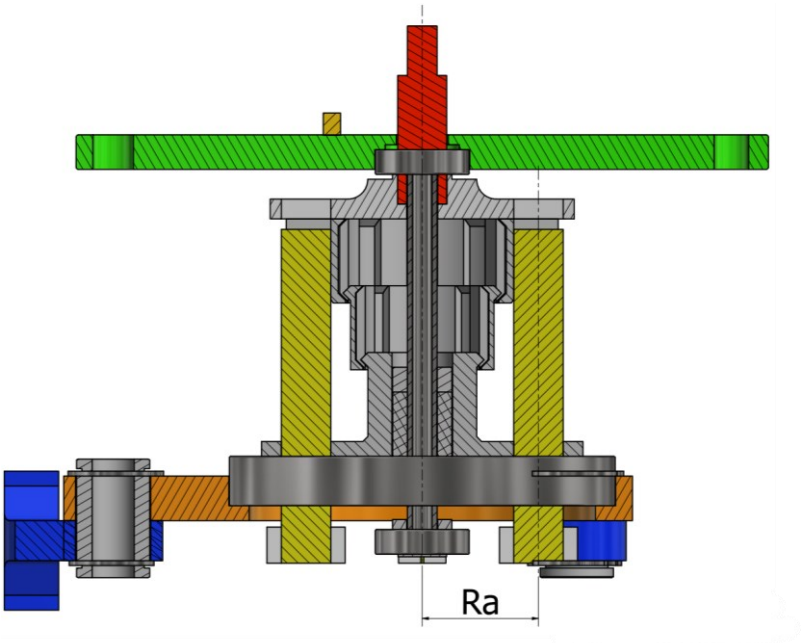


Figure 2.19 Ra

Rewriting Equation 2-13 and combining it with Equation 2-22, it is possible to calculate the applied torque directly from the measurement of φ:

$$C = \frac{4{R_a}^2 \tan\varphi}{\left(\frac{x^3}{6EJ} - \frac{Lx^2}{4EJ}\right)} \qquad 2\text{-}23$$

The spring system is shown in Figure 2.20 and it is possible to see that is connected to the structure through the upper base (6). The core of this module are the leaf springs (1) which are fixed at one end to the actuator rotating base through adaptable feet needed to allow easy replacements of the springs.

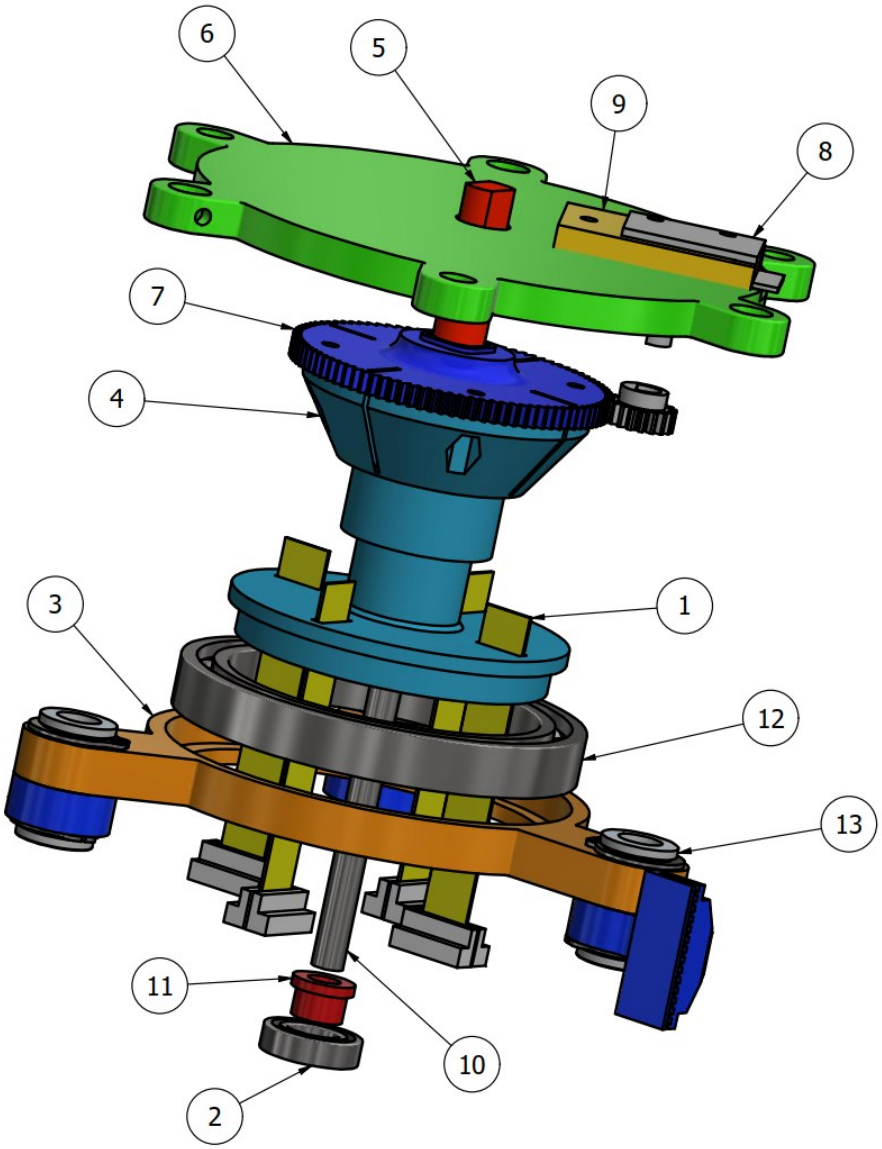


Figure 2.20 Springs system

Spring length can be modified thanks to the slider (3), held in position by three linear bearings (13) which slide over the three steel rods previously mentioned. The slider does not rotate and, for this reason, a ball bearing (12) has to be positioned between it and the telescopic shaft (4).

The shaft has a particular shape that allows to modify the length of the spring maintaining a low occupied volume. It is telescopic and is composed by three sections that, when needed, can go one into the other as shown in Figure 2.21 and Figure 2.22.

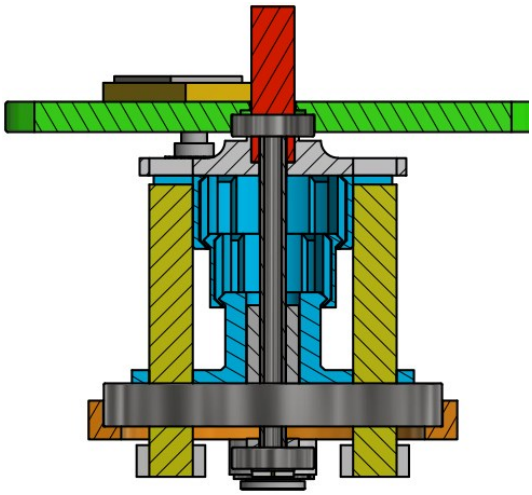


Figure 2.21 Maximum length of telescopic shaft

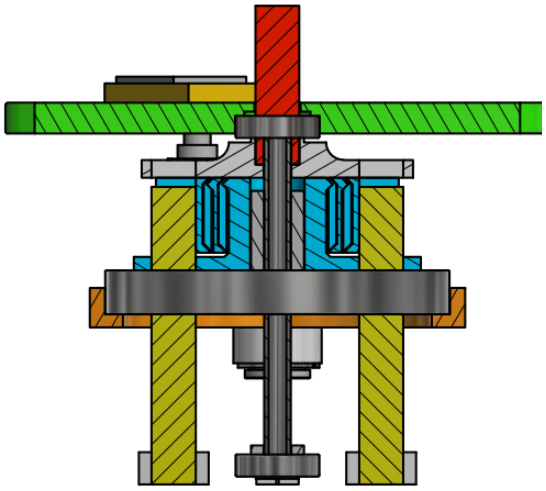


Figure 2.22 Minimum length of telescopic shaft

Figure 2.23 shows how the rigidity in rotation is possible. The three sections of the telescopic shaft are similar to gears because they have teeth that do not allow slipping between one section and the other.

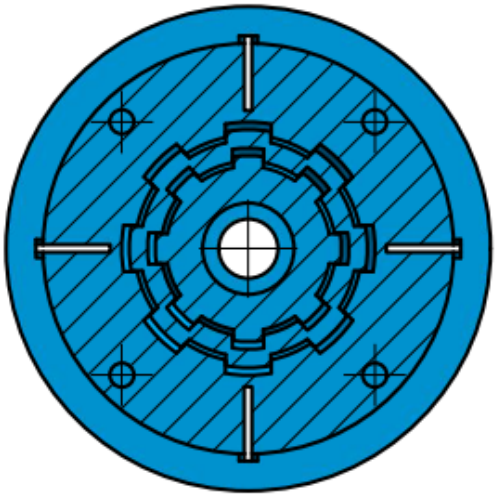


Figure 2.23 Telescopic shaft section

The outer section of the shaft is rigidly connected, through bolted joints, to a gear (7) that has a reduction rate of 50/10.5 and is needed to transfer the rotation to an encoder (8) which reads it. The gear has a second function: being a single part with the telescopic shaft, it maintains the uniaxiality because it is linked to the output shaft (5), through a square section, that is held in position by a steel rod of 6 mm of diameter (10).

To remove all the possible friction, a linear bearing is positioned between the inner section of the telescopic shaft and the steel rod and a ball bearing (11) is placed at the end of the rod to connect it with the actuator.

Between the rod and the ball bearing, there is an adapter (2) to maintain in position the rod and to compress and pre-load all the components connected to it.

2.3.4 Belt group

The last part of the design concerned the application of the device to modify the stiffness of the leaf springs. It is responsible for controlling the sliding of part 3 in Figure 2.20 and, therefore, of the length of the steel foils.

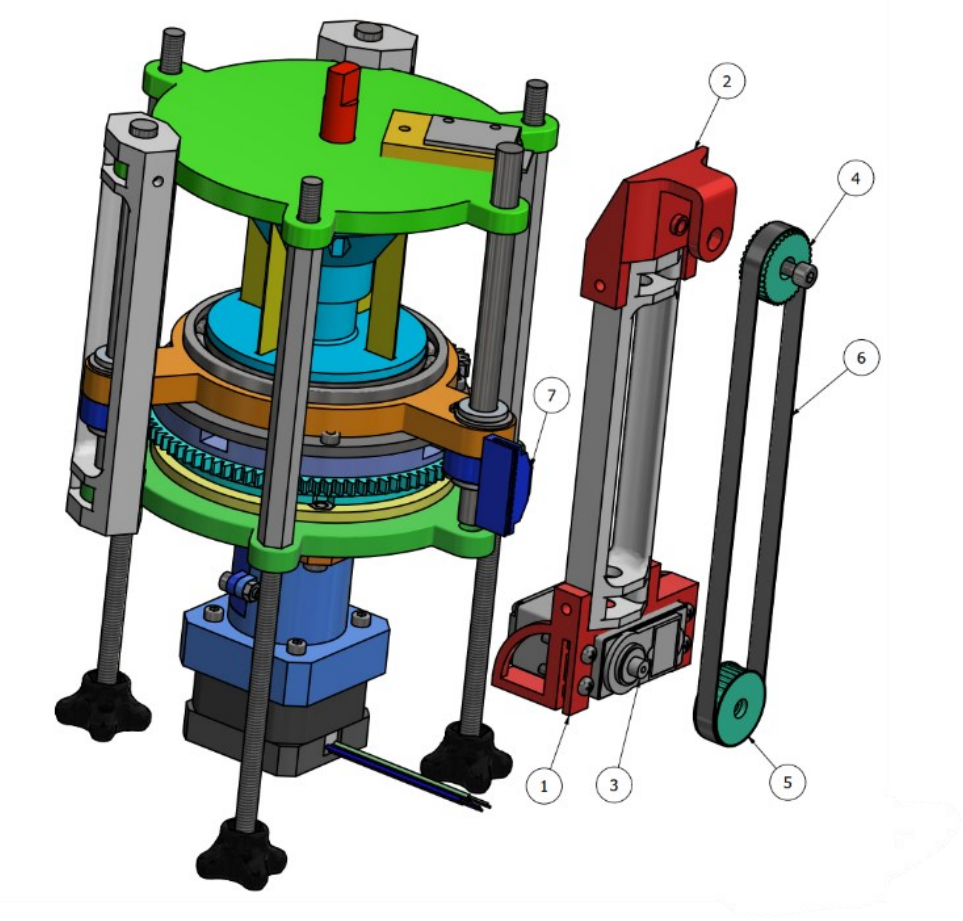


Figure 2.24 Belt group

The belt system (Figure 2.24) is applicated directly on the external structure of the device and, as for the other groups of elements, it is designed to be replaced or modified according to the needs of the application.

It is composed by two structural elements (1) and (2) used to define the distance between the two pulleys (4) and (5) and to maintain in position the servo motor (3) that actuates the belt (6). The sliding base is connected to the pulley through the belt tensioner (7).

The pulleys diameter was designed to avoid that the length of motion, to modify the stiffness of the spring system, needs a rotation of the servo axis greater than 180° which is its maximum angle of rotation.

3 Hardware, software and control design

The aim of the control strategy of the VSA was inspired by the behaviour of Cobots in presence of humans; in particular, the VSA should be flexible when there is the risk of external unpredictable interactions as well as rigid when needed. The project was developed to reduce to the minimum the number of sensors and to not have the necessity to make the machine perfectly aware of the external environment with ad-hoc programming.

Traditional robot most common task is “pick&place” and their rigidity is what ensures the best performance in case the environment is defined and there is no possibility of unpredicted interactions. Their programming is written to perfectly control the speed of motion or the torque but not at the same time. While rotating, they continue the motion even if meet an obstacle because they are not aware of the external environment if it is not programmed.

On the other hand, cobots working in flexible configuration can be controlled knowing both the speed and the acceptable force. The control calculates the external force, by measuring the difference between the input and output angle, and acts in relation to this data.

The thesis purpose was to make the VSA aware of the most common external interactions, such as hitting or being forced to move, and to generate the correct response. Obviously, it was impossible to cover all the risky circumstances but the easy setup, both mechanical and hardware, ensured that the programming of new tasks is very simple.

3.1 Hardware Design

The hardware design was developed with the purpose of connecting more devices at the same time, for this reason every aspect was deeply studied to find the most suitable hardware configuration. In Figure 3.1 the connection scheme is shown. In the following paragraphs each device is listed, and its connection is analysed.

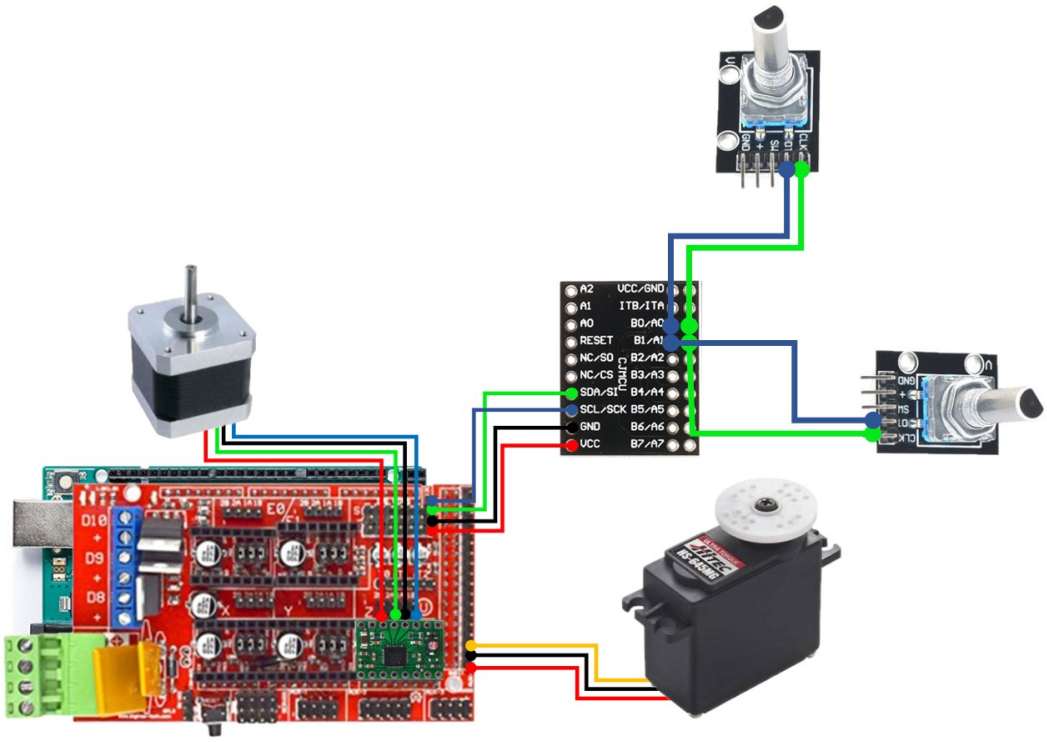


Figure 3.1 Electrical Scheme

3.1.1 Devices choice

As written in previous chapters, the device motion is based on the difference between the input and output measured angle. There was the need of sensors which continuously measure the rotation before and after the spring system, such as a “rotary encoder”. For this application, it was used a mechanical version called “contacting incremental rotary encoder”. It generates two quadratic signals which indicate position change and direction of rotation in relation to their modification priority. Figure 3.2 shows that the two signals are offset from one another of a quarter of a period [14].

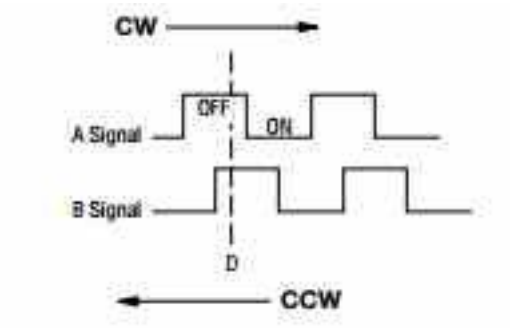


Figure 3.2 Encoder quadratic signals

The use of this kind of encoder was suitable for the project because the information about the direction of rotation is fundamental as well as the continuous measurement.

One encoder, shown in Figure 3.3, is placed on the output of the actuator and the other one on the output shaft of the spring system; the measurements are taken thanks to gear wheels placed in both the positions and, to have similar data, the reduction rates are theoretically equal.

Figure 3.3 Encoder

For what concerns the rotation, a stepper motor was the best choice because of its precision during the motion. The chosen model is a NEMA17 which is a standard motor with flange width equal to 42mm. Three different version of NEMA17 are present on the market and are diversified by the torque they can generate. Each version is applicable to the device but the chosen one has holding torque equal to 0,42 Nm and can reach the speed of almost 8000 full steps per second.

The rotation speed needed in this application is low in comparison to the top one shown in the trend, for this reason the working torque was approximated to the holding torque of the datasheet.

Motor control can be based on the number of steps to take, setting a specific rotational speed, or on the amount of time between each step. The first solution is the most basic and easiest one and it is how the standard control libraries work; on the other hand, the second one is more complicated to be developed but ensures that the motor always rotates at the

right speed. The control analysis and choice will be dissected later in the thesis (Paragraph 3.2) because a full picture of the system hardware is needed.

In addition to the stepper motor, also a driver was needed. It is the responsible of the sequencing of phases and of the amount of current every phase receives. For the thesis purpose, the driver A4988 was used.

The last device to be chosen was the motor to move the belt, it must have the torque needed to overcome the friction generated by the internal sliding. The motion length of the sliding base is 30 mm and the motor pulley has a diameter of 23 mm; for these reasons, the angle range of rotation, the motor has to provide, is not 360° but only 150°. The best option was a servo motor because its rotation can be easily limited and, usually, it is compact.

The used servo was an Hitec HS-645MG which has a torque of 0,75 Nm if powered by 4,8 V.

All the devices were powered by switched-mode power supply (SMPS). It is a power supply able to vary the voltage and the amperage in relation to the request it receives. For this application the voltage is constant at 12 V and the amperage is variable.

3.1.2 Devices connection

As previously written, the hardware design was developed to be able to control up to five devices which means a high number of sensors and actuators that works simultaneously.

The best set up involved the use of a microcontroller called Arduino Mega 2560 and a shield called Ramps 1.4.

Arduino is a controller with an easy interface which allows to connect many devices through its 54 digital and 17 analogical pins. It can be directly plugged via USB with the computer or, as in this application, it can be powered by an external power supply at 12 V.

If Arduino had been used alone, the hardware set up would have been disordered because of the amount of wires to connect everything with the control board; for this reason, the use of a shield was necessary.

A shield is a second board to be connected directly to Arduino and that reorganizes the position of the pins in relation to its purpose. The chosen shield has been developed to be used in 3D printers because it allows to connect to Arduino five plus one (5^{th} and 6^{th} ones are in parallel) stepper motors and their drivers as well as some servo motors and other kinds of sensor. It also has the predisposition to have all the motor drivers plug directly on the board without the use of wires.

In Figure 3.4 is shown a schematic of the Ramps shield and it is possible to see how the pins are organized.

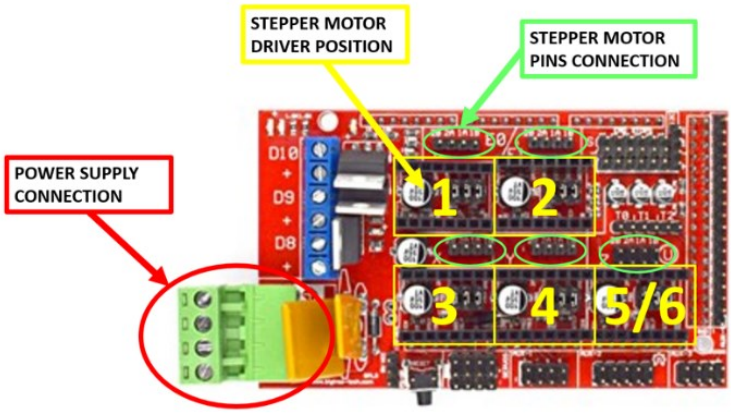


Figure 3.4 Ramps 1.4

In red is highlighted the connector of the power supply, in green stepper motors pins and in yellow the locations of stepper motor drivers.

Drivers are chips, to be plugged directly on the shield, which control a stepper motor motion; it is responsible for the enabling and disenabling of the motor and, moreover, it gives the right intensity of current every motor phase needs.

Drivers, like A4988 are not developed for a specific stepper motor but can be used with a large number of them. For this reason, they must be calibrated adjusting VREF by turning the screw shown in Figure 3.5.

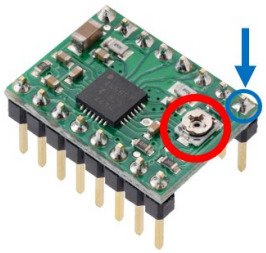


Figure 3.5 Driver A4988

The calibrating procedure is quite easy because it consists of reaching the right VREF measuring it between the ground pin (blue circle) and the screw (red circle) with a multimeter.

The value of VREF can be calculated with the following equation 3-1:

$$VREF = 0{,}75 \times \frac{(number\ of\ phase\ \times phase\ current)}{REF\ current}$$

$$VREF = 0{,}75 \times \frac{2 \times 1{,}5}{2{,}5} = 0{,}9\ V \qquad 3\text{-}1$$

Stepper rotation is generated by a sequence of signals sent to the two coils of the motor. Each coil has two connections to the driver, respectively 1A+1B and 2A+2B, and the way signals are sent through these wires generates the rotation and its direction.
The control action of the main board goes through pins DIR and STEP, while pins MS1, MS2 and MS3 are used to change the step resolution of the motor.

Figure 3.6 A4988 Pins

The resolution control has to be set-up manually, using jumpers, directly on the Ramps shield. How they are used allows or interrupts the connections of driver pins and the combination generates the chosen resolution. Figure 3.7 shows the possible combinations, “low” means that the jumper is not plugged in that pin. Step resolution goes from “full step” to “sixteenth step” and it affects the speed of rotation, the torque and the smoothness of the motion. Speed decreases from full to sixteenth, torque trend is the same of the speed, but the reduction is slight and smoothness increases reaching sixteenth step.

MS1	MS2	MS3	Resolution
LOW	LOW	LOW	Full Step
HIGH	LOW	LOW	Halft Step
LOW	HIGH	LOW	Quarter Step
HIGH	HIGH	LOW	Eighth step
HIGH	HIGH	HIGH	Sixteenth Step

Figure 3.7 Step Resolution

In Figure 3.8 it is possible to see the connection between the shield and the stepper and where the driver is positioned.

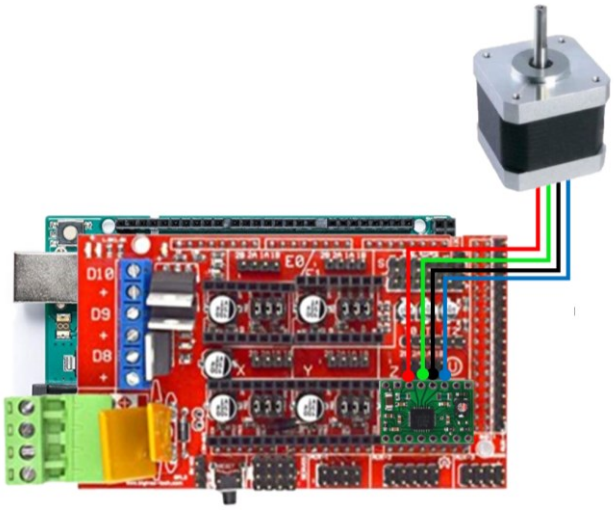


Figure 3.8 Connection between Ramps 1.4 and Stepper

The second motor, to be connected to the board, was the servo. Ramps 1.4 (Figure 3.9) provides four positions for servos (green) but they could not be used in this specific application. This was due to synchronization problem generated by both hardware set-up and software coding. To overcome the problem, there was the need to plug the servo motor in other available pins: they are shown in orange and are GND, 5V and D32 starting from below.

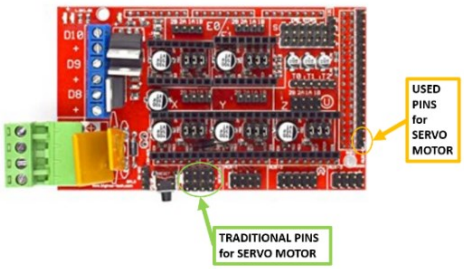


Figure 3.9 Servo Motor Pins on Ramps

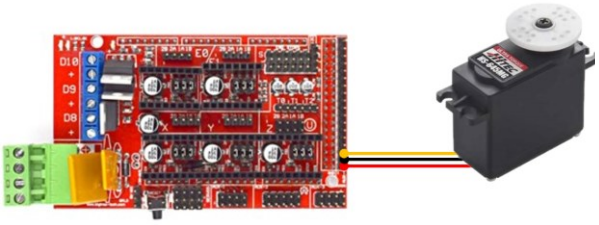


Figure 3.10 Between Ramps 1.4 and Servo

The last connection was the more complicated one. The encoders could not be plugged directly to the control board because a single encoder needs at least four pins to be powered and read, every device required two encoders and the hardware set-up had been thought to control up to five devices. This means that 40 pins must have been connected to the shield which, clearly, did not have so many connections available after the application of motors.
The solution was found with an arrangement that allowed to separate the encoders from the control board and to use only four wires to connect all the sensors. Nevertheless, this solution moved the problem from the hardware to the software control and it complicated the communication because it must follow a rigid protocol called I2C.

The connection pins for a I2C port expander are the ones shown in Figure 3.11 which are GND, 5V, PIN D20 and PIN D21 of Arduino board. The port expander is a model called MCP-23017, shown in Figure 3.12, and it is connected to the shield as Figure 3.13 shows.
This kind of expander allows to connect as much as 8 devices with the possibility to use up to 8 expanders at the same time. This means that all the encoders could be connected with only four wired plugged in the main control board.

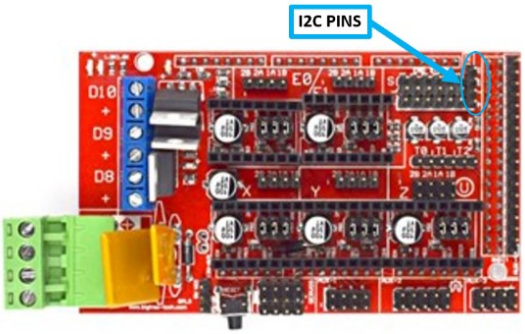


Figure 3.11 I2C Pins on Ramps

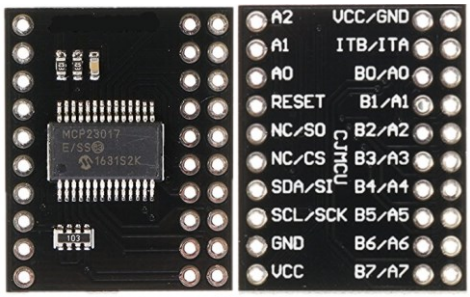


Figure 3.12 MCP-23017 Port Expander

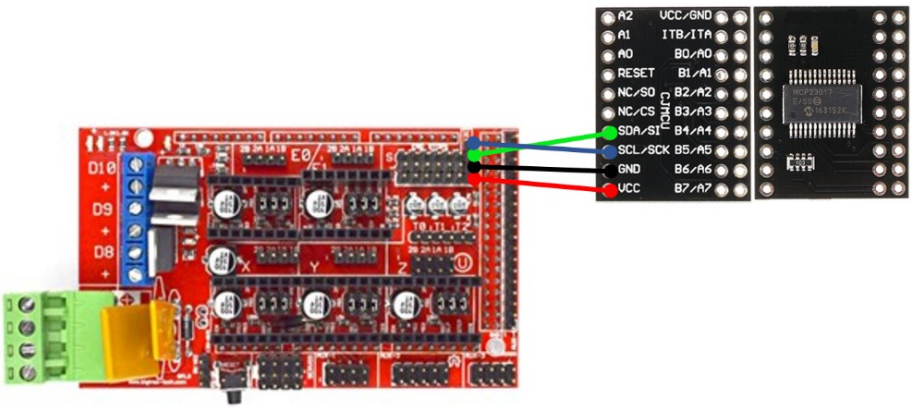


Figure 3.13 Connection between Ramps 1.4 and MCP-23017

Going in detail, the wiring of the encoders to the MCP-23017 was done with the use of an external breadboard. As it can be seen in Figure 3.14, the connection to the control board was set-up with the use of only four wires which helped to neatly arrange the wiring in case the number of sensors would have increased. Additional encoders' power wires can be plugged directly on the breadboard while data wires can be connected in the available spots of MCP-23017. As general rule, "A" row of the port expander was used for clock wire and "B" row for data wire even if, as it will be shown in software design chapter, "clock" and "data" are just labels to describe the wires and do not have the usual meanings.

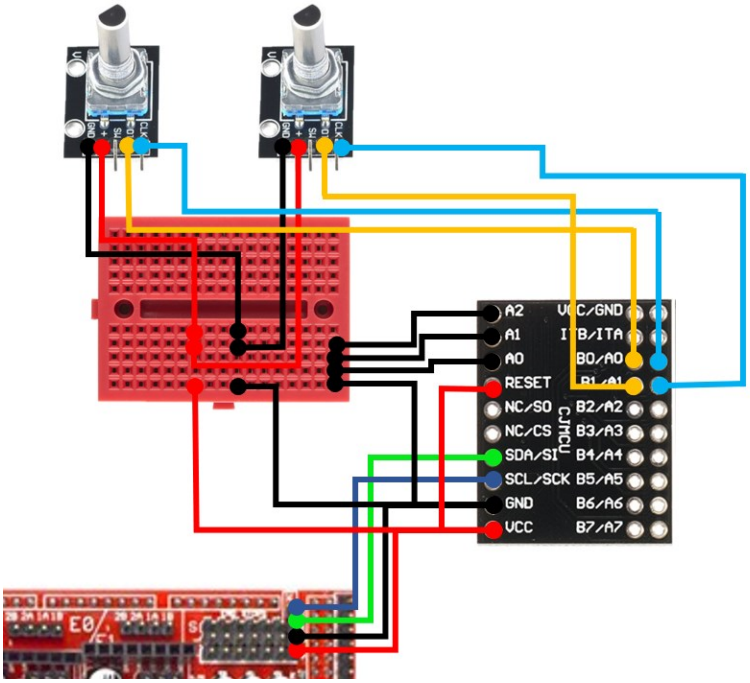


Figure 3.14 MCP-23017 and Encoders Wiring

Figure 3.15 shows that multiple port expanders must be connected together in parallel because SDA and SCL wires must be synchronized to ensure the communication.

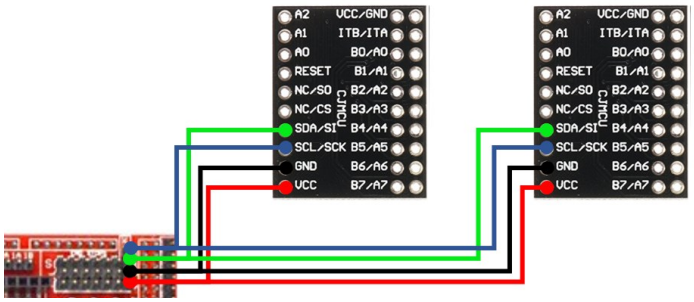


Figure 3.15 Multiple MCP-23017 Wiring

As previously stated, the set-up of the hardware was developed to connect and control as much as five devices but, actually, it was realized with only one because the thesis purpose is the merely study of a VSA module and not its multiple application on a cobot.

3.2 Software Design

The control design was developed to allow the VSA to behave as its application demands. In other words, it receives information from the sensors, analyses it and sends outputs to the actuator to generate the correct reaction to the external interaction.

“External interaction” means every possible situation in which the device may come into contact, directly or indirectly, with the environment including both humans and other machines. It is clear that it was impossible to cover all the scenarios, and, for this reason, the chosen approach was a control strategy which allowed to adapt to most of the situations but that was not specific for a particular application.

More precisely, it was taken into account a possible obstacle that forces or stops the rotation of the output shaft of the device.

The aim of the control was to react to a force in relation to the estimation of its intensity, computed with Equation 2-23, after the measurement of the rotation of each encoder. The reaction must have been converted in a signal understandable by the stepper motor and, then, it must have been sent to be performed.

Each calculation, taken individually, was not a problem to be carried out by the chosen control board but, performing them together and at a frequency high enough to ensure an acceptable performance, it might overload the calculation generating overall performance worsening. Furthermore, most of the present cobots are designed to work plug to an external computer.

These reasons helped to define how to design the control algorithm for the device under study and to develop it in an environment born to manage the communication between different typologies of calculators.

The control includes a code in native Arduino IDE to be run by Arduino control board, a code in a more general C++ language to be run by the computer itself and a series of commands to be run directly in the terminal windows thanks to a system called ROS.

ROS is a system not entirely supported by Windows but only by Linux based operating systems among which Ubuntu is the most used for robotics.

Before analysing actual codes, a brief explanation is needed about the working principle of ROS to understand what it is and how it manages different boards which exchange information and about the communication protocol used to connect the encoder with the main board.

3.2.1 ROS Environment

ROS, abbreviation of Robot Operative System, is software platform that simulate an operative system for robots without the use of virtual machine programs. It provides basic services to manage and control the activities related to robots thanks to a series of easily installable packages including libraries and tools to write, build and run code across multiple control boards (computer and other kind of controllers such as Arduino). It connects the board and the computer via serial communication through a USB cable.

For the thesis purpose, the most useful application of Ros is the ability to synchronize Arduino and the computer and to behave as the communication manager between them.

The working principle is simple: every code, called node from now on, is seen as an entity that can publish and receive message on and from ROS itself. The shared information is called topic and can be updated by a node or by a command written in the terminal window.

Communications between ROS and nodes are enabled by the creation of objects that state which information has to be read or published. A "publisher" is needed when topics must be sent from a node to ROS and its general structure is shown in Code 3.1. The following example is written in the language of Arduino IDE and, thus, sentences may be different if the language varies but the defining structure remains the same.

```
1   #include <ros.h>
2   #include <std_msgs/Int16.h>
3   ros::NodeHandle  nh;
4   std_msgs::Int16 msg1;
5   ros::Publisher p1("en1", &msg1);
6   void setup() {
7       nh.initNode();
8       nh.advertise(p1);
9   }
10  void loop() {
11      msg1.data = 10;
12      p1.publish(&msg1);
13  }
```

Code 3.1 Publisher Example

The “publisher” has been created in line 4 and 5 and set-up in line 8. Its value is declared in line 11 while the publishing is shown in line 12. Other lines are needed to complete the setting of the node.

On the other hand, a “subscriber” is the object that allows to read the value of a topic from ROS. A general example of subscriber declaration is shown in Code 3.2.

```
1   #include <ros.h>
2   #include <std_msgs/Int16.h>
3   ros::NodeHandle  nh;
4   std_msgs::Int16 msg1;
5   void CallBackFcn(const std_msgs::Int16 msg1) {
6       value = msg1.data;
7   }
8   ros::Subscriber<std_msgs::Int16> message("msg", &CallBackFcn);
9   void setup() {
10      nh.initNode();
11      nh.subscribe(reset);
12  }
13  void loop() {
14  }
```

Code 3.2 Subscriber Example

Every subscriber needs a “CallBackFunction” (line 7-8-9) which recalls the topic value from ROS and assigns it to a variable so that the value can be used inside the main code. The actual declaration of the subscriber is written in line 10 where it has been stated the type of message, the ROS name of the topic to be recall and the call back function.

3.2.2 I2C Protocol

I2C is a synchronous serial communication protocol widely used to connect multiple secondary devices, called slaves, to one or more main devices, called masters. The physical implementation is simple because data transfer is possible with only two wires connected to the master. The communication is performed using a bus of data that can be of 7 or 10 bits.

7 bits addressing allows to connect up to 112 devices and it is the typology used for the control of the system under study.

Each connected device is characterized by a unique address that allows the master to choose with which device it exchanges data.

The wires of the connection have two different tasks: one is called Serial Clock (SCL) and it is responsible to the synchronization of the data transfer between the master and the slaves; the other one is called Serial Data (SDA) and it is in charge of data transfer [15].

The data is transferred in the sequence of bits shown in Figure 3.16.

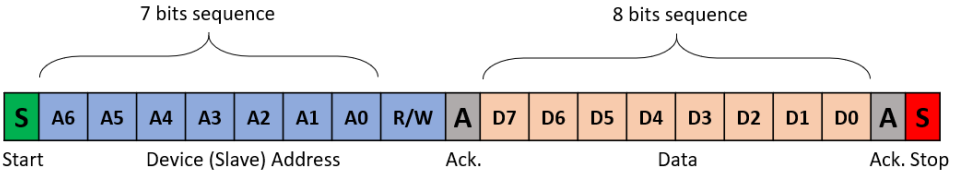


Figure 3.16 I2C Bus

It is possible to split the sequence in an addressing part which has a starting bit, a sequence of 7 bits that identify the device address, a bit that identifies if the slave has the permission to read or write and an acknowledge bit and in a data part which transfers the data in packs of 8 bits divided by an acknowledge bit. At the end there is always a stopping bit. Usually A6 bit is the MSB (Most Significant Bit), A0 is the LSB (Least Significant Bit), R/W bit is the "read or write" bit and acknowledge bit is used by the slave to indicate if it has received the previous sequence of bits.

The rhythm, at which every data bit is sent, has been defined by SCL wire that notifies the pulsing of the clock of the master device.

As previously mentioned in Paragraph 3.1.2, encoders are connected to Arduino control board through the MCP-23017 port expander which works thanks to I2C protocol. It is wired with SCL and SDA lines and transfers the data with two packages of 8 bits each. It has 2 ports of 8 bits each that allow to control 16 terminals which can be inputs or outputs.

This setting allows to reduce the number of I2C addresses because only MCP-23017 address has to be declared and there is no need to have one address for each encoder, as it would have been if the encoders had been directly connected.

I2C communication uses a serial port which is different from the one used to connect Arduino and the computer and, for this reason, the information exchange between encoders and ROS is not direct. Arduino reads encoders data from the I2C bus and publishes it as topic in ROS. This way of communication allows to handle the whole I2C network directly through Arduino code itself without the need of managing it in one of the other ROS nodes.

3.3 Control Design

Control design has been developed with the intention of clearly separating the tasks of each nodes. There are two kinds of control to deal with: the first one is related to the hardware and it sets-up the physical communication between the control board and the sensors/actuators; the second one manages the information flow and generates the response sent to the control board. They are both crucial to the control aim but the approaching method is very different because the former is completely technical and requires strict rules while the latter accepts different coding solutions.

Arduino is responsible for controlling how external devices, as sensors or actuators, send and receive data. It is controlled by a code, written directly from its own software with a language based on C++, which represents the first node of ROS. The second node is a C++ code, processed by the computer, which receives data from ROS, analyses it through the equations of Paragraph 2.3.3 and sends back the correct response to ROS. These two nodes manage every aspect of the hardware and mechanical control, but they cannot work without external inputs which represent how the motion command would have sent.

These parameters are declared, via terminal window, as ROS topics to reduce the complexity of the prototype software but they could be integrated directly in a more elaborated robot control algorithm. Two of them are motion inputs and the third one is a control needed to reset data values.

The topic called “joystick” enables the motion and declares the direction of rotation. If it is equal to “0”, no inputs are sent to the stepper motor; if it is equal to “1”, the rotation starts in clockwise direction and if it is equal to “-1”, the rotation is counter-clockwise. The following Code 3.3 shows how it is sent via terminal window in all the possible configurations:

```
rostopic pub -1 /VSA0/joystick std_msgs::Int16 1

rostopic pub -1 /VSA0/joystick std_msgs::Int16 0

rostopic pub -1 /VSA0/joystick std_msgs::Int16 -1
```

Code 3.3 Joystick publication

The second topic is called “pot_read” and simulates the action of a potentiometer. It is used to change the position of the servo motor and, therefore, to change the value of the stiffness. It can assume a value between 0 and 1000. This is how it is sent:

```
rostopic pub -1 /VSA0/pot_read std_msgs::Int16 500
```

Code 3.4 Pot_read publication

The third one is called “res” and it is a Boolean variable whose value can be 0 or 1. When it is sent to ROS and its value is different from 0, encoders measurements restart from 0. This procedure has been developed mainly to reset the values immediately after the code starts running in order to have an ideal initial situation. The declaration is shown in Code 3.5:

```
rostopic pub -1 /VSA0/res std_msgs::Int16 1
```

Code 3.5 Res publication

After the analysis of those parameters whose way of generation does not affect the system, it is possible to start the dissection of each nodes and of how they manage information exchange with other devices.

The first step consists into defining how Arduino commands the stepper motor followed by the reading procedure of encoder signals. The analysis proceeds to show how the motion is calculated from the input measurements and how the stiffness is modified. The last part regards the explanation of the algorithm reaction to an external interaction.

3.3.1 Standard motion

The motion of device is mechanically generated by the stepper motor. As already written in Paragraph 3.1.1, stepper motor is controlled by a driver board which manages every phase signal sequence. This process is controlled directly by Arduino board and, consequently, it is managed by the 1st ROS node.

Arduino coding is usually developed with the use of libraries that identify standard objects. The wide range of application of this control board allows to have a great number of libraries already written and, as obvious, a library to control a stepper motor has been already developed.

This library allows an easy control of the stepper, but it has some unacceptable drawbacks. First and greater problem is that the rotational speed can be set only when the board is resetting which means that it is not possible to change the speed during the motion as every robot controller should allow. Moreover, if the rotation starts, it must be completed before another control can be read by the motor. This aspect can be seen as an advantage but, for the thesis purpose, it is a great issue because it is in contrast with the safety requirements stated at the beginning of the study.

Clearly, if a standard library cannot be used, the coding needs to involve also the basic hardware features of the board such as interrupts and timers.

Interrupts are the way Arduino manages priority among routines devices. Usually an external device can ask for Arduino "attention" sending a signal through an Interrupt Pin. This kind of pins is physically present on the board, but it can also be simulated by the code. The presence of a signal on an Interrupt pin means that control board suspends the standard routine, no matter what it is processing, and starts the required action. In particular, Arduino Mega has six interrupt pins: D2, D3, D18, D19, D20, D21.

If a part of the routine becomes so crucial that it cannot be stopped, it is possible to disenable the action of interrupts. Code 3.6 shows how:

```
1    void setup() {}
2    void loop() {
3    noInterrupts();
4        // critical, time-sensitive code here
5    interrupts();
6        // other code here
7    }
```

Code 3.6 "noInterrupts" Routine Arduino

In lines 3 and 5 are sentenced two functions which disenable (3) and re-enable (5) interrupts action. The key routine must be written between these two functions to elevate it to the absolute priority.

More in depth of Arduino construction, it is possible to manage timers which are chips built-in the hardware that can be used to measure time events. Arduino Mega has six timers and they can be configured independently one from each other. Timer0 and Timer2 are 8bit timers and the other are all 16bit timers. The difference is the maximum possible resolution because 8bit timers have 256 while 16bit timers have 65536 values [16]. The main task of a timer is to count an amount of clock pulses to synchronise external devices with the main board.

There was the need to set Timer3 to be followed by the stepper motor routine because Timer1 was already busy by the servo motor standard library.

For the thesis purpose, Timer3 prescaler was set equal to 8 which means that the stepper would had been controlled every 8 clock pulses.

Arduino Mega works with a frequency of 16MHz and thus the stepper receives information every 0,5 µs as shown inn Equation

$$f_{Timer3} = \frac{working\ frequency}{prescaler\ value} = \frac{16000000}{8}\ Hz = 2000000\ Hz$$

$$dt = \frac{1}{f_{Timer3}} = 0{,}5\ \mu s \qquad 3\text{-}2$$

In order to set the prescaler with the correct value it was used the parameter called CS31 as shown in line 5 of Code 3.7

```
1    TCCR3A = 0;
2    TCCR3B = 0;
3    OCR3B = 30;
4    TCCR3B |= (1 << WGM32);
5    TCCR3B |= (1 << CS31);
6    TIMSK3 |= (1 << OCIE3B);
```

Code 3.7 Timer3 setting

Line 1 and 2 are the Timer/Counter Control Registers and were needed to control the prescaler. The OCR3B is an Output/Counter Register and its value is compared with the maximum value the resolution of the timer allows. Line 4 sets the mode to configure the timer and setup the interrupts. The most common mode is called CTC (Clear Timer on Compare Match) and it is the one used.

To control the stepper motor, a standard Arduino routine was used. It was recalled when an output compare match occurs and it is shown in the following code:

```
1   ISR(TIMER3_COMPB_vect) {
2       TCNT3  = 0;
3       if (count1_stepper[2]!=0){
4           step_motor();
5           if (sign_data[2] > 0) digitalWrite(motZ[1], LOW);
6           if (sign_data[2] <= 0) digitalWrite(motZ[1], HIGH);
7           digitalWrite(motZ[0], digitalRead(motZ[0]) ^ 1);
8           counter_stepper[2] = counter_stepper[2]+1;
9       }
10  }
```

Code 3.8 ISR Routine

Stepper motor rotation is strictly related to the measurement of the encoders, as it will be shown later. Encoders, as the one shown in Paragraph 3.1.1, are widely used and it is possible to find libraries to control their readings but, once again, it is not possible to incorporate them due to technical incompatibility.

The analysis to codify encoder control starts from the dissection of the four possible combinations of signals. The encoders generate two quadratic signals, their value can be 1 or 0 and, thus, the combinations are 11, 10, 01 and 00 [14], as shown in Figure 3.17.

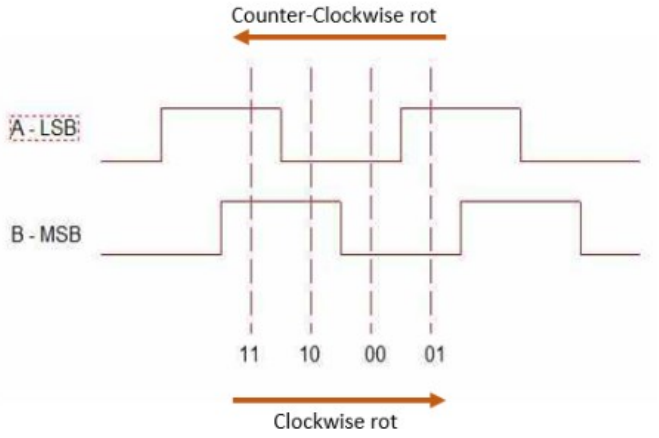


Figure 3.17 Encoder Rotation Combinations

To understand the rotation direction, the sequence of combinations must be investigated. If the encoder is rotation in clockwise direction the sequence is: 11 → 10 → 00 → 01 → 11 →….

While if the rotation is counter-clockwise the series will be the opposite of the previous one: 01 → 00 → 10 → 11 → 01 → ….

As explained in Paragraph 3.1.2, encoders are connected to MCP-23017 serial port A and B through two wires called, for simplicity, CLK and DATA. The following code highlights how the encoders are set-up to allow the control to work properly.

```
1    const int encCount0 = 2;          // number of rotary encoders
2    int encSelect[encCount0][2] = {
3        {101, 0},
4        {101, 0}
5    }
6    const int encPins0[encCount0][2] = {
7        {0,8},                        // enc:0 → GPA0,GPB0 → 0,8
8        {1,9}                         // enc:1 → GPA1,GPB1 → 1,9
9    }
```

Code 3.9 Encoder initialisation

During setting-up phase, the 1st needed information (line 1) is the number of encoders connected to the device. This data configures the number of rows of the matrixes initialized in lines 2 and 6. Line 2 matrix shows if the encoder rotates and in which direction and, thus, it is a dynamic matrix whose elements are changed by the routine. Line 6 matrix has fixed elements which state the input pins of each encoder.
The control has been developed from this data as the following code shows:

```
1   unsigned char readEnc(Adafruit_MCP23017 mcpX, const int *pin,
                          unsigned char prev, int encNo) {
2       unsigned char encA = mcpX.digitalRead(pin[0]);
3       unsigned char encB = mcpX.digitalRead(pin[1]);
4       if((encA!=prev)) {
5           encSelect[encNo][0] = encNo;
6           if(encB!=encA) {
7               encSelect[encNo][1] = 1;  // counter-clockwise
8           }
9           else {
10              encSelect[encNo][1] = 2;  // clockwise
11          }
12          change[encNo]=true;
13      }
14      return encA;
15  }
```

Code 3.10 Encoder Reading Function

The function shown in Code 3.10 explains how the behaviour of the encoders signals is analysed: first of all the value of the signals is read (lines 2-3) from the port expander pins stated in the 2nd matrix of Code 3.9; secondly the value of pin A is compared to its previous value (line 4) to understand if a rotation happened; if it is true, the values of pin B is confronted to the one of pin A (line 6). If they are different the rotation is considered counter-clockwise, otherwise clockwise.

The routine to check the status of the encoder signals, shown in Code 3.10, is derived from the previously stated method but it has a simplified approach to understand if the encoder rotates. Instead of checking both signals each time the function is recalled, only pin A is checked which means the possibility to skip the remaining part of the routine if the new value is equal to the previous one.

When the routine understands that one of the encoders rotates, the main function modify the value of its counter in relation to the direction of rotation (line 5-8) as Code 3.11 shows.

```
1   if (change[0] == true) {
2       if (encSelect[0][0] < 100) {
3           switch (encSelect[0][1]) {
4               case (1): // counter-clockwise
5                   jj++;
6                   break;
7               case (2): // clockwise
8                   jj--;
9                   break;
10          }
11      }
12      encSelect[0][0] = 101;
13      change[0] = false;
14  }
```

Code 3.11 Rotation counter of encoder

The main function of the 1st node is shown in the following code:

```
1   void loop() {
2       currentTimeMot = millis();
3       encoder();
4       if (currentTimeMot >= (loopTimeMot + delayTimeMot)) {
5           msg1.data = jj;
6           p1.publish(&msg1);
7           msg2.data = kk;
8           p2.publish(&msg2);
9           nh.spinOnce();
10          loopTimeMot = currentTimeMot;
11      }
12  }
```

Code 3.12 Main function Arduino node

In order to reduce the number of publications and to avoid that it happens at every clock pulse, the “if” clause between line 4 and line 10 will be true every time period stated by the variable “delayTimeMot” which is declared at the beginning of the script.

When the topics are sent to ROS, the 2nd node receives and analyses them to understand what is happening and to reply in the correct way.
This node is responsible for the movement logic. The routine to only active the rotation is an "if" clause whose task is the definition of the number of steps the motor has to perform every time cycle. The sign of the steps number declares the rotation direction.

```
1    if (joystick<0) {
2        jR = -5;
3    } else if (joystick>0) {
4        jR = 5;
5    } else {
6        jR = 0;
7    }
```

Code 3.13 "If" clause Number of Steps

Code 3.13 shows the routine of the "if" clause and it is possible to see that the specified condition is related to the value of the external topic "joystick" described at the beginning of this Paragraph.
The variable "jR" is the number of steps per period. Its value is equal to 5 because it ensures a trade-off between torque and speed of rotation, but it can be different in relation to the task. The last passage to make Arduino aware of the motion order is the publication of the variable as a ROS topic.

3.3.2 Stiffness regulation

The definition of the standard motion was only the first step to realize the logic to control the VSA, a further step was the implementation of routine to modify the stiffness of the leaf springs. As written before, the stiffness modification was allowed by a servo motor which controls the belt. Arduino is responsible of the signal sending towards the device while the other node calculates the angle of rotation in relation to "pot_read" topic value.
In order to control the servo motor a standard library was used, as shown in line 2 of Code 3.14.

```
1		pot_angle= (unsigned long) pot_angle;
2		myservo.write(pot_angle);
```

Code 3.14 Servo Motor control

The servo library is set on the first Arduino timer by default, and this is the reason why stepper motor was configured on the third timer.

"Pot_angle" variable is a ROS topic published by C++ node which represents the angle needed by the servo to complete the correct rotation. To correctly set the range of angles, a characterization of the servo motor was carried out. The results showed that, in order to achieve the rotation to change the leaf spring length between the limits, the servo needs to rotate from 20 degrees to 145 degrees. The following code is the function which converts the value of "pot_read" in degrees of rotation:

```
1		int flessione(int pot_read, double en_diff) {
2			pot_angle= 20 + pot_read * 125/1000;
3			return pot_angle;
4		}
```

Code 3.15 "Flessione" Function

The main routine is shown in Code 3.16 and it is possible to see that "pot_angle" is sent to Arduino in line 2. In addition to its direct utilisation, this variable is used to calculate the value of "delta_angle" which is the maximum difference between the encoders measurements.

```
1		pot_angle = flessione(pot_read,en_diff);
2		msg6.data = pot_angle;
3		if (pot_angle <145/3) {
4			delta_angle = 5;
5		} else if (pot_angle <2*145/3) {
6			delta_angle = 6;
7		} else {
8			delta_angle = 7;
9		}
```

Code 3.16 Use of "pot_angle"

For the sake of simplicity, “delta_angle” can assume only three integer values but they ensure that the system overcomes the maximum difference value only if an external interaction happens. The following code shows how the main routine has been modified in order to include the stiffness modification. Lines 1 and 2 allow to calculate the rotation degrees of the shafts from the values measured by the encoders. Line 3 calculates the difference between the measurements.

The “if” clause from line 4 to line 10 compares the difference with its maximum value and, if it is less, sends “jR” variable to Arduino node otherwise sends zero. In the first case the stepper moves while in the second it stays still.

```
1     en1_angle = en_to_degree * en1_to_shaft * en1_val;
2     en2_angle = en_to_degree * en2_to_shaft * en2_val;
3     en_diff = en1_angle - en2_angle;
4     if (jR != 0){
5         if (abs(en_diff) < delta_angle) {
6             msg4.linear.z = jR;
7         } else {
8             msg4.linear.z = 0;
9         }
10    }
```

Code 3.17 Main routine for movement in case of stiffness variation

3.3.3 Collision reaction

The routine shown in Code 3.17 was the base of the development of the collision reaction control. It was developed to respond to an external interaction which stops or forces its rotation. The control is related on the difference between the measurement of the encoders and manages the response according to its value. The following Code 3.18 shows the final routine.

```
if (jR != 0){
        if (abs(en_diff) < delta_angle) {
            msg4.linear.z = jR;
        } else if (abs(en_diff) > 1.35*delta_angle) {
            msg4.linear.z = inseguimento(en_diff);
        } else {
            msg4.linear.z = 0;
        }
} else {
        if (abs(en_diff) > 1.35*delta_angle) {
            msg4.linear.z = inseguimento(en_diff);
        } else {
            msg4.linear.z = 0;
        }
}
```

Code 3.18 Final Main routine for collision reaction

This routine is an update of the "if" clause shown in Code 3.17. It can be seen that the code is divided in two parts: the former is active when a motion signal is sent through ROS and the latter when there is no motion input.

The main difference with the previous routine is the inclusion of a condition to be followed if the measured difference is higher than 1,35 of the maximum stated value. If this happens the following function is recalled:

```
int inseguimento(double en_diff) {
        int data = -3*en_diff/abs(en_diff);
        return data;
}
```

Code 3.19 "Inseguimento" Function

This function is needed to allow the stepper to rotate in the correct direction if an external interaction forces the device to move.

4 Prototype production and testing

The results of the design were considered as acceptable as it was chosen to physically realize the device. As the guidelines stated, the manufacturing method was the 3D printing because it is cheaper than other machining, it allows more complex shapes and, due to the material used, the final product is very light.

4.1 Physical realization

In order to produce the device, it was used a 3D printer called Zortrax M200 which is able to work with a tolerance in the range of a 10^{th} of millimetre.

To reduce the printing time, every part was analysed in terms of needed precision and structural strength. In particular, parts as the gear wheels or the cam were printed with high precision to reduce to risk of unwanted interferences (Figure 4.1 and Figure 4.2) even if the printing time was high in relation to the size of the parts. The setting parameters were 0,09 mm of layer high, 60% of infill and low printing speed.

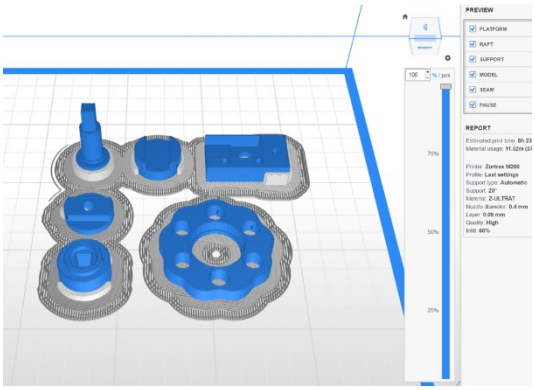


Figure 4.1 0.09mm 60% high

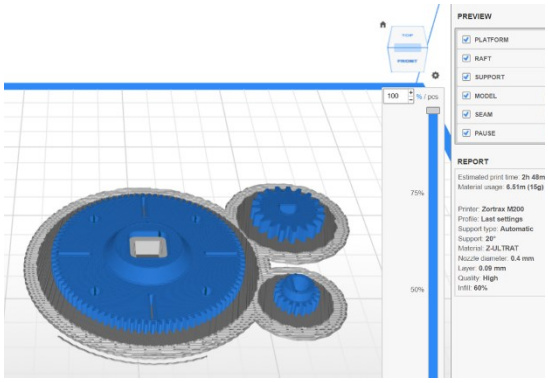


Figure 4.2 0.09mm 60% high

Other parts as the base plate or the motor case which were structural components but did not require high precision, were printed faster and with less infill percentage (Figure 4.3 and Figure 4.4). The parameters were 0,14mm of layer height, 40% infill and fast printing speed.

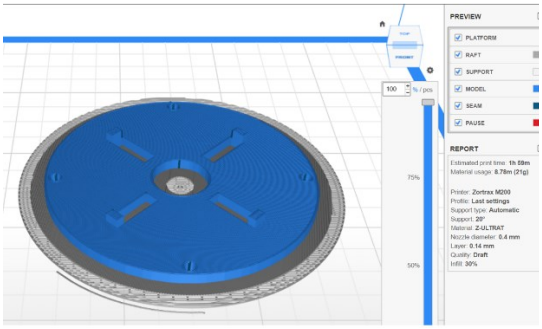


Figure 4.3 0.14mm 40% draft

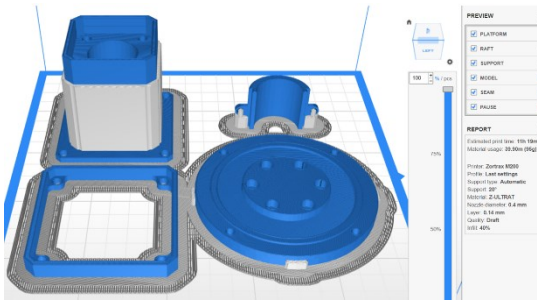


Figure 4.4 0.14mm 40% draft

The most challenging part to be correctly printed was the telescopic shaft. Comparing the shaft with all the other components, its study was carried out differently. They were designed and then adapted to the manufacturing method modifying, for example, the tolerances. The telescopic shaft was developed with the idea to be 3D printed as one part even if it is composed by three different sections. In Figure 4.5 it is possible to see how it was avoided that the section slipped out of their housings. Those teeth allow the axial elongation and the torsional rigidity.

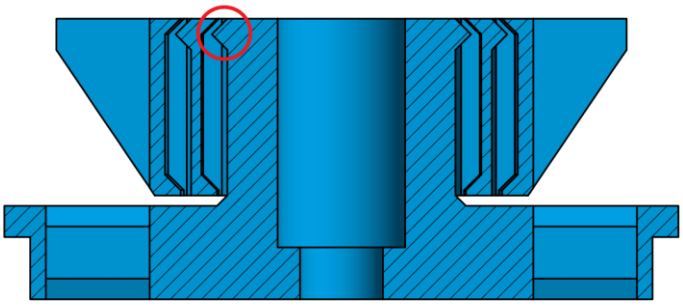


Figure 4.5 Telescopic shaft section

In order to have as less play as possible, the internal walls of this component were moved close to each other until there was only 0,1 mm between them. This value was not random, but it was the minimum value which the printer accepted before joins the walls together. It was the results of some tries because 3D printing parameters are variable in relation to the material, the temperature, the nozzle diameter, the layer height and the printing speed.
The printing was carried out with 0,9 mm of layer high, 60% of infill and a high-quality setting which means low speed and high wall thickness. Figure 4.6 shows the printing plate.

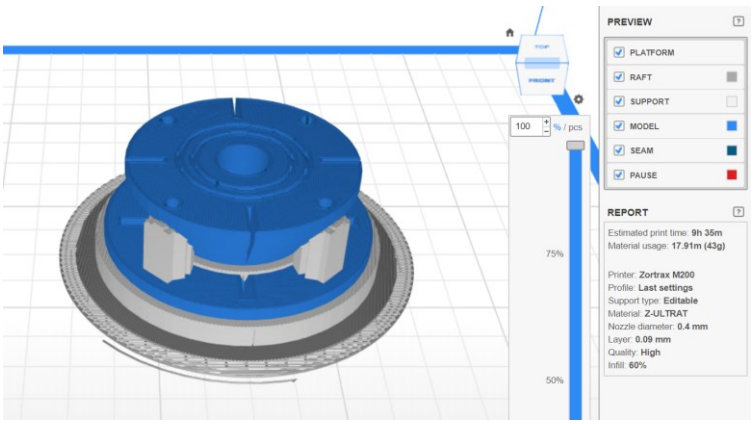


Figure 4.6 0.09mm 60% high

Once the components had been printed, the device was not assembled entirely but the components were divided in the groups shown in Paragraph 2.3 to be separately tested. First, the motor section was connected with the lower base in order to test the functionality of the Oldham joint (Figure 4.7).

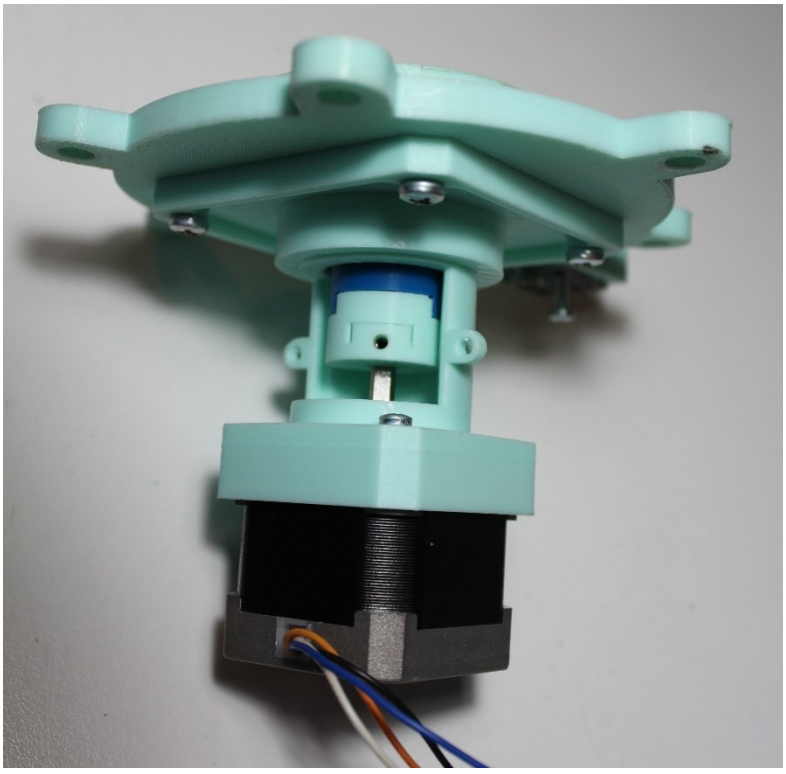


Figure 4.7 Motor group

The assembly continued with the application of the cam (Figure 4.8), in order to check if its rotation was precluded by interferences between its profile and the base pins, and the remaining components of the actuator.

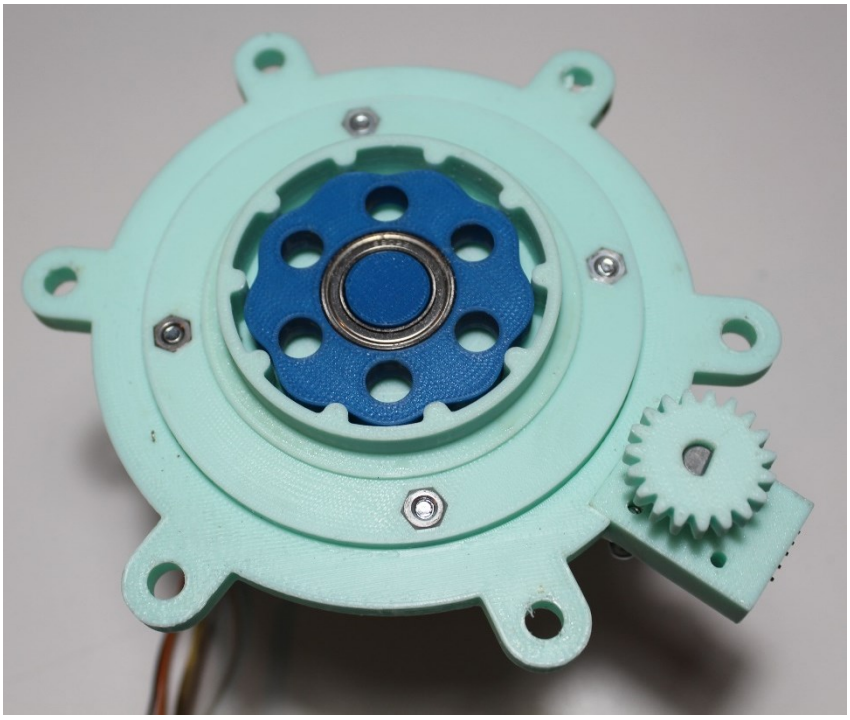


Figure 4.8 Actuator Cam

In Figure 4.9 it is possible to see the rotating base and the six steel rods which transfer the rotation from the cam to the spring system.

The whole actuator is shown in Figure 4.10.

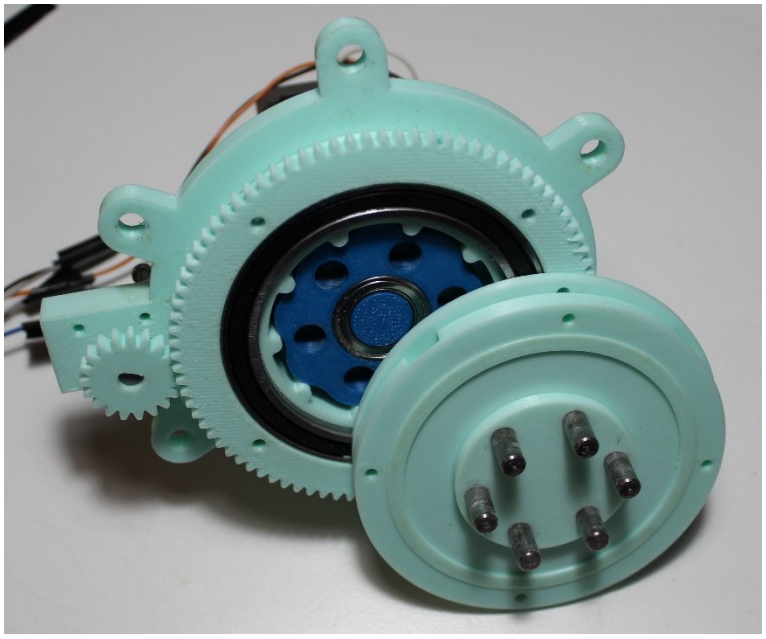


Figure 4.9 Actuator parts

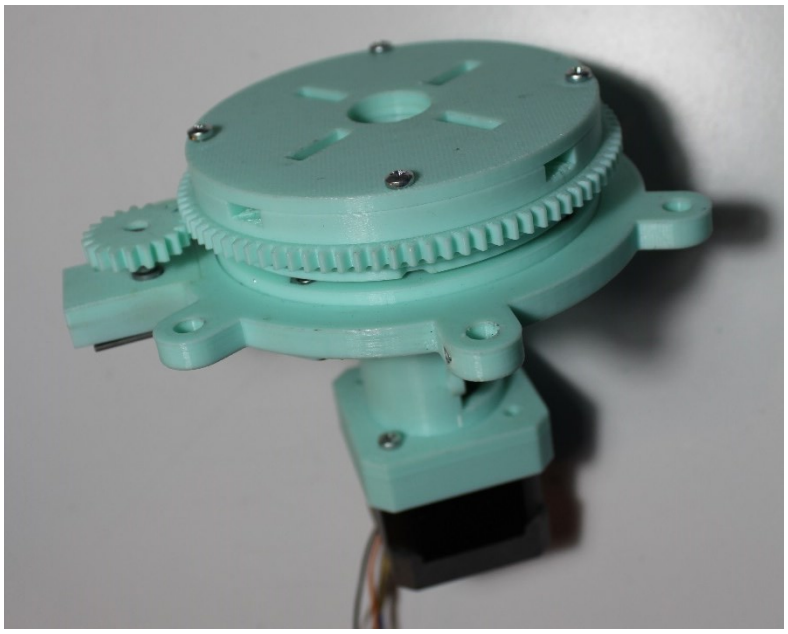


Figure 4.10 Actuator

The first components of the spring system to be assembled were the telescopic shaft and the parts rigidly connected to it. This group represents the out shaft of the device and it is shown in the following figures in its reduced and extended positions.

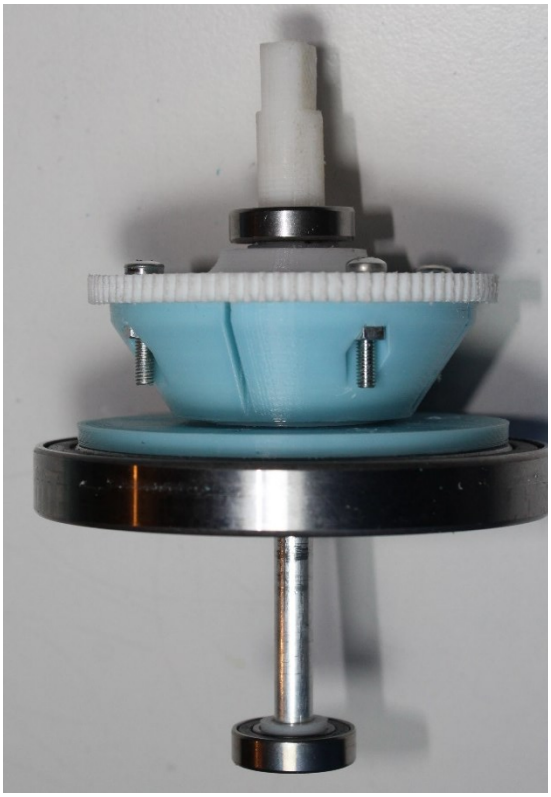


Figure 4.11 Output shaft reduced position

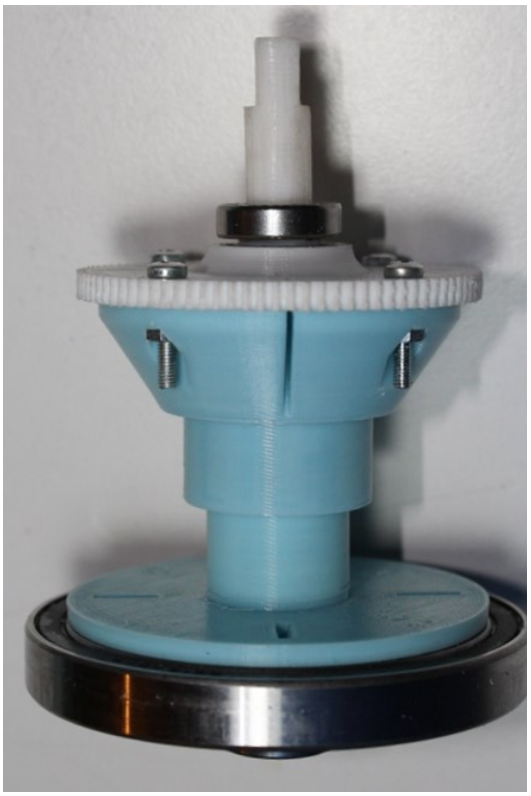


Figure 4.12 Output shaft extended position

Figure 4.13 shows the spring group assembly:

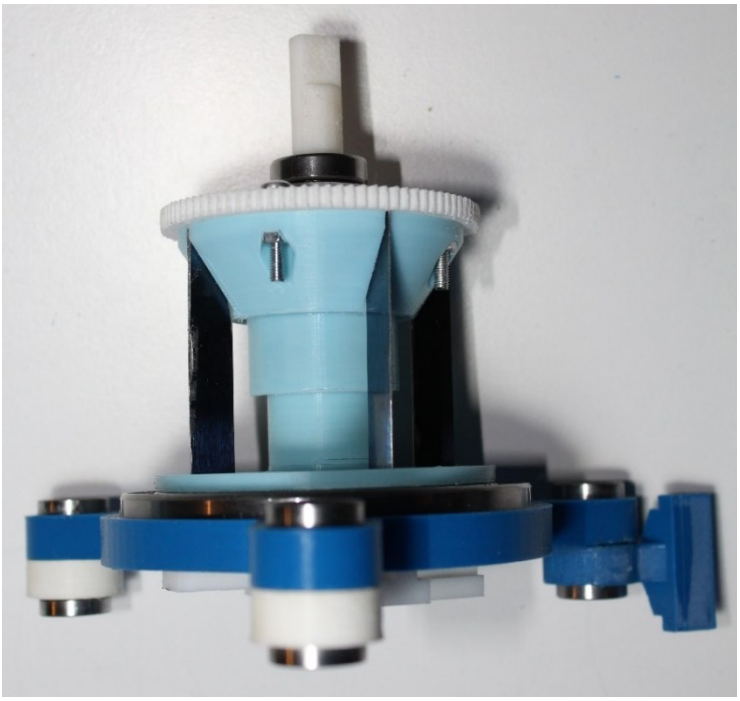


Figure 4.13 Spring Group

It was connected to the actuator through the leaf spring housings as shown in Figure 4.14, while Figure 4.15 shows the belt group.

Figure 4.14 Actuator and spring group connected

Figure 4.15 Belt Group

The final assembly resulted in the device shown in the following two figures:

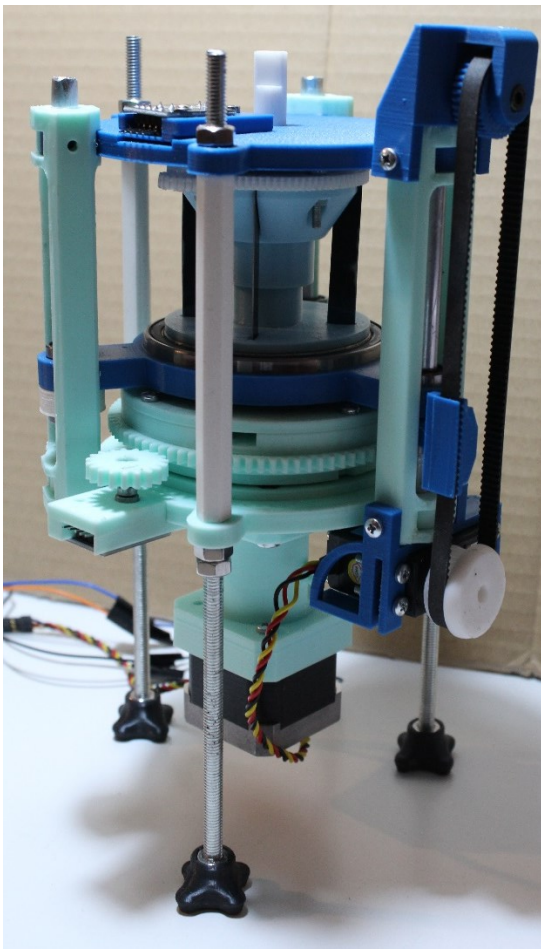


Figure 4.16 VSA device

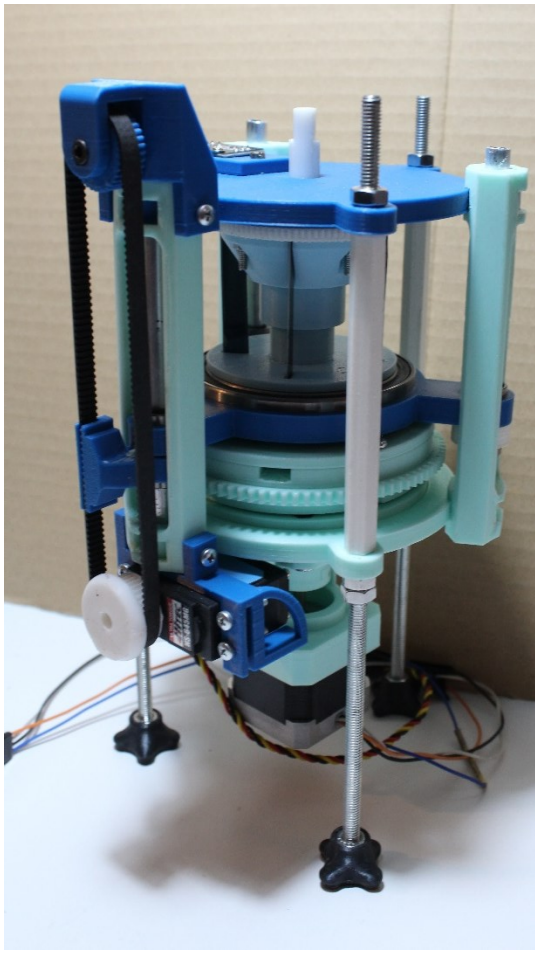


Figure 4.17 VSA device

4.2 Characterization

In order to compare the prototype behaviour and the initial mathematical analysis, a first characterization of the device was carried out.

The test setup was very simple because it was performed applying a torque to the output with the use of a dynamometric wrench and acquiring the results from the control output.

In order to read the absolute rotation of the output shaft, the motor was removed, and the input shaft rotation was locked.

The rotation resolution was determined by the hardware setup and thus the minimum rotation angle the encoders could measure was 2,48°.

The applied torque value went from 450 Nmm to 2486 Nmm and the measured rotations are shown in Figure 4.18.

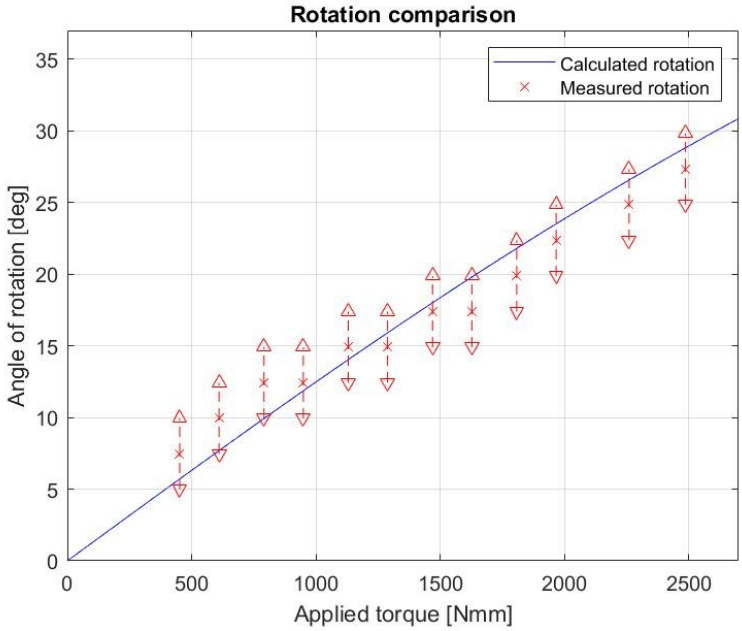


Figure 4.18 Rotation Comparison

Blu line represents the rotation calculated with the mathematical analysis proposed in paragraph 2.3.3 and red "x" points are the results of the measurement.

The values of "±Δ" points are calculated in the following way:

$$angle_{\pm\Delta} = angle_x \pm resolution$$

Due to the resolution, the actual rotation can be between "x" and "±Δ" values.

It is possible to see that measured values approximate calculated rotations, validating the initial mathematical analysis. Even if the shown results are acceptable, the performed test represents an estimation and a more rigorous characterization must be performed to confirm what just presented.

4.3 Control testing

The last proposed test was peformed to validate the control strategies discussed in paragraph 3.3.3. The aim of the test was to show how the difference between encoders measures controls the motor rotation.

As already written, the device control was developed in order to react to an external interaction. The most common circumstance a cobot may face is presence of an obstacle on its trajectory. The programmed reaction was the immediate stop of the rotation as soon as the difference in rotation exceed a imposed value.

Figure 4.19 shows the encoders signals while the device is rotating. The green line is the signal of the bottom encoder, which is the one rigidly connected to the actuator, while the black line is the signal of the upper encoder, which is rigidly connected to the output shaft.

The red dashed line shows the maximum acceptable difference from the bottom encoder values. It is possible to see that between 11920 ms and 12151 ms the upper encoder signal exceeds the threshold.

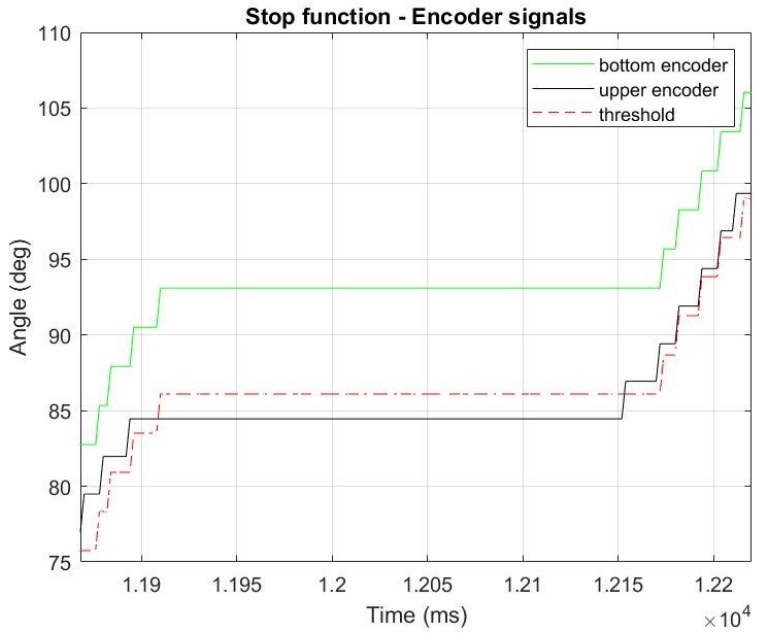


Figure 4.19 Stop function - Encoders signals

The same time range can be seen in Figure 4.20 in which the difference between the values of the two encoders (red) is compared with the value of the motor input variable (blue). Before 11910 ms and after 12154 ms the value of the motor input variable is 5 and the red line value is smaller than threshold (7°). Between these two time instants, the motor input goes to 0 and the motor rotation stops because the difference exceed 7°.

Black and green vertical axes showns the time lag between the exceeding of the threshold and the motor input reaction. It is possible to notice that the time lag is always equal to 10 ms which is the frequency of the control routine.

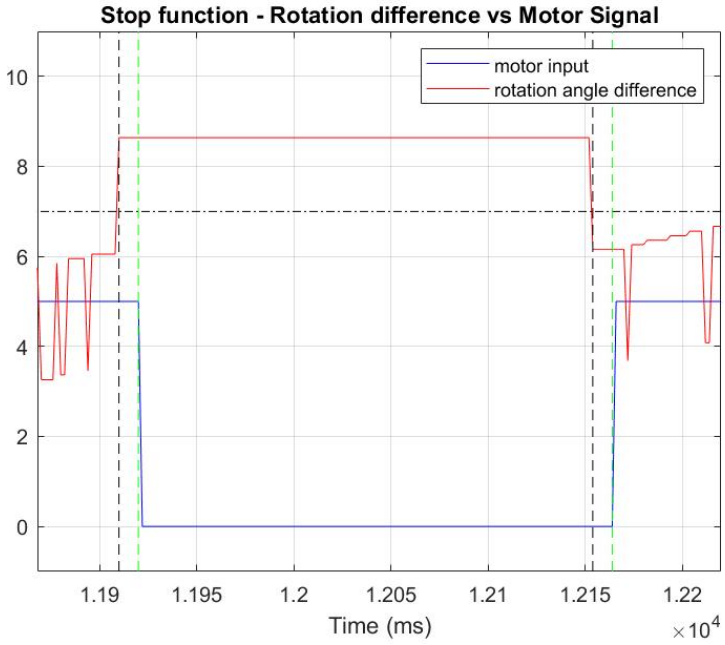


Figure 4.20 Stop function - Rotation difference vs Motor signal

The second possible circumstance a cobot has to deal with is a forced movement. The control was development to enable the motor rotation if the angle difference exceeds 135% of the maximum acceptable difference.

Figure 4.21 shows the encoders signals and the threshold to activate the “following” reaction. As before, green line is the bottom encoder and black line is the upper one.

The upper encoder signal exceeds the threshold between 3866 ms and 4082 ms.

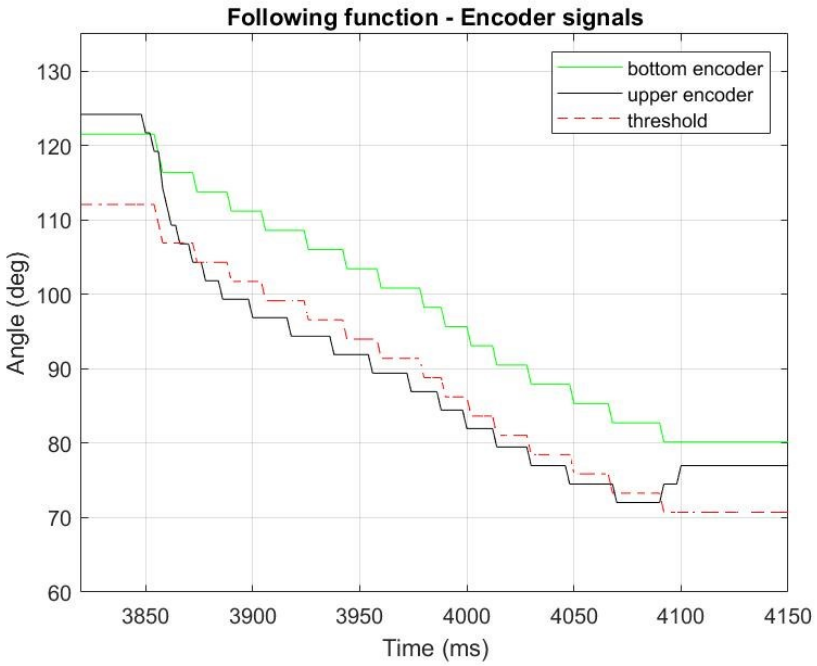


Figure 4.21 Following function - Encoders signals

The reaction of the motor is shown in Figure 4.22. Motor input variable is 0 before 3866 ms and after and 4082 ms because the rotation angles difference is less than 9,45° (threshold value) and becomes equal to -3 in between the two time instants. Also, in this figure the time lag can be seen, and it is always equal to 10 ms.

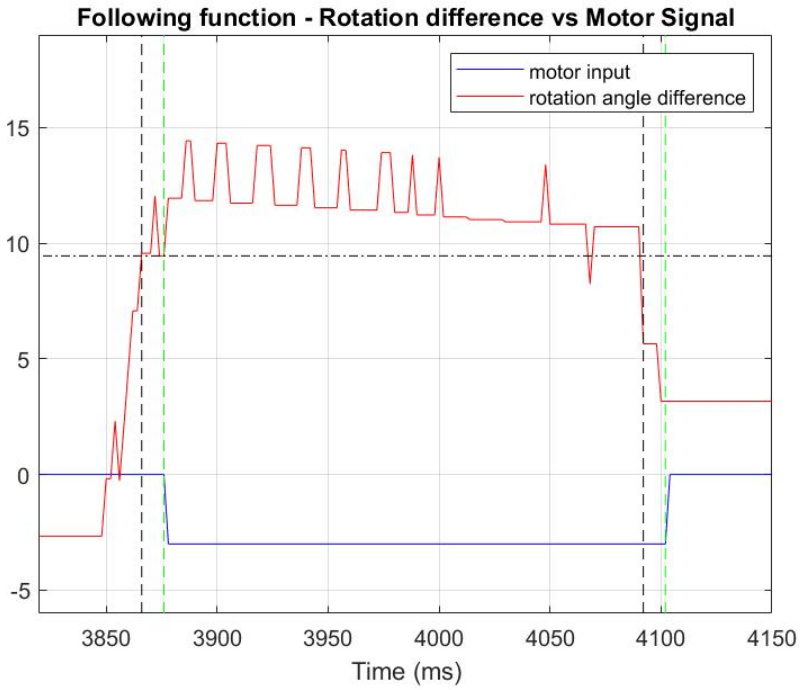


Figure 4.22 Following function - Rotation difference vs Motor signal

Conclusions

The aim of project was the mechanical and software development of a VSA device to be safely used by a collaborative robot working in contact with humans. Furthermore, the device should had been easy to use and low cost.

On the mechanical point of view, it had been shown a functional prototype which behaves as estimated by the theoretical analysis despite the issues generated by the friction.

Regarding the control, the software was written to be easily understood and customized by every user, but it ensures that, if an external interaction happens, the device reacts to avoid risks for human safety. It can be optimized but it already has all the needed functionality.

The routine to control the reactions of the device manages a blocking obstacle and a forcing interaction which summarize the most part of the possible situations a cobot deals with.

Summarizing, it can be stated that the designed prototype is a good basis for more detailed investigations and developments regarding continuous rotation VSA.

Improvements and future works

The possible mechanical improvements to be realize can be grouped in two different areas: the first one regards the mechanisms and the second the material.

Being the VSA developed to be apply to a cobot, the first improvement to be considered should be the resizing of the whole device. While studying to reduce the dimension, the stepper motor could be rotated of 90° but a new transmission shaft will be needed.

The mechanism to control the position of the slider could be rethought to be smaller and more accurate. For the actual design, the belt group actuates only one of the three contact points of the slider which may generate an unwanted inclination of the working plane. One idea could be a system in which three ball screws are used instead of the three steel rods and are actuated by the planets of an epicycloid gear.

Different material and manufacturing method could widely improve the performance of the VSA because, even if the 3D printing is perfect to protype, the precision and the accuracy of steel machining cannot be achieved. Clearly the cost of the device will be higher, but the reduction of the friction ensures a huge increase of precision.

In order to improve the control, different and more accurate encoders can be applied directly on the shafts (motor and output) without any additional gear.

Regarding the software, the interface with the cobot controls must be developed and the control should be extended to manage five steppers simultaneously.

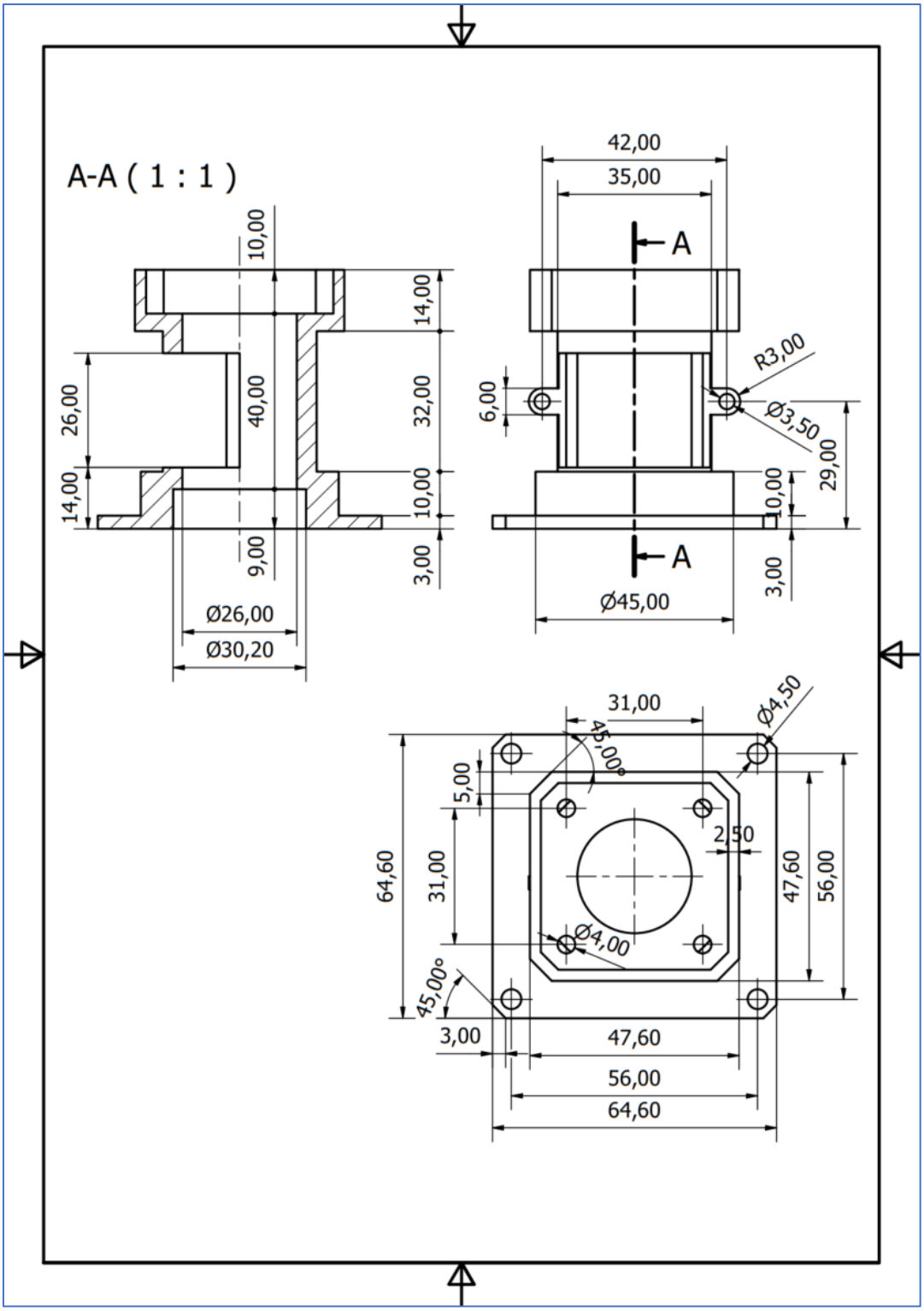
As last, a more precise characterization test should be performed after the re-size in order to acquire all the information to perfectly tune the control. The ideal test set-up would have included the use of a torque meter, a measured amplifier and an analog-to-digital converter. In addition to the already performed test which should repeat in a more accurate way, it could be useful to measure the no-load running torque minimum, which is the torque able to let the actuator rotate, and to confirm the gear ratios measuring the output shaft rotation with an external encoder.

Appendixes

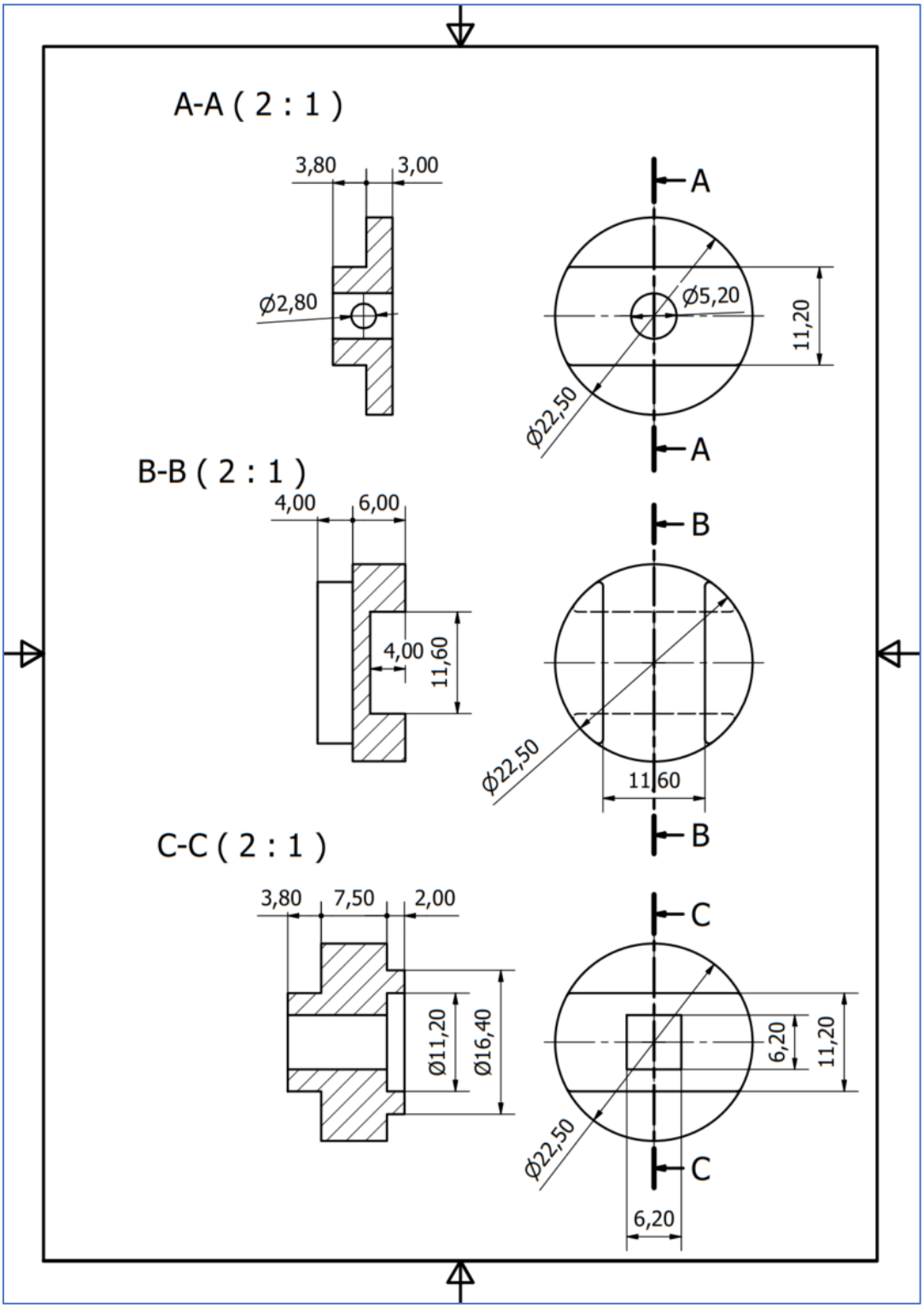
4.4 Appendix A

A.1 Technical drawings

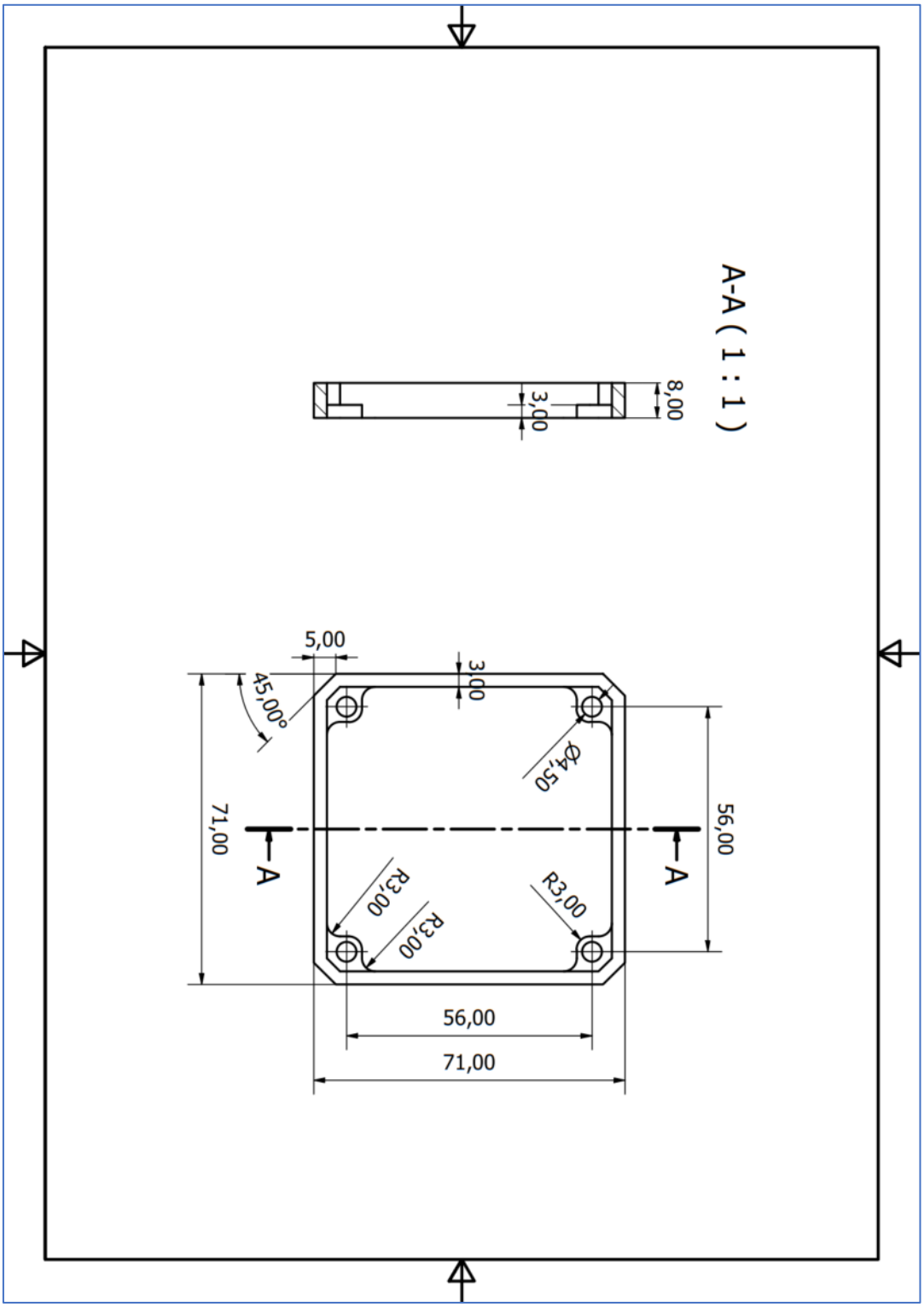
Technical drawing 1: Motor case



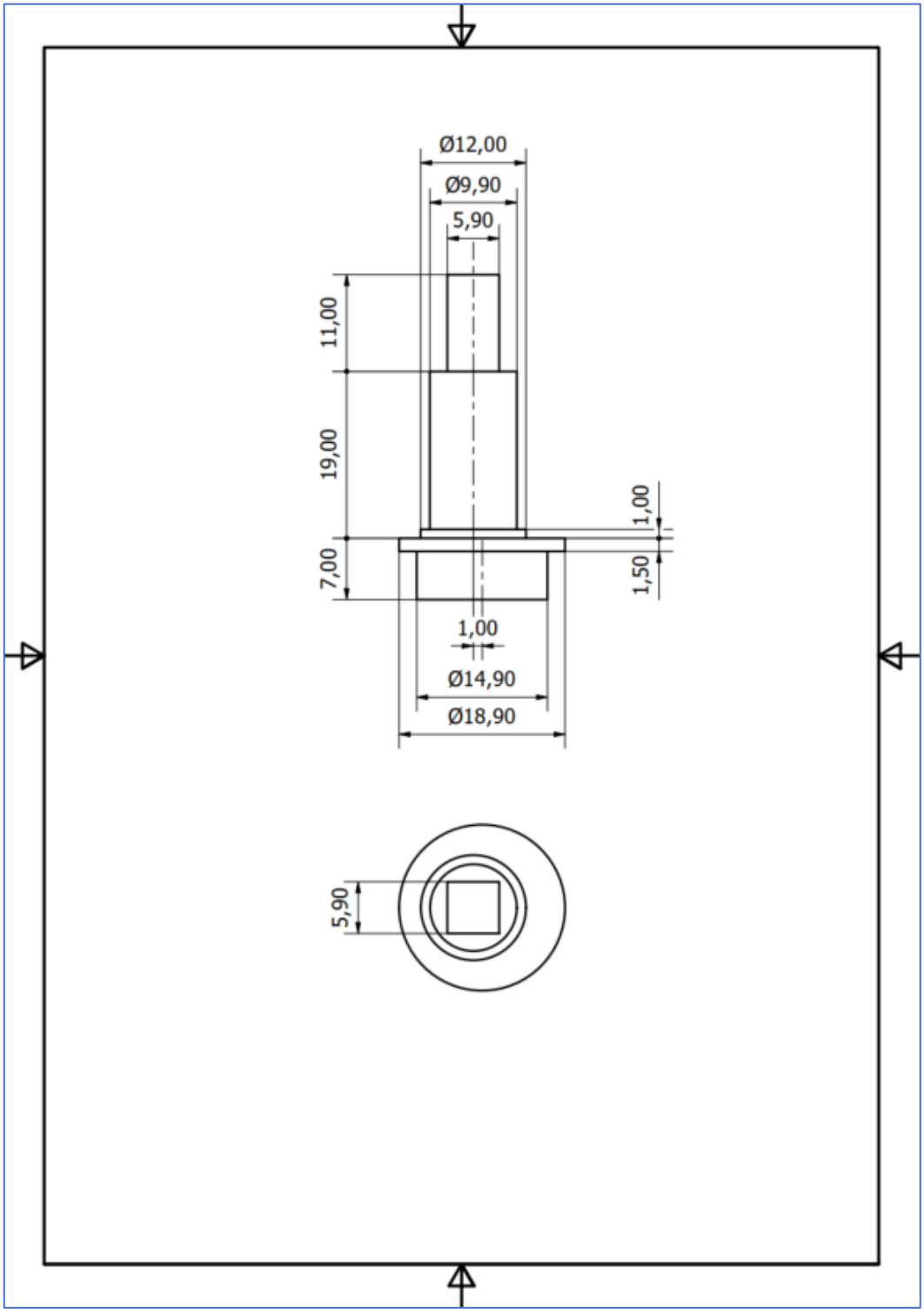
Technical drawing 2: Oldham joint three parts



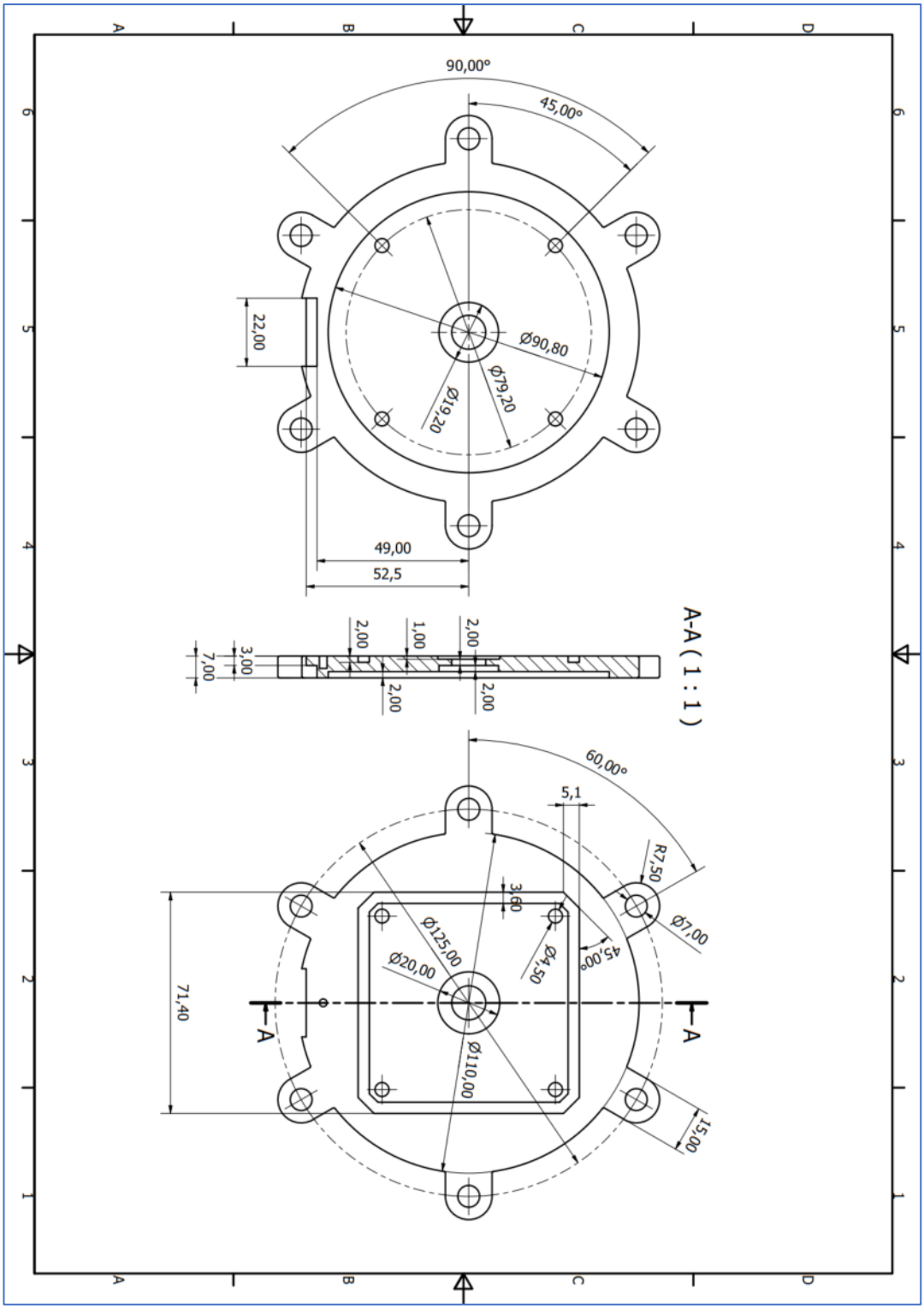
Technical drawing 3: Adapting case



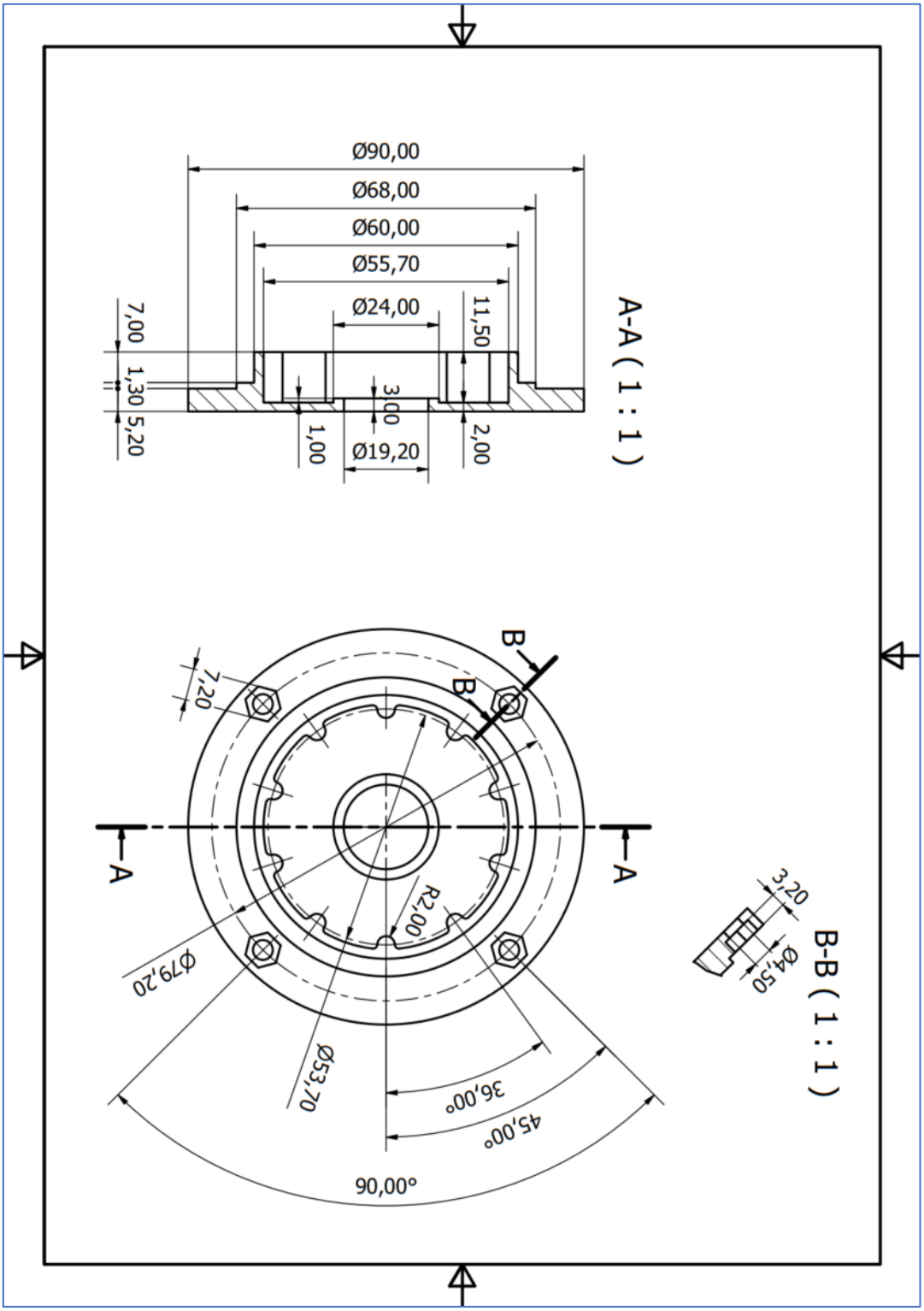
Technical drawing 4: Cam shaft



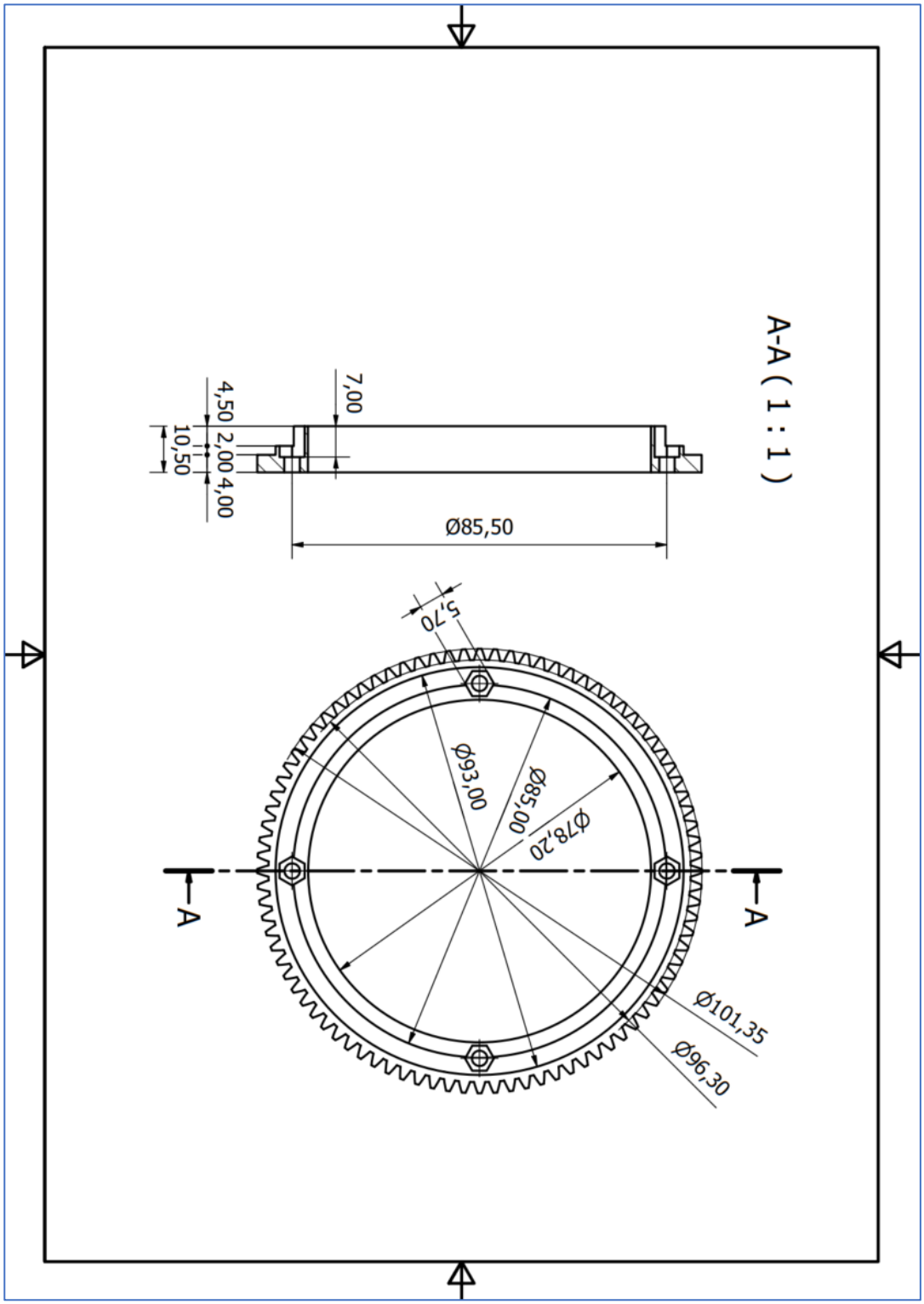
Technical drawing 5: Lower external case



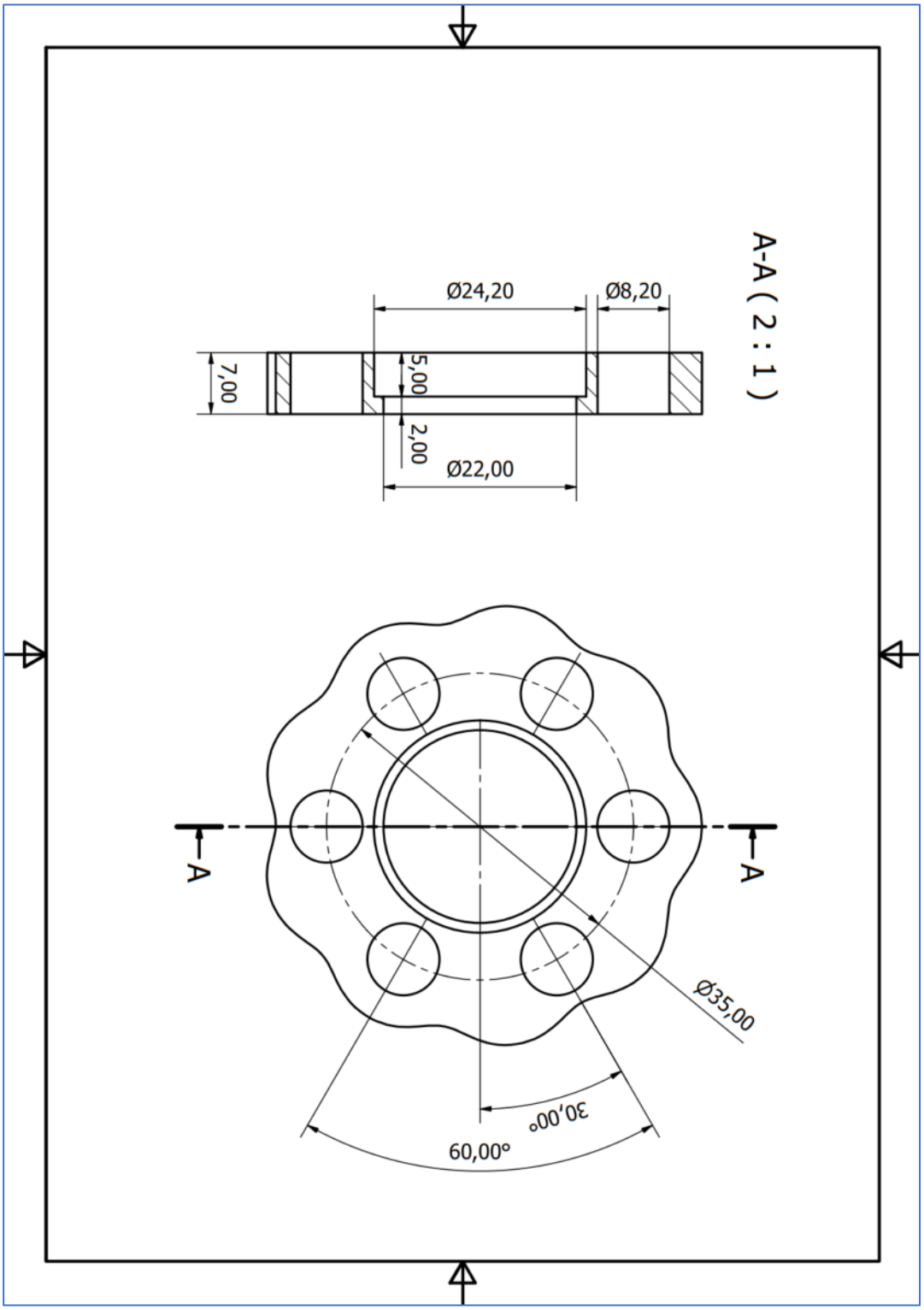
Technical drawing 6: Base



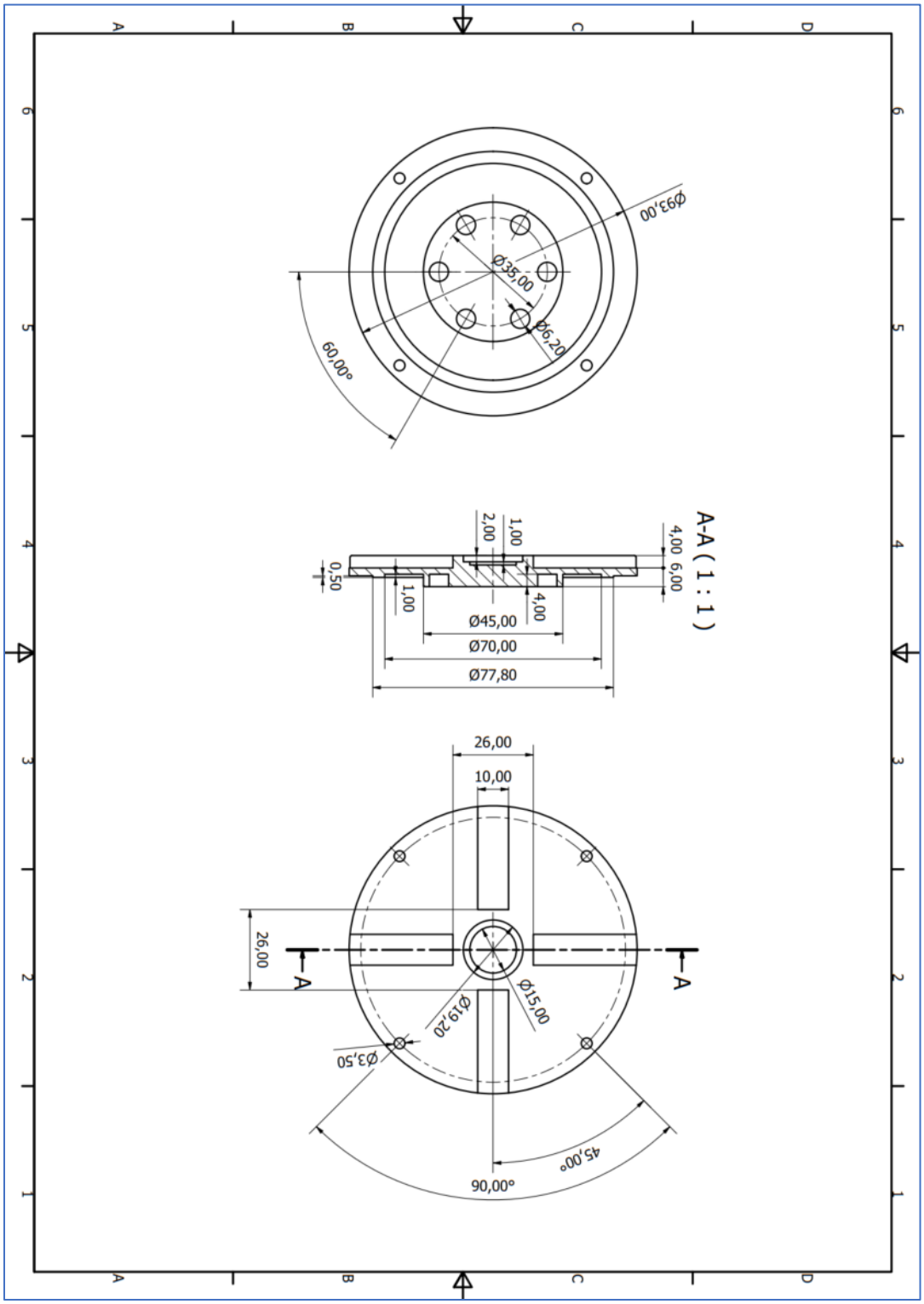
Technical drawing 7: Bottom encoder gear



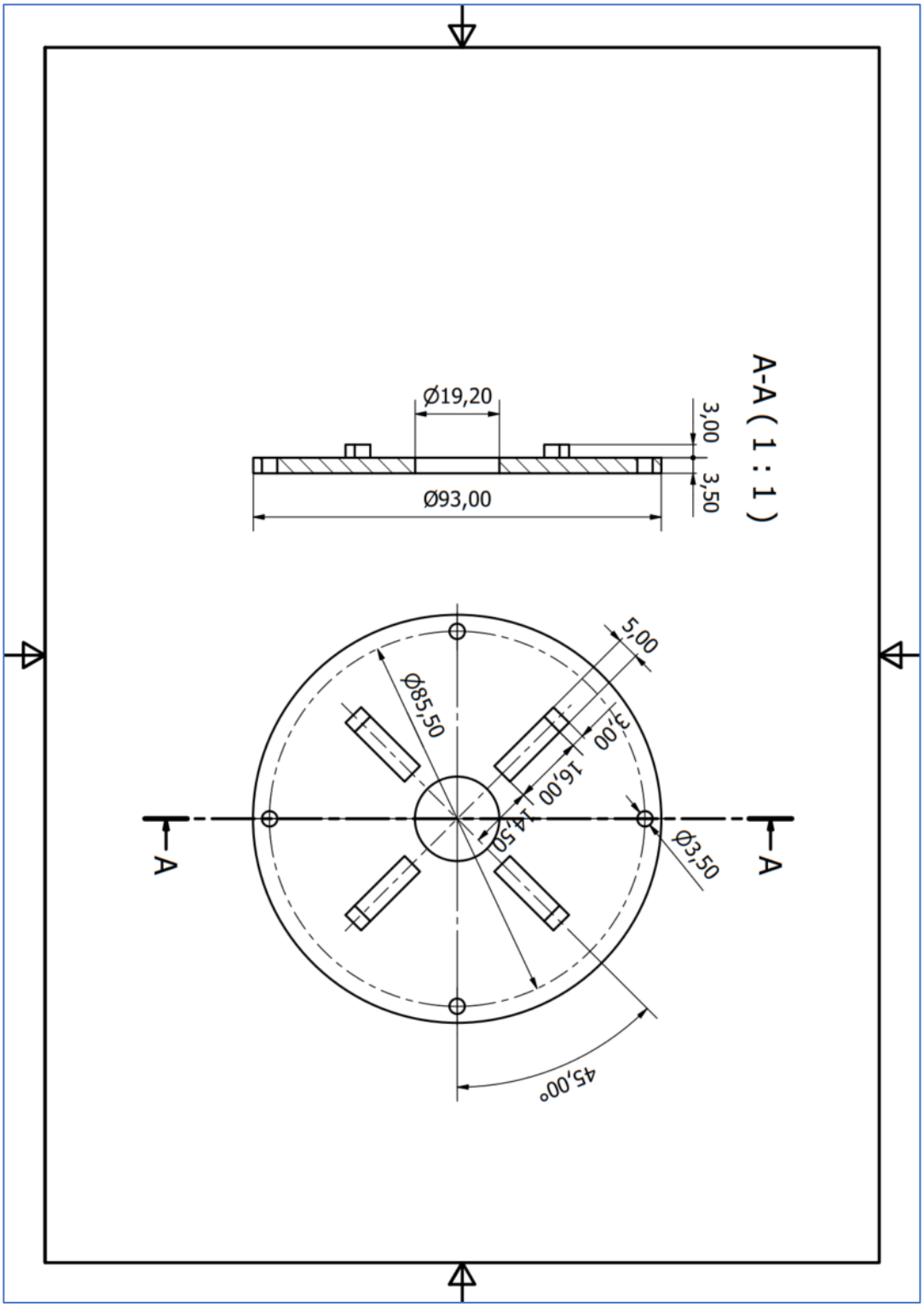
Technical drawing 8: Cam



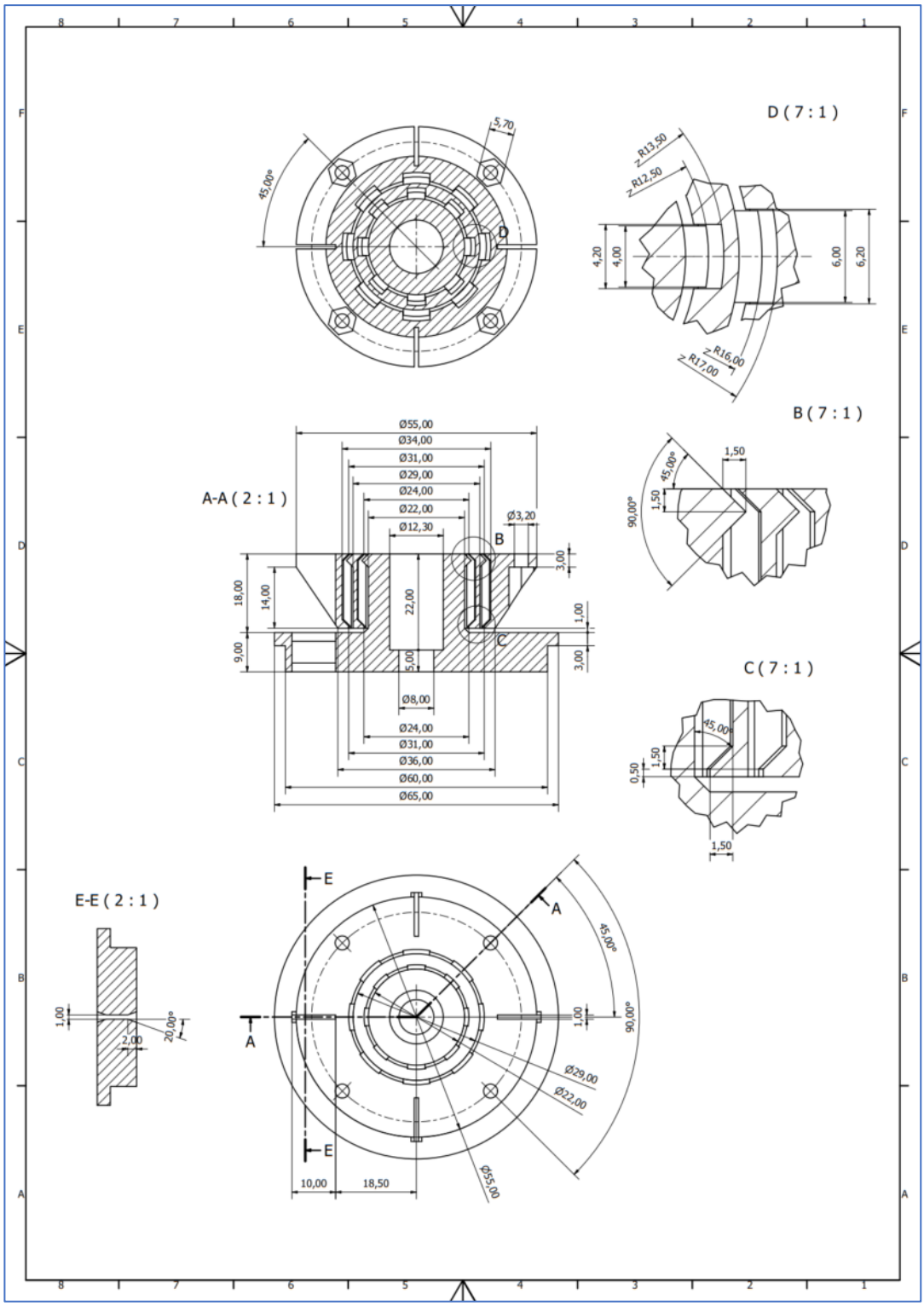
Technical drawing 9: Rotating base



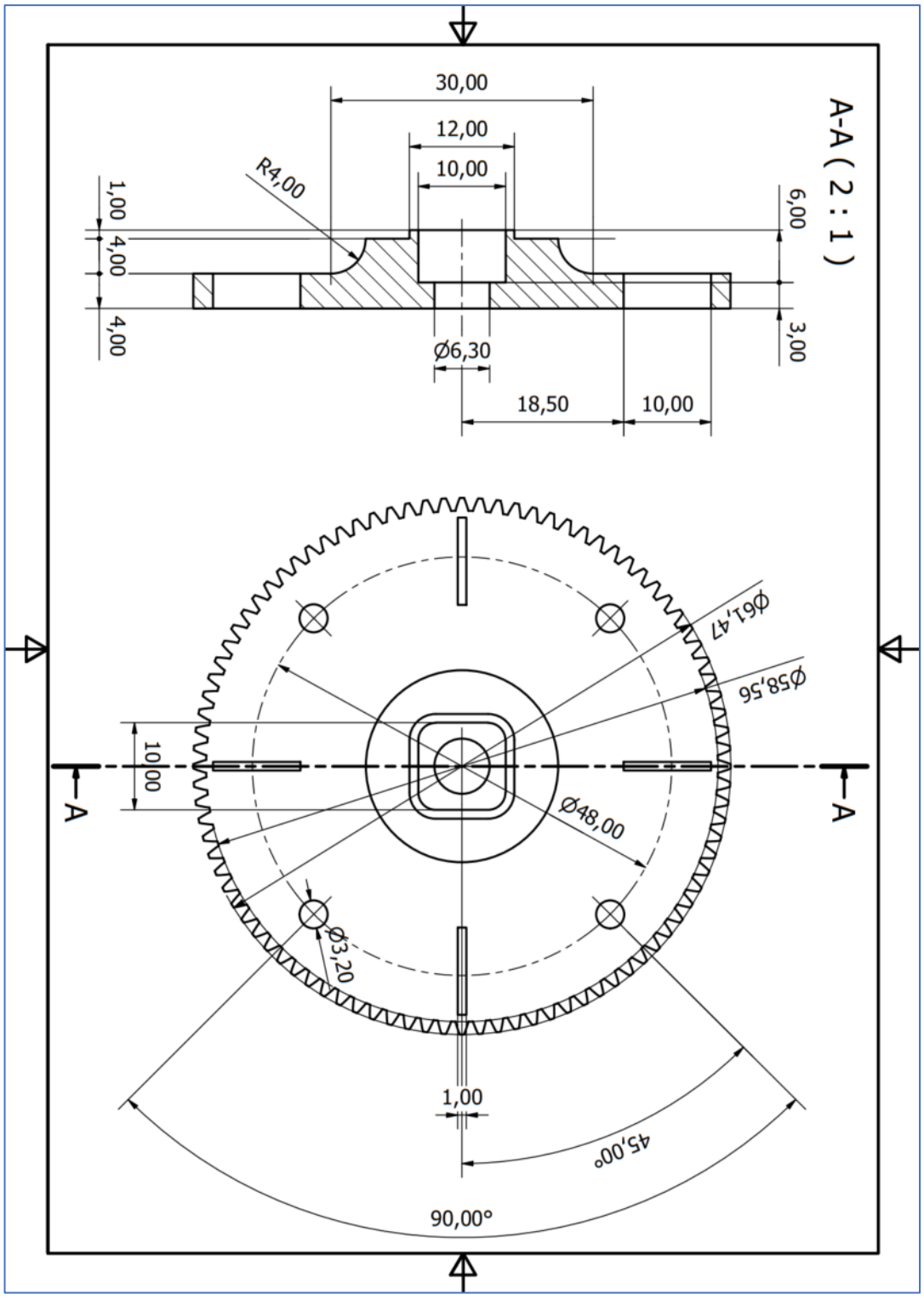
Technical drawing 10: Closing plate



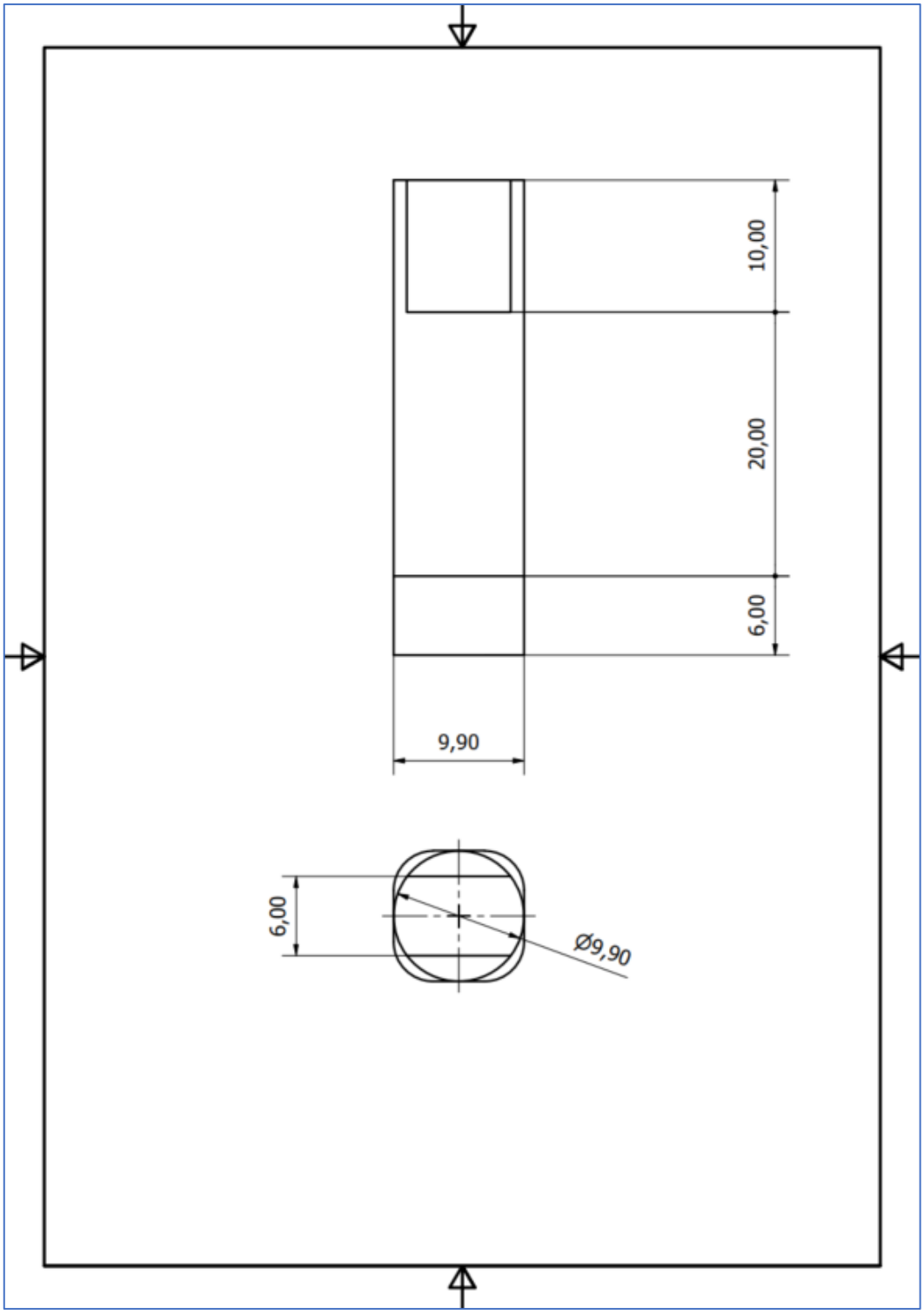
Technical drawing 11: Telescopic shaft



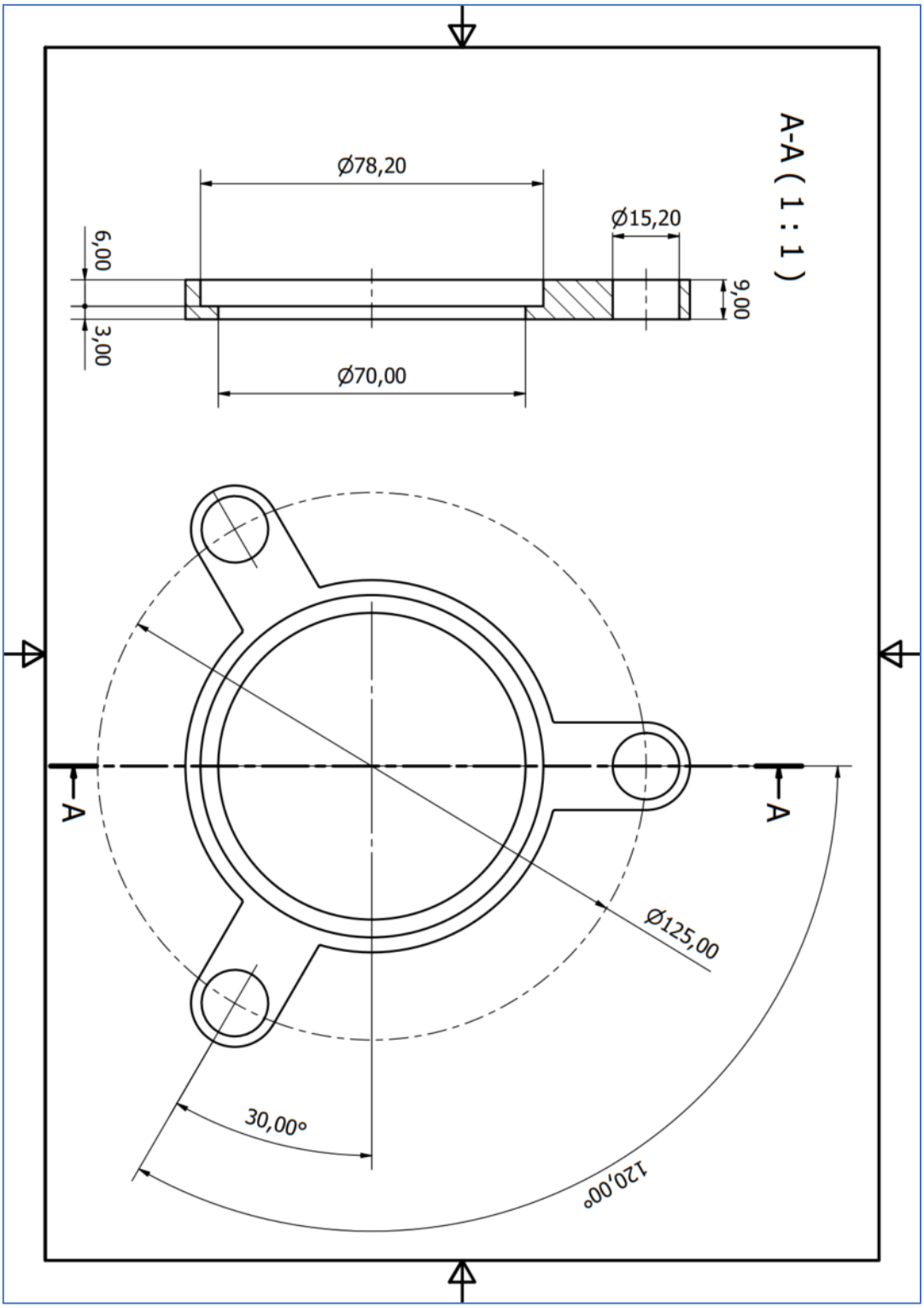
Technical drawing 12: Upper encoder gear



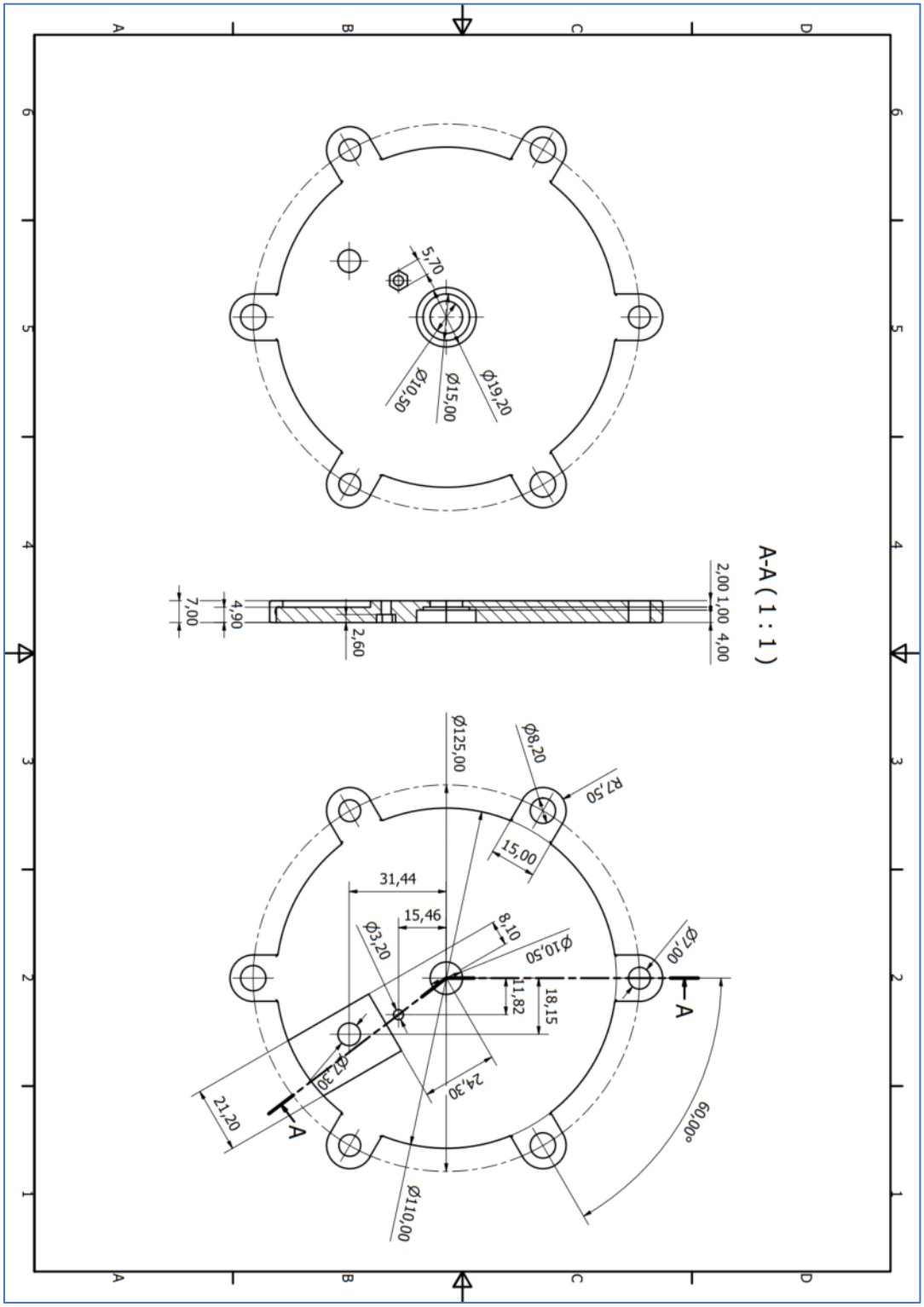
Technical drawing 13: Output shaft



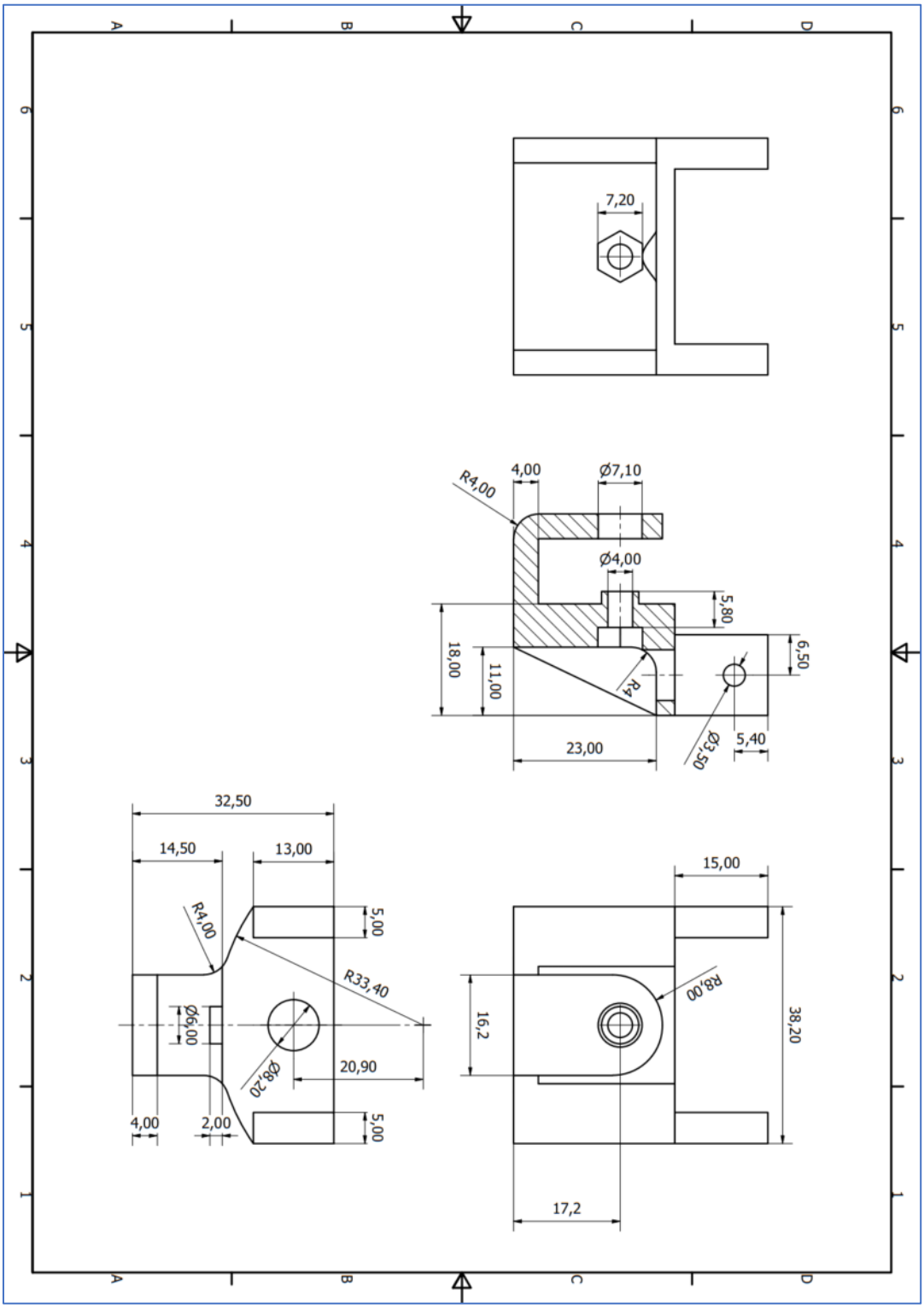
Technical drawing 14: Slider



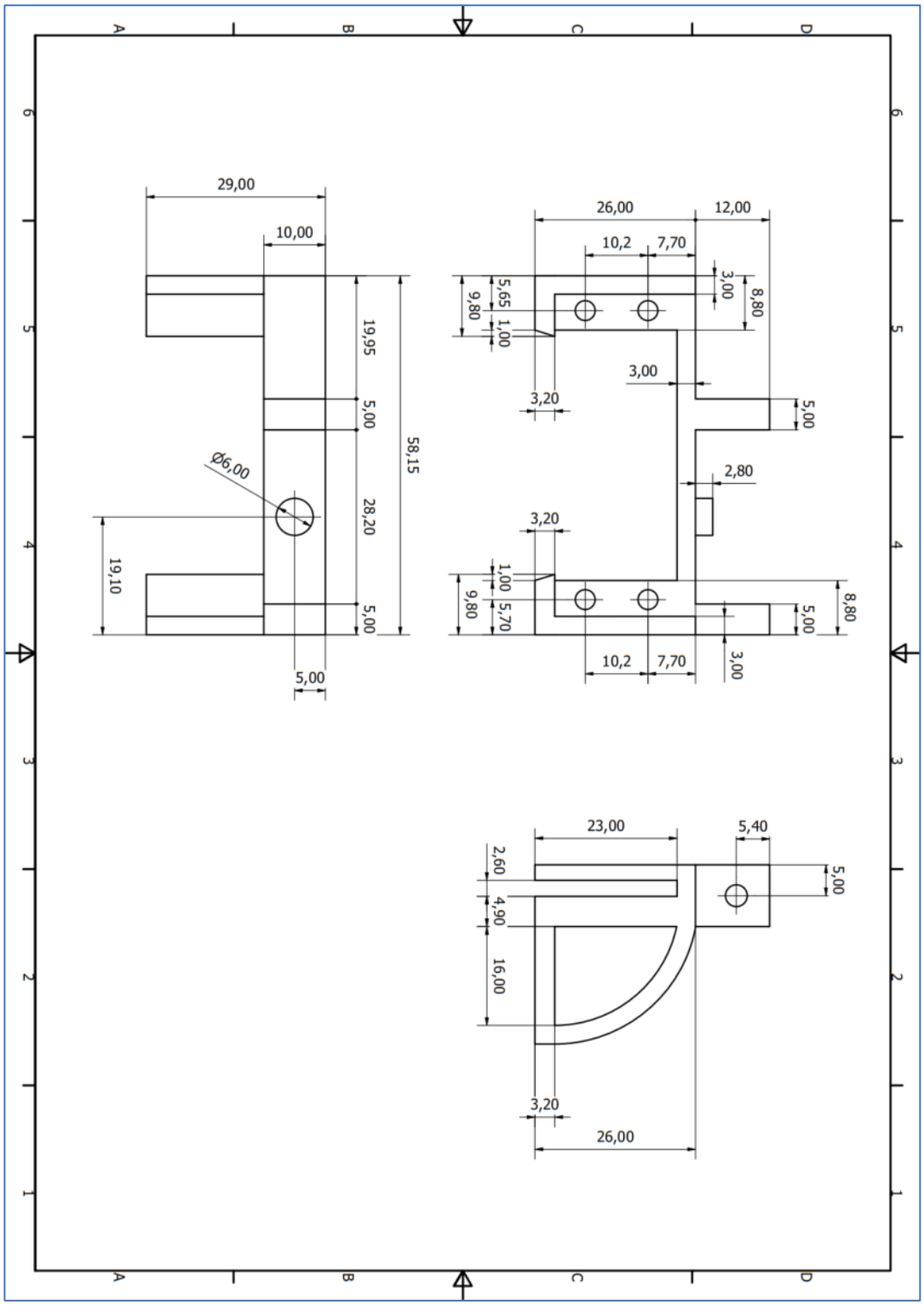
Technical drawing 15: Upper external case



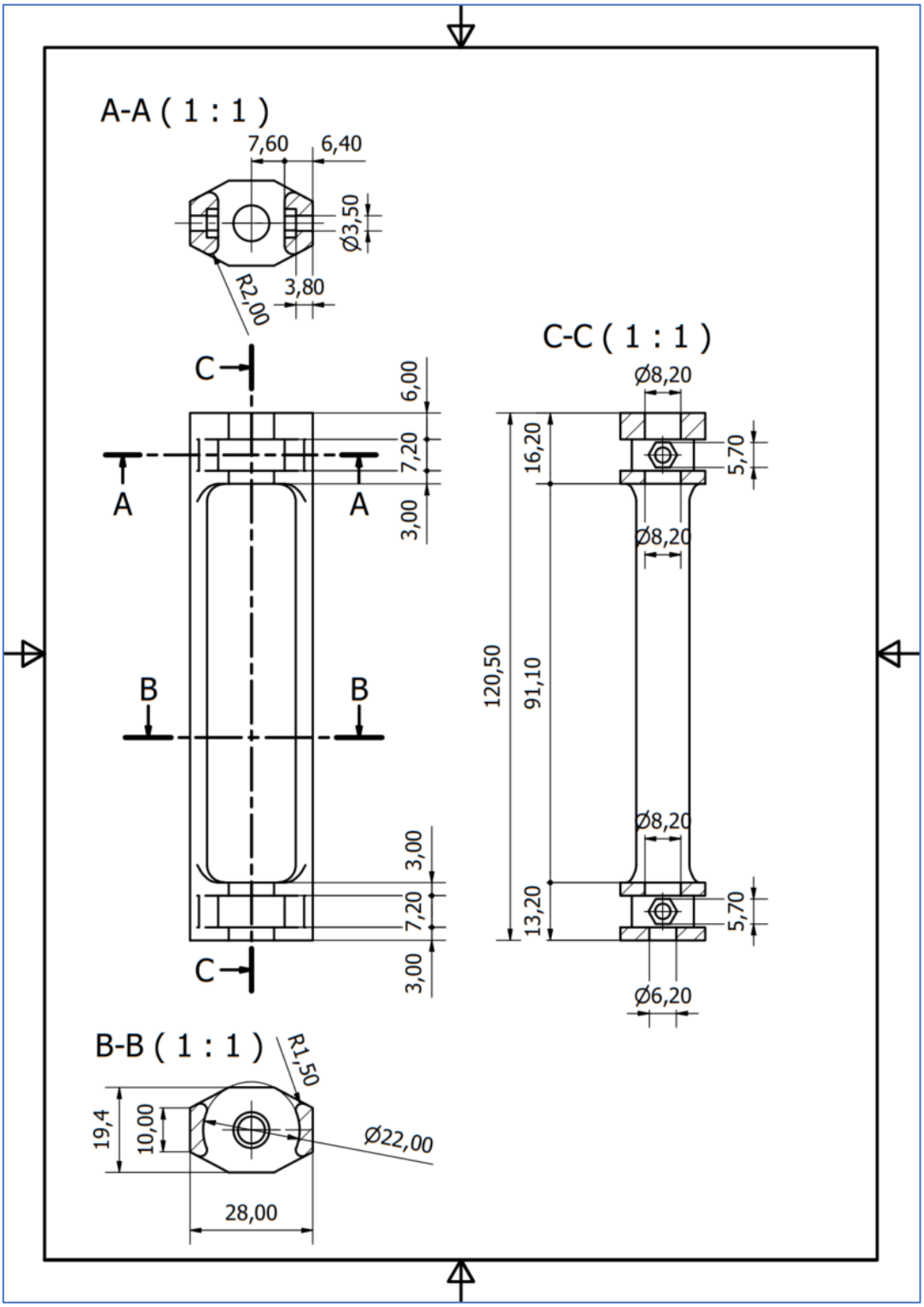
Technical drawing 16: Pulley holder



Technical drawing 17: Servo holder



Technical drawing 18: Case spacer

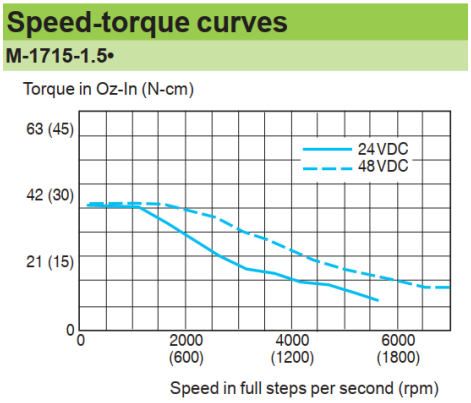


A.2 Datasheet

Datasheet 1 Stepper motor NEMA 17 Specifications

Electrical and mechanical data				
NEMA17		**M-1713-1.5•**	**M-1715-1.5•**	**M-1719-1.5•**
Stack length		single	double	triple
Phase current	**amps**	1.5	1.5	1.5
Holding torque	**oz-in**	32	60	75
	N-cm	23	42	53
Rotor inertia	**oz-in-sec²**	0.000538	0.0008037	0.0011562
	kg-cm²	0.038	0.057	0.082
Phase inductance	**mH**	2.1	5.0	3.85
Phase resistance	Ω	1.3	2.1	2.0
Weight	oz	7.4	8.1	12.7
	grams	210	230	360

Datasheet 2 NEMA17 Torque Plot



Datasheet 3 Servo motor Hitec HS-645MG Specifications

Performance Specifications	
Operating Voltage Range (Volts DC)	4.8V ~ 6.0V
Speed (Second @ 60°)	0.24 ~ 0.20
Maximum Torque Range oz. / in.	107 ~ 133
Maximum Torque Range kg. / cm.	7.7 ~ 9.6
Current Draw at Idle	9.1 mA
No Load Operating Current Draw	450 mA
Stall Current Draw	2,500 mA
Dead Band Width	8 µs

4.5 Appendix B

B.1 Arduino code

B.1.1 Arduino - Main

```
#include <Servo.h>
#include <ros.h>
#include <std_msgs/Empty.h>
#include <std_msgs/Int8.h>
#include <std_msgs/Int16.h>
#include "geometry_msgs/Twist.h"
#include "i2c_encoder_ros.h"
#include "stepper.h"

Servo myservo;
ros::NodeHandle  nh;
geometry_msgs::Twist msg4;
std_msgs::Int16 msg1;
std_msgs::Int16 msg2;
std_msgs::Int16 msg5; // lettura encoder2
ros::Publisher p1("en1", &msg1);
ros::Publisher p2("en2", &msg2);
ros::Publisher p3("pot", &msg5);
std_msgs::Int16 msg6;

int rpm = 0;
unsigned long pwm = 0;
int potpin = 3;
int pot_read = 0;
int servo = 32;
float dtZ = 0;
int stepZ = 0;
int RS = 0;
float zz = 1;
unsigned long currentTimeMot;
unsigned long loopTimeMot;
unsigned long delayTimeMot = 10;

void pwmFcn(const std_msgs::Int16 msg6) {
    pwm = msg6.data;
}
void resetFcn(const std_msgs::Int16 msg10) {
    RS = msg10.data;
    if (RS > 0) {
        jj = 0;
        kk = 0;
    }
}
```

```
void startMotor(const geometry_msgs::Twist msg4) {

    zz = msg4.linear.z;

    if (zz == 0) {
        digitalWrite(motZ[2], HIGH);
        dtZ = 0;
    } else {
        digitalWrite(motZ[2], LOW);
        dtZ = ((float)(delayTimeMot * 500) / (float)(zz));
    }

    sign_data[2] = 1;

    if (dtZ < 0) {
        sign_data[2] = 0;
    }

    if (abs(dtZ) > 30) {
        interrupts();
        count1_stepper[2] = abs(dtZ);
        count1_stepper[2] = round((float)(16 * count1_stepper[2] / 8
                                                              - 1));
        count_stepper[2] = count1_stepper[2];
        counter_stepper[2] = 0;
    }
    else {
        count1_stepper[2] = 0;
    }
}

ros::Subscriber<geometry_msgs::Twist> motor("step", &startMotor);
ros::Subscriber<std_msgs::Int16> reset("res", &resetFcn);
ros::Subscriber<std_msgs::Int16> PWM_val("pwm", &pwmFcn);
```

```
void setup() {
    nh.initNode();
    nh.advertise(p1);
    nh.advertise(p2);
    nh.advertise(p3);
    nh.subscribe(motor);
    nh.subscribe(reset);
    nh.subscribe(PWM_val);
    mcpX.begin(0);

    for (int n = 0; n < encCount0; n++) {
        encPinsSetup(mcpX, encPins0[n]);
        encoders0[n] = 0;
    }
    currentTime = millis();
    loopTime = currentTime;
    currentTimeMot = millis();
    loopTimeMot = currentTimeMot;
    pinMode(motZ[0], OUTPUT);
    pinMode(motZ[1], OUTPUT);
    pinMode(motZ[2], OUTPUT);
    digitalWrite(motZ[2], LOW);
    pinMode(servo, OUTPUT);
    interrupts_setup();
    myservo.attach(servo);
}

void loop() {
    currentTimeMot = millis();
    encoder();
    pwm = (unsigned long)pwm;
    myservo.write(pwm);
    if (currentTimeMot >= (loopTimeMot + delayTimeMot)) {
        msg1.data = jj;
        p1.publish(&msg1);
        msg2.data = kk;
        p2.publish(&msg2);
        msg5.data = (int)pot_read;
        p3.publish(&msg5);
        nh.spinOnce();
        loopTimeMot = currentTimeMot;
    }
}
```

B.1.2 Arduino - i2c_encoder_ros

```
#include "Adafruit_MCP23017.h"
#include <Arduino.h>
Adafruit_MCP23017 mcpX;
boolean change[2]= {false, false};
int jj=0; //enc1 counter
int kk=0; //enc0 counter
unsigned long currentTime;
unsigned long loopTime;
const int encCount0 = 2;
int encSelect[encCount0][2] = {
    {101, 0},
    {101, 0}
}
const int encPins0[encCount0][2] = {
    {0,8},   // enc:0 AA GPA0,GPB0 - pins 21/22 on MCP23017
    {1,9}    // enc:1 BB GPA1,GPB1 - pins 24/25 on MCP23017
};
unsigned char encoders0[encCount0];
unsigned char readEnc(Adafruit_MCP23017 mcpX, const int *pin,
                                    unsigned char prev, int encNo) {
    unsigned char encA = mcpX.digitalRead(pin[0]);
    unsigned char encB = mcpX.digitalRead(pin[1]);
    if((encA!=prev)) {
        encSelect[encNo][0] = encNo;
        if(encB!=encA) {
            encSelect[encNo][1] = 1
        } else {
            encSelect[encNo][1] = 2;
        }
        change[encNo]=true;
    }
    return encA;
}
unsigned char encPinsSetup(Adafruit_MCP23017 mcpX, const int *pin) {
    mcpX.pinMode(pin[0], INPUT);
    mcpX.pullUp(pin[0], HIGH);
    mcpX.pinMode(pin[1], INPUT);
    mcpX.pullUp(pin[1], HIGH);
}
```

```
void encoder() {
    currentTime = millis();
    if (currentTime >= (loopTime + 2)){
        for (int n = 0; n < encCount0; n++) {
            encoders0[n] = readEnc(mcpX, encPins0[n],
            encoders0[n],n);
        }
        if (change[0] == true) {
            if (encSelect[0][0] < 100) {
                switch (encSelect[0][1]) {
                    case (1): // counter-clockwise
                        jj++;
                        break;
                    case (2): // clockwise
                        jj--;
                        break;
                }
            }
            encSelect[0][0] = 101;
            change[0] = false;
        }
        if (change[1] == true) {
            if (encSelect[1][0] < 100) {
                switch (encSelect[1][1]) {
                    case (1): // counter-clockwise
                        kk++;
                        break;
                    case (2): // clockwise
                        kk--;
                        break;
                }
            }
            encSelect[1][0] = 101;
            change[1] = false;
        }
    loopTime = currentTime;  // Updates loopTime
    }
}
```

B.1.3 Arduino - stepper

```
int motZ[3] = {46, 48, 62}; // Z motor {step pin, dir pin, enable pin}
float count1_stepper[4] = {0,0,0,0};
float count_stepper[4] = {0,0,0,0};
int counter_stepper[4] = {0,0,0,0};
float max_count = 0;
unsigned long previousMillis = 0;
int sign_data[4] = {0,0,0,0};

void interrupts_setup(){
    noInterrupts();           // disable all interrupts
    TCCR3A = 0;
    TCCR3B = 0;
    OCR3B = 30;
    TCCR3B |= (1 << WGM32);   // CTC mode
    TCCR3B |= (1 << CS31);    // 8 prescaler  //(1 << CS12);
    TIMSK3 |= (1 << OCIE3B);  // enable timer compare interrupt
    interrupts();             // enable all int errupts
}
//////////////////////////////////////////////////////////////////////////
void step_motor() {
    noInterrupts();
    max_count = count_stepper[2];
    OCR3B = 1 * max_count;
    if (OCR3B>65536){OCR3B=65535;}
    interrupts();
    max_count = 0;
}
//////////////////////////////////////////////////////////////////////////
ISR(TIMER3_COMPB_vect) {
    TCNT3 = 0;
    if (count1_stepper[2]!=0){
        step_motor();
        if (sign_data[2] > 0) digitalWrite(motZ[1], LOW);
        if (sign_data[2] <= 0) digitalWrite(motZ[1], HIGH);
        digitalWrite(motZ[0], digitalRead(motZ[0]) ^ 1);
        counter_stepper[2] = counter_stepper[2]+1;
    }
}
```

B.2 C++ code

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <sensor_msgs/JointState.h>
#include <std_msgs/Int8.h>
#include <std_msgs/Int16.h>
#include <std_msgs/Float32.h>
#include <std_msgs/String.h>
#include <rosserial_arduino/Adc.h>
int joystick = 0;
int jR = 0;
int i = 0;
int ii = 0;
int aa = 1;
int RS = 0;
float Dt = 0.01;
int en1_val = 0;
int en2_val = 0;
int pwm = 0;
int pot_read = 0;
int max = 360;
double en1_angle = 0;
double en2_angle = 0;
double en_diff = 0;
int delta_angle = 0;
int s = 0.6; //mm
int b = 10; //mm
int H_max = 67;
int C = 0;
float Ra = 23.5;
float x = 0;
int E = 206000;
int J = 1/12*b*s^3;
double en_to_degree = 360/30;
double en1_to_shaft = 0.2155;
double en2_to_shaft = 0.207;//0.2069;
void messageCallback1(const std_msgs::Int16::ConstPtr & msg1) {
    en1_val = msg1->data; //UP
}
```

```
void messageCallback2(const std_msgs::Int16::ConstPtr & msg2) {
    en2_val = msg2->data; //DOWN
    en2_val = en2_val;
}

void messageCallback3(const std_msgs::Int16::ConstPtr& msg3) {
    joystick = msg3->data;
}

void messageCallback5(const std_msgs::Int16::ConstPtr& msg5) {
    pot_read = msg5->data;
}

void messageCallback7(const std_msgs::Int16::ConstPtr& msg7) {
    max = msg7->data;
}

void messageCallback10(const std_msgs::Int16::ConstPtr& msg10) {
    RS = msg10->data;
    if (RS > 0){
        en1_angle = 0;
        en2_angle = 0;
    }
    ROS_INFO("I heard: [%d]",RS);
}

int inseguimento(double en_diff) {
    int data = -3*en_diff/abs(en_diff);
    return data;
}

int flessione(int pot_read, double en_diff){
    pot_angle = 20 + pot_read * 125/1000;
    return pot_angle;
}
```

```
int main(int argc, char **argv) {
    ros::init(argc, argv, "VSA");
    ros::NodeHandle nh;
    ros::Publisher pub = nh.advertise<geometry_msgs::Twist>("step",1000);
    ros::Publisher pub2 = nh.advertise<std_msgs::Int16>("pwm",1000);
    ros::Subscriber s1 = nh.subscribe("en1", 1000, messageCallback1);
    ros::Subscriber s2 = nh.subscribe("en2", 1000, messageCallback2);
    ros::Subscriber s3 = nh.subscribe("joystick", 1000, messageCallback3);
    ros::Subscriber s10 = nh.subscribe("res", 1000, messageCallback10);
    ros::Subscriber s5 = nh.subscribe("pot", 1000, messageCallback5);
    ros::Subscriber s7 = nh.subscribe("max", 1000, messageCallback7);
    ros::Rate loop_rate((int)(1/Dt));
    sleep(6);

    while (ros::ok()) {
        i++;
        geometry_msgs::Twist msg4;
        std_msgs::Int16 msg6;
        en1_angle = en_to_degree * en1_to_shaft * en1_val;
        en2_angle = en_to_degree * en2_to_shaft * en2_val;
        en_diff = en1_angle - en2_angle;
        pot_angle = flessione(pot_read,en_diff);
        msg4.linear.z = 0;
        msg6.data = pot_angle;
        if (pot_angle<145/3){
            delta_angle = 5;
        } else if (pot_angle<2*145/3){
            delta_angle = 6;
        } else {
            delta_angle = 7;
        }
        if (joystick<0){
            jR = -5;
        } else if (joystick>0){
            jR = 5;
        } else {
            jR = 0;
        }
```

```
        if (jR != 0){
            if (abs(en_diff) < delta_angle) {
                msg4.linear.z = jR;
            } else {
                msg4.linear.z = 0;
            }
        }
        if (jR != 0){
            if (abs(en_diff) < delta_angle) {
                msg4.linear.z = jR;
            } else if (abs(en_diff) > 1.35*delta_angle) {
                msg4.linear.z = inseguimento(en_diff);
            }
        } else {
            if (abs(en_diff) > 1.35*delta_angle) {
                msg4.linear.z = inseguimento(en_diff);
            } else {
                msg4.linear.z = 0;
            }
        }
        if (abs(en1_angle) >= max) {
            msg4.linear.z = 0;
            joystick = 0;
        }
        x = pot_angle/145*H_max;
        C = (4*Ra*Ra*E*J*tan(en_diff*3.14159265/180))/(x*x*x/6-H_max*x*x/4);

        if (i=1000){
            ROS_INFO("delta_angle: [%d]",delta_angle);
            ROS_INFO("coppia: [%d]",C);
        }
        pub.publish(msg4);
        pub2.publish(msg6);
        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```

Acknowledgements

References

1. **Vanderborght, Bram and al.** Variable impedance actuators: A review. *Robotics and Autonomous Systems.* December 2013, pp. 1601-1614.

2. **Pratt, Gill A and Williamson, Matthew M.** Series elastic actuators. *IEEE International Workshop on Intelligent Robots and Systems.* 1990.

3. **Tagliamonte, Nevio Luigi, et al.** Double actuation architectures for rendering variable impedance in compliant robots: A review. *Mechatronics.* 2012, 22.

4. **Van Ham, Ronald, et al.** Compliant actuator designs. *IEEE Robotics & Automation Magazine.* September, 2009, Vol. 16, 3.

5. **Albu-Schaeffer, Alin, et al.** Viactors. *https://www.viactors.org/.* [Online] https://www.viactors.org/.

6. **Carloni, Raffaella, Visser, Ludo C and Stramigioli, Stefano.** Variable Stiffness Actuators: A Port-Based Power-Flow Analysis. *IEEE Transactions on Robotics.* February, 2012, Vol. 28, 1.

7. **Jafari, Amir, et al.** A Novel Actuator with Adjustable Stiffness. *IEEE/RSJ International Conference in Intellligent Robots and Systems.* October, 2010, Vol. 18, 22.

8. **Jafari, Amir, Tsagarakis, Nikos and Caldwell, Darwin.** AwAS-II: A New Actuator with Adjustable Stiffness based on the Novel Principle of Adaptable Pivot point and Variable Lever ratio. *IEEE International Conference on Robotics and Automation.* 2011.

9. **Tonietti, Giovanni, Schiavi, Riccardo and Bicchi, Antonio.** Design and Control of a Variable Stiffness Actuator for Safe and Fast Physical Human/Robot Interaction. *IEEE International Conference on Robotics and Automation.* April, 2005.

10. **Schiavi, Riccardo, et al.** VSA-II: a Novel Prototype of Variable Stiffness Actuator for Safe and Performing Robots Interacting with Humans. *IEEE International Conference on Robotics and Automation.* May, 2008.

11. *VSA-CubeBot: a modular variable stiffness platform for multiple degrees of freedom robots.* **Catalano, Manuel, et al.** Shanghai : IEEE INternational Conference on Robotics and Automation, 2011.

12. *A New Variable Stiffness Design: Matching Requirements of the Next Robot Generation.* **Wolf, Sebastian and Hirzinger, Gerd.** Pasadena, CA, USA : IEEE International Conference on Robotics and Automation, 2008.

13. **Robinson, Jacob Marc.** A Compliant Mechanism-Based Variable-Stiffness Joint. s.l. : Brigham Young University , 2015.

14. **Rotary encoder.** ***Best Microcontroller Projects.*** **[Online] https://www.best-microcontroller-projects.com/rotary-encoder.html.**

15. How I2C communuication works and howto use it with Arduino. ***How to Mechatronics.*** **[Online] https://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/.**

16. Arduino 101 timers and interrupts. ***Robotshop.*** **[Online] https://www.robotshop.com/community/forum/t/arduino-101-timers-and-interrupts/13072.**

17. Introduction. ***Wiki ROS.*** **[Online] http://wiki.ros.org/ROS/Introduction.**