POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

Smart soft robot: integration of soft robotics with machine learning processes

Supervisor: prof. Giuseppe BUCCA

Co-Supervisor: Ing. Pierpaolo RUTTICO

Federico RUSCONI

875783

Academic Year 2017/18

RINGRAZIAMENTI

Desidero innanzi tutto ringraziare il Professor Giuseppe Bucca per i preziosi insegnamenti durante questi anni di studi e per le ore e le risorse investite nella mia tesi. Inoltre, ringrazio sentitamente il laboratorio IndexLab e più in particolare il suo responsabile, l'ingegner Pierpaolo Ruttico, per aver messo a mia disposizione un luogo di lavoro tanto accogliente quanto ricco di strumenti utili a portare a termine il lavoro nel miglior modo possibile. Vorrei soprattutto ringraziare con affetto la mia famiglia e in particolare i miei genitori, per il sostegno ed il grande aiuto che mi hanno dato. Senza di loro il raggiungimento di questo importante traguardo non sarebbe stato possibile. Un ringraziamento è doveroso anche alla mia ragazza, Alice, per essermi stata vicina ogni momento durante questi due anni di studio. Ringrazio poi mia zia, Maria Domenica, per aver speso del tempo nella correzione del mio inglese sicuramente non perfetto. Infine, ho desiderio di ringraziare con tutto il cuore la mia cara nonna per avermi sostenuto in tutti questi anni di studio, spero di regalarle una grande gioia con questo mio traguardo.

ABSTRACT

The thesis work described in the following pages is developed around the field of robotics and artificial intelligence. In particular, in this work the field of *soft robotics* and *machine learning* is taken into consideration, focusing on the integration of these topics among them. As regards the part related to soft robotics, finite element analysis was performed to theoretically study the behavior of the soft robot according to some design parameters, in order to choose the most suitable solution. Then the work focused on the development of tools and on a process able to create the soft robot, and on the realization of an electric circuit to control, by means of suitable actuators, the movement of the soft robot itself. The crucial point of work is the integration of a sensor inside the robot that can monitor its behavior. Thanks to this sensor it was possible to make data acquisition useful to create predictive models through machine learning techniques. Considering the part inherent to machine learning, the work does not deal with the development of specific algorithms but rather with the integration of this area with soft robotics. In particular, a MATLAB code, able to process the acquired data making it suitable for conventional machine learning processes through the MATLAB tool called *Classification Learner*, was created. The goal was to create a predictive model that can recognize if an object touched by the soft robot is hard or soft. The last part of the work is focused on predicting the feature of an unknown object using previously developed models and on the analysis of the results of the predictions themselves.

Il lavoro di tesi descritto nelle pagine seguenti è sviluppato attorno al campo della robotica e dell'intelligenza artificiale. In particolare, in questo lavoro viene preso in considerazione l'ambito della *soft robotics* e del *machine learning*, focalizzando l'attenzione sull'integrazione di questi argomenti tra di loro. Per quanto riguarda la parte relativa alla robotica, si è eseguita un'analisi ad elementi finiti per studiare teoricamente il comportamento del robot in base ad alcuni parametri di progetto, al fine di scegliere la soluzione più adatta. Quindi il lavoro si è concentrato sullo sviluppo di strumenti e su un processo in grado di creare il robot, e sulla realizzazione di un circuito elettrico per controllare il movimento del robot stesso attraverso attuatori scelti in modo appropriato. Punto cruciale del lavoro è l'integrazione di un sensore all'interno del robot in grado di monitorare il suo comportamento. Grazie a questo sensore è stata possibile l'acquisizione di dati utili per creare modelli predittivi attraverso tecniche di apprendimento automatico. Considerando la parte inerente al machine learning, il lavoro non affronta lo sviluppo di algoritmi specifici, ma bensì l'integrazione di quest'area con la soft robotics. In particolare, si è creato un codice MATLAB, in grado di elaborare i dati acquisiti rendendoli adatti ai processi di apprendimento automatico convenzionali attraverso un tool di MATLAB chiamato *Classification Learner*. L'obiettivo è creare un modello predittivo in grado di riconoscere se un oggetto toccato dal robot sia duro o morbido. L'ultima parte del lavoro si è concentrata sulla predizione della caratteristica di un oggetto

ignoto utilizzando i modelli precedentemente sviluppati e sull'analisi dei risultati delle predizioni stesse.

TABLE OF CONTENT

INDEX OF IMAGES

1. INTRODUCTION

The work described in the next pages will be developed in the field of robotics. Inside robotics, there is a specific application that is known as “soft robotics”. In this specific technology soft materials and compressed air are used to create motion of components that can work for example as actuators. Therefore, it is a quite cheap technology adaptable to different applications. The growing importance of robotics will also be followed step by step by a massive use of artificial intelligence (AI). Nowadays it is easy to find AI in different common applications, such as smartphones or cars, synonymous with a growing interest in developing this technology. In industrial application AI will increase the flexibility of the process, making the industrial machines “more intelligent” with a consequent improvement of the performances. Hence, the decision to introduce in the thesis work also a part related to the so-called machine learning. This process is responsible for the creation of a large database and trained model that can be used in artificial intelligence application.
The work will therefore be focused on the development of a smart soft robotics component able to be trained with machine learning techniques.

STATE OF THE ART

Soft Robotics

Soft robotics technologies use soft materials like those found in living organisms. The objective of soft robotics is to create components or more in general robots that can adapt to the surrounding environment increasing flexibility and adaptability for accomplishing tasks, as well as improving safety when working around humans. These features allow soft robotics to be used in medicine and manufacturing application. Soft robots can be implemented in the medical profession, specifically for invasive surgery. Soft robots can be made to assist surgeries due to their shape-changing properties. Shape change is important as a soft robot could navigate around different structures in the human body by adjusting its form. Soft robots may also be used for the creation of flexible exosuits, for rehabilitation of patients, assisting the elderly, or simply enhancing the user’s strength. Traditionally, manufacturing robots have been isolated from human workers due to safety concerns, as a rigid robot colliding with a human could easily lead to injury due to the fast-paced motion of the robot. However, soft robots could work alongside humans safely, as in a collision the compliant nature of the robot would prevent or minimize any potential injury. The most common material used is silicone but there are many applications in which other soft materials such as TPU can be considered.

Figure 1-1 Soft robotic rehabilitation glove

Fluids are used for the actuation of soft robots. Generally, compressed air is the most common and simplest solution.
Nowadays soft robotics is integrated also in many manufacturing applications and a lot of grippers designed with this technology exists. Due to the high flexibility of soft robotics technology, they are mainly used in the food industry or industrial processes that involve light, small or fragile objects. Nevertheless, until now this gripper is not integrated with sensors so that their “intelligence” is null.

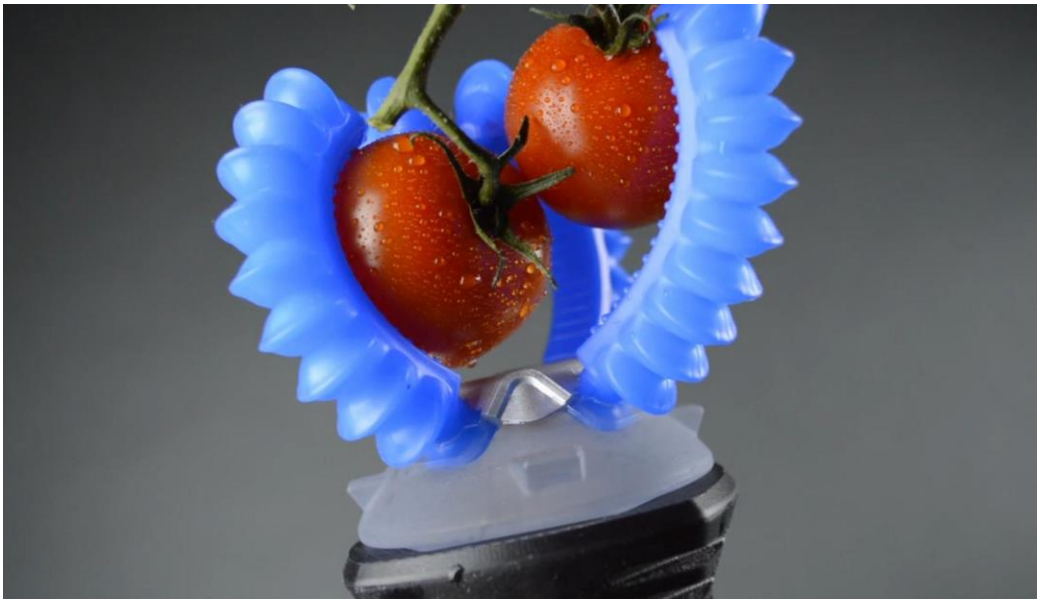

Figure 1-2 Soft robotic industrial gripper

Machine Learning

Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) with data, without being explicitly programmed.
The name machine learning was coined in 1959 by Arthur Samuel. Evolved in artificial intelligence from the study of pattern recognition and computational learning theory,

machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms overcome strictly static instructions of conventional programs by making data, driven predictions or decisions, through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance are difficult or infeasible.

Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data.

Tom M. Mitchell provided a widely quoted, more formal definition of the algorithms studied in the machine learning field: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." This definition of the tasks in which machine learning is concerned offers a fundamentally operational definition rather than defining the field in cognitive terms. This follows Alan Turing's proposal in his paper "Computing Machinery and Intelligence", in which the question "Can machines think?" is replaced with the question "Can machines do what we (as thinking entities) can do?".

Machine learning tasks are typically classified into different categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:

- Supervised learning: the computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback.
- Semi-supervised learning: the computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.
- Active learning: the computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labeling.
- Reinforcement learning: training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.
- Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

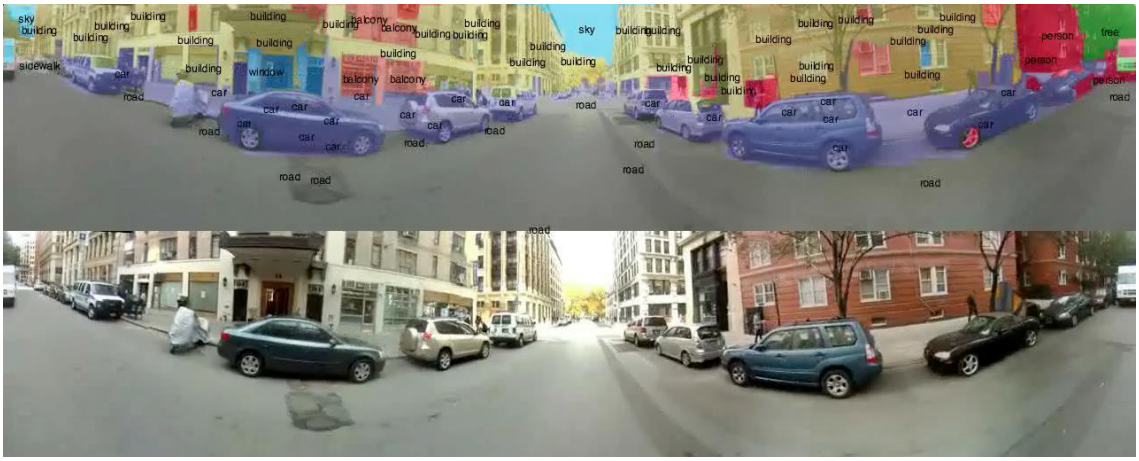

Figure 1-3 Result of AI on Google car

There is no shortage of machine learning algorithms. They range from the fairly simple to the highly complex. Since this work is not focused on the development of algorithms but on their integration in the soft robotic technology only the most common and those that will be used later are briefly described:

- Decision tree learning: Decision tree learning uses a decision tree as a predictive model, which maps observations about an item to come to conclusions about the item's target value.
- Artificial neural networks: An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is vaguely inspired by biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.
- Deep learning: Falling hardware prices and the development of GPUs for personal use in the last few years have contributed to the development of the concept of deep learning which consists of multiple hidden layers in an artificial neural network. This approach tries to model the way the human brain processes light and sound into vision and hearing. Some successful applications of deep learning are computer vision and speech recognition.
- Support vector machines: Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.
- K-nearest neighbors' algorithm: the k-nearest neighbors' algorithm (k-NN) is a non-parametric method used for classification and regression. In k-NN classification, the output is a class membership. An object is classified by a

majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors. K-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally, and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

Figure 1-4 Example of AI application in domotic field

Although machine learning has been transformative in some fields, effective machine learning is difficult because finding patterns is hard and often not enough training data are available; as a result, many machine-learning programs often fail to deliver the expected value. Reasons for this are numerous: lack of (suitable) data, lack of access to the data, data bias, privacy problems, badly chosen tasks, and algorithms, wrong tools and people, lack of resources, and evaluation problems.

Machine learning approaches, in particular, can suffer from different data biases. A machine learning system trained on your current customers only may not be able to predict the needs of new customer groups that are not represented in the training data. When trained on man-made data, machine learning is likely to pick up the same constitutional and unconscious biases already present in society. Language models learned from data have been shown to contain human-like biases. Machine learning systems used for criminal risk assessment have been found to be biased against black people. In 2015, Google photos would often tag black people as gorillas, and in 2018 this still was not well resolved, but Google reportedly was still using the workaround to remove all gorilla from the training data, and thus was not able to recognize real gorillas at all. Similar issues with recognizing non-white people have been found in many other systems. In 2016, Microsoft tested a chatbot that learned from Twitter, and it quickly

picked up racist and sexist language. Because of such challenges, the effective use of machine learning may take longer to be adopted in other domains. In 2018, a self-driving car from Uber failed to detect a pedestrian, who got killed in the accident. Attempts to use machine learning in healthcare with the IBM Watson system failed to deliver even after years of time and billions of investments.

Figure 1-5 Autonomus Uber car

Machine learning algorithms have been around for decades, but they've attained new popularity as artificial intelligence (AI) has grown in prominence. Deep learning models in particular power today's most advanced AI applications.
Machine learning platforms are among enterprise technology's most competitive realms, with most major vendors, including Amazon, Google, Microsoft, IBM and others, racing to sign customers up for platform services that cover the spectrum of machine learning activities, including data collection, data preparation, model building, training and application deployment. As machine learning continues to increase in importance to business operations and AI becomes ever more practical in enterprise settings, the machine learning platform wars will only intensify.
Continued research into deep learning and AI is increasingly focused on developing more general applications. Today's AI models require extensive training in order to produce an algorithm that is highly optimized to perform one task. But some researchers are exploring ways to make models more flexible and able to apply context learned from one task to future, different tasks.

The work will be focused on the development of a soft robot able to recognize the object touched, simulating the behavior of human fingers. In the first part, a finite element model of the soft robot will be created to theoretically study its behavior. Then the work will proceed with the design and the development of the soft robot itself and the components able to help the realization of this technology. Then the work will be focused on the motion of the Soft robotic finger and the integration of sensors that collect data about the object touched. The last part of the thesis will be about machine learning including the training of the model and the prediction of unknown data.

2. Finite Element Model Of The Soft Robot

First it is requested to design the soft robot. For the declared purpose this robot must have the rough shape of a finger and a bending motion is required to allow the touching and grabbing of objects.
The idea is to create a series of channels and chambers. These channels inflate when pressurized, creating motion. The nature of this motion is controlled by modifying the geometry of the embedded chambers.
To improve the desired motion different materials can be used in combination to silicone. For this reason, it is possible to put inside rubber layers of materials with different elastic behavior. The "stretchy" material will expand more than the "rigid" material when the robot is pressurized. The "differential strain" effect can be used to achieve useful motions such as bending and twisting.
The simplest and cheapest material that can be combined with silicone is a piece of paper.

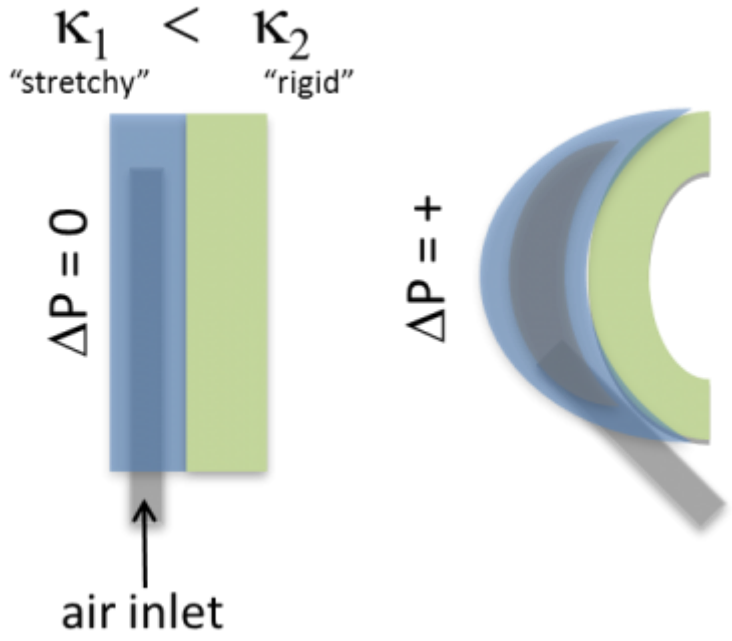

Figure 2-1 Soft robotics physical principle

Finite element analysis can be used to verify the theoretical principle introduced above. Furthermore, with different analysis it is possible to understand how the geometry of the chamber can modify the motion of the soft robot. FE analysis have been made with Abaqus.
In particular, the analyses are carried out on four different solutions, that differ one from another with regard to two parameters. The first one is the length of the chambers, that is reflected in the number of chambers itself. In two scenarios chamber of 5 mm length are

considered, in the remaining cases this dimension is equal to 10 mm. The length of the chamber will be called ‘w’. Parameter w influences the chamber number, as this quantity increases when the length decreases. The second dimension is the depth of the space between one chamber and the following one. For this parameter, values of 6 mm and 11 mm are taken into account. This parameter will be called ‘h’ (*figure 2-2*).
Below the four solutions proposed for the analysis are summarized.

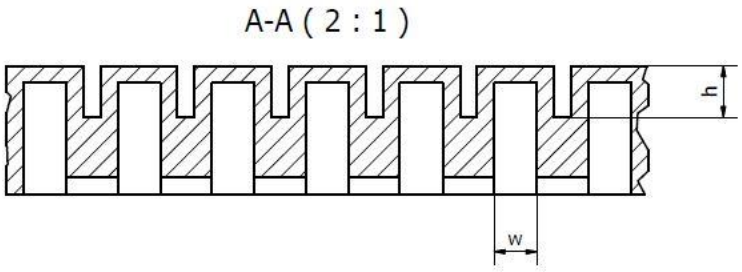

Figure 2-2 Soft robot's section

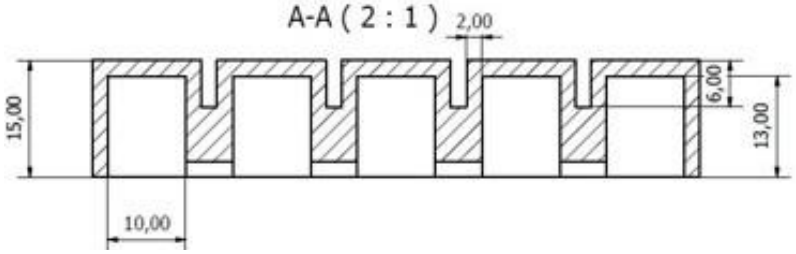

Figure 2-3 Model r04 section

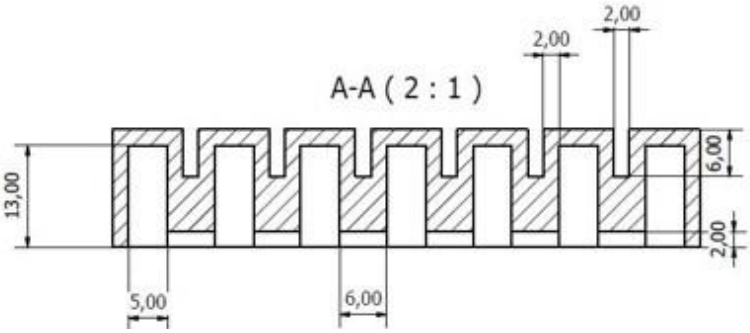

Figure 2-4 Model r05 section

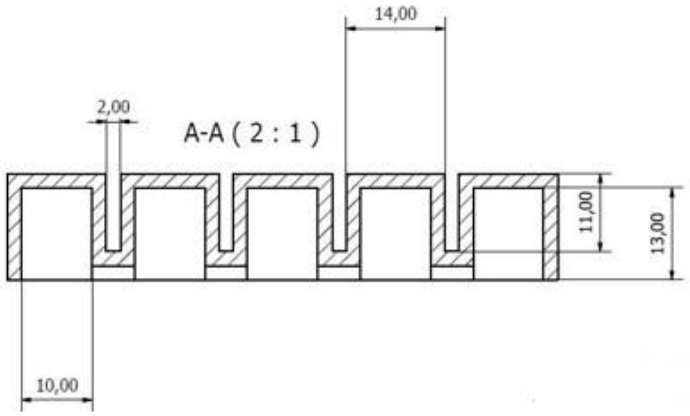

Figure 2-5 Model r07 section

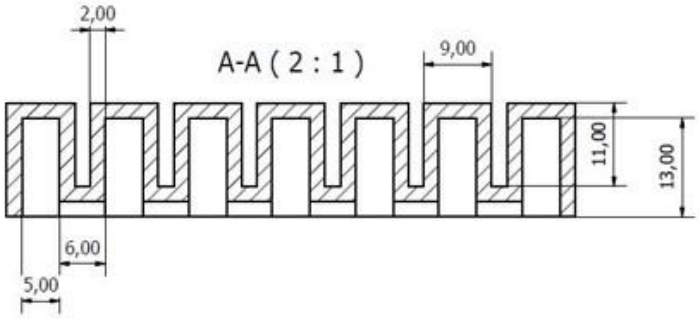

Figure 2-6 Model r08 section

In the following lines, it will be explained the procedure to obtain a finite element analysis of the soft robot, with particular regard on the chosen parameters. As this procedure is equal considering the four cases, only one of them is considered.

FE ANALYSIS PARAMETER

All the analysis consider the soft robot divided in three different parts. The first one is the one related to the inflation chamber, the other two are equal to each other and represent the below part of the soft finger. With the assembly tool of Abaqus it is possible to combine the three different parts and merge them in a single component.

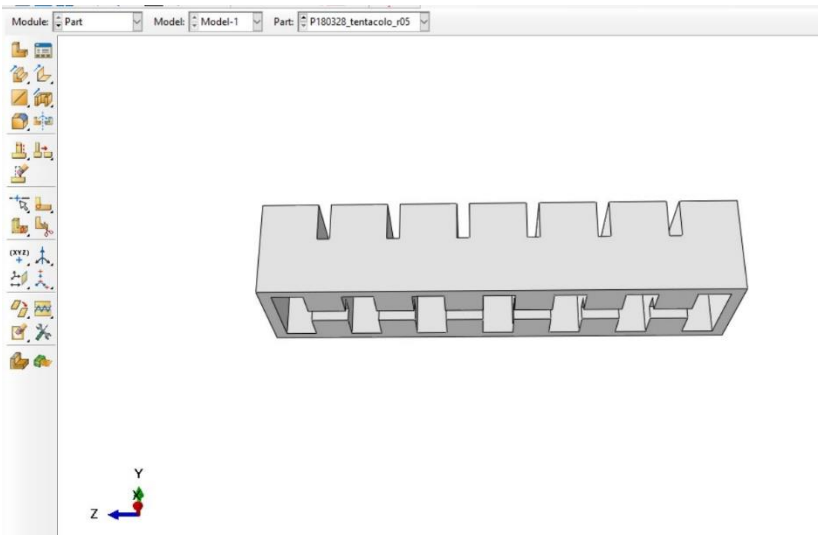

Figure 2-7 Upper part of the soft robot

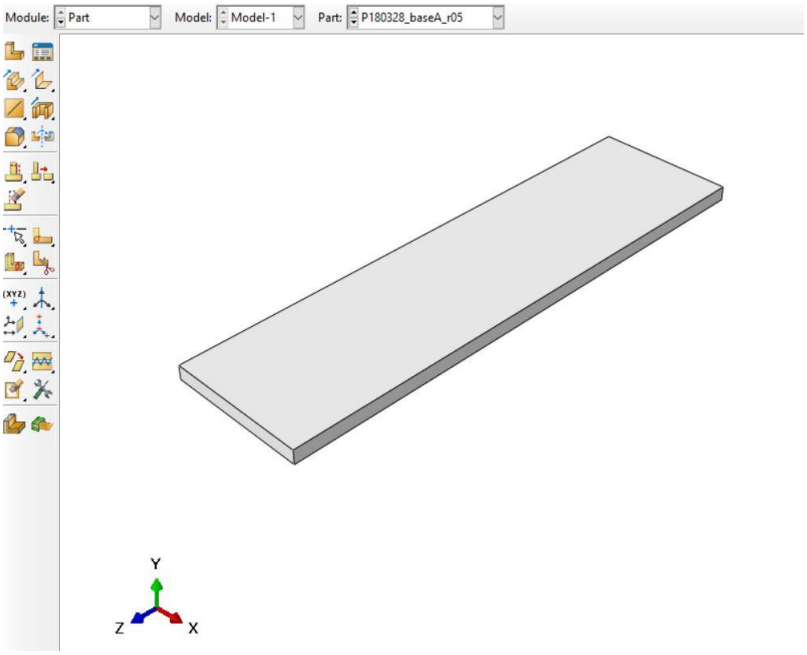

Figure 2-8 Bottom part of the soft robot

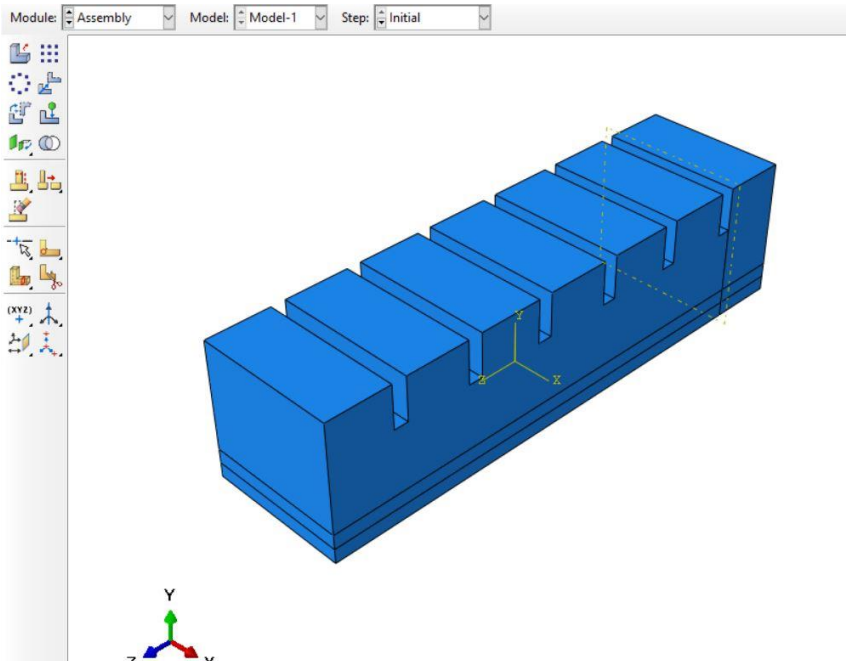

Figure 2-9 3D model of the soft robot

The materials considered in the analysis are simple silicone and paper. These materials have the following properties, useful for the analysis:

- Silicone rubber: Yeoh strain energy potential defined by the coefficients C10 = 0.11, C20 = 0.02. Density of 1130 Kg/m³, assumed to be hyperelastic and isotropic.
- Paper: density of 750 Kg/m³, a Young's Modulus of 6.5 Gpa and a Poisson's ratio of 0.2.

As explained before, to achieve the desired motion a piece of paper needs to be put inside the silicone rubber. In FE analysis to simulate the inextensible layer, a surface on the upper part of the second bottom layer is created. Once this surface is created, it is required to create a skin on this surface by means of the homonymous command. Now it is necessary to create the section to assign the created materials. Again, two different sections are required. The first one is related to the silicone rubber and it is defined as uniform solid. This section is assigned to the main body and to the two bottom layers. The second one, assigned to the inextensible layer, is defined as uniform shell.

Before introducing loads and constraints it is necessary to create two steps for the analysis. The first one is referred to the gravity effects. It is important to set this step so that non-linear effects of large displacement are included. The second step is the one responsible for the effect of the pressurized air. Being subsequent to the gravity step, also the pressure step takes into account the non-linearity.

For each step it's necessary to create a specific load. For the first step to represent the gravity action it is possible to select the 'Gravity' option to assign this type load on the

vertical direction according to the soft robot and insert the correspondent value. In this way, the gravity effect will affect the whole component. Before creating the load related to the pressurized air action, it is necessary to create the surface on which will act the air itself. This new surface includes all the inside chamber walls and the connecting channels. These are all the surfaces where the pressurized air exercises its action. Once the inner surface is created it is possible to set the load related to the second step.

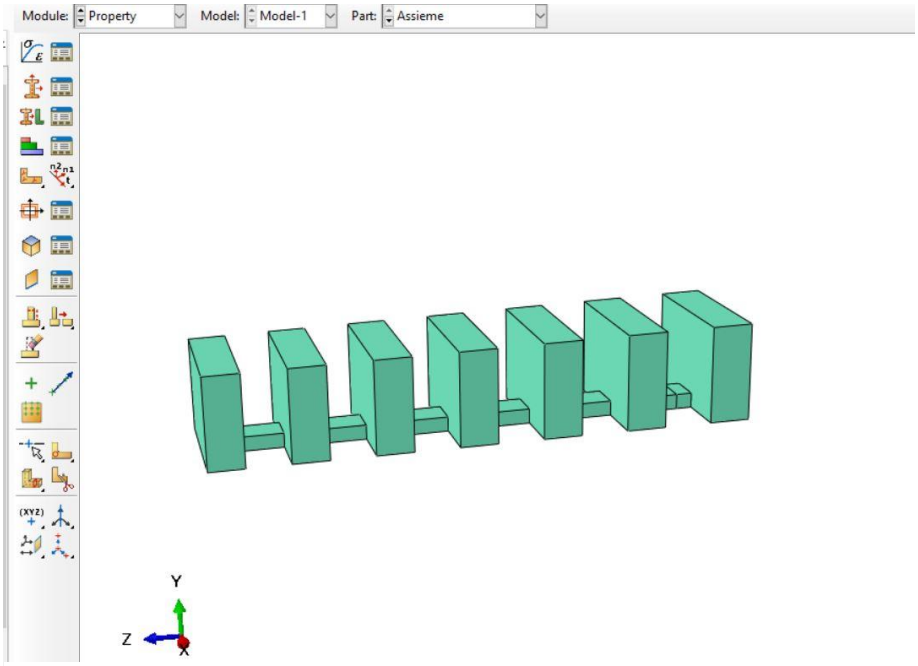

Figure 2-10 Inner surface

Regarding the boundary conditions, they are the same passing from the gravity step through the pressure step. They can be created for the first step and then extended to the second one. To simulate the complete clamp of the soft robot, the boundary conditions are modeled as 'Encastre', and assigned to the external wall of the first chamber.

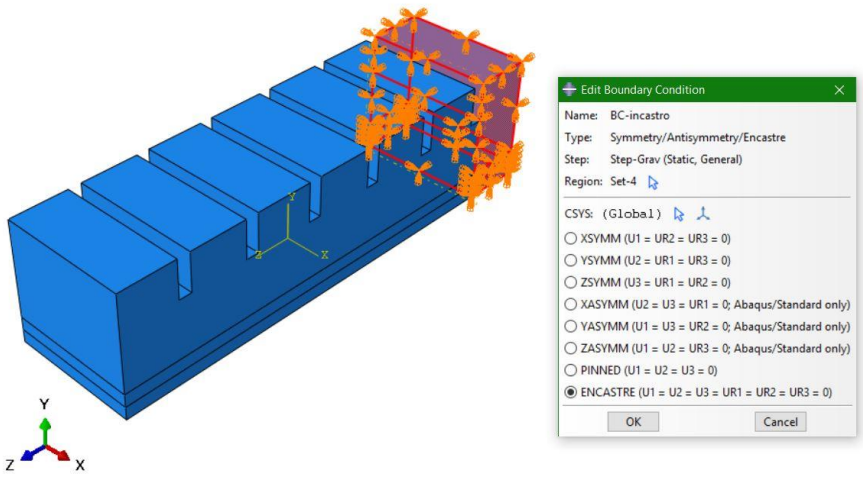

Figure 2-11 Boundary conditions

When the soft robot is sufficiently inflated, the walls of adjacent chambers will come into contact with each other. However, Abaqus explicitly needs to be told to take this physical interaction into account; otherwise, the modeled result will have the walls simply passing through each other, as seen below:

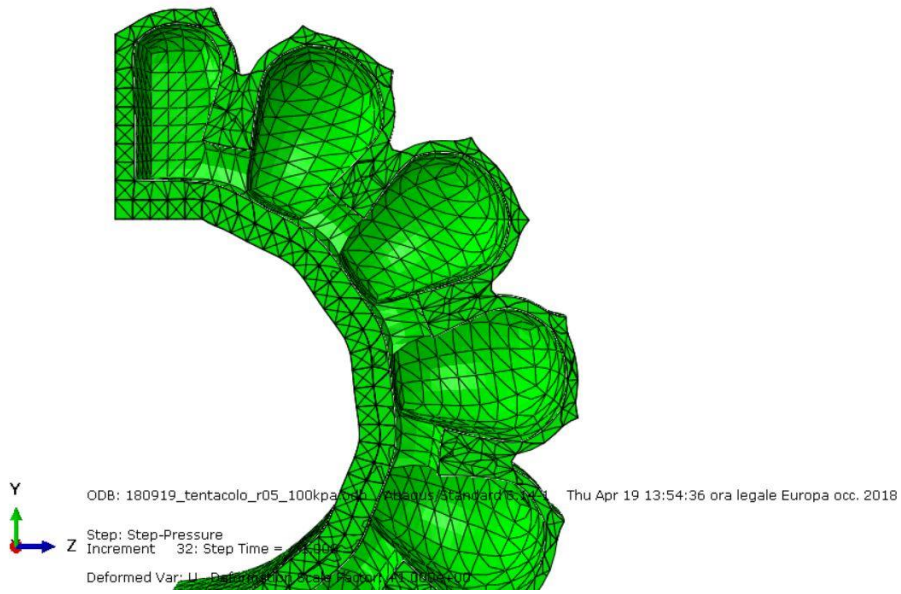

Figure 2-13 Inflated soft robot's section

The interactions have to be modeled as a contact. In particular, it is modeled choosing the 'Tangential Behavior' option.
Being the contact between the surface of the same body, the interaction is modeled as 'Self-contact' and assigned to the related surfaces.

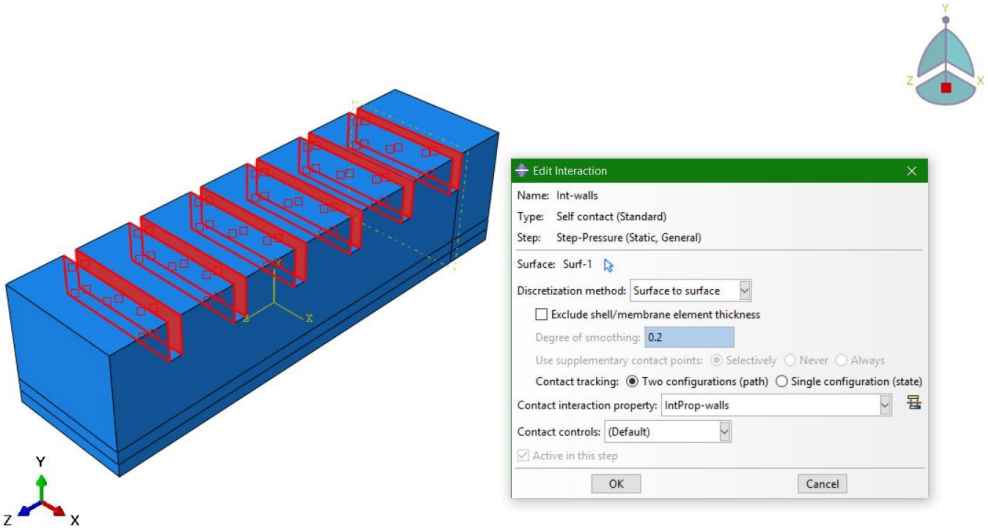

Figure 2-14 Contact interaction parameters

The last step is the meshing of the soft robot. From the ‘Mesh Controls’ tetrahedral elements with free techniques are selected. A crucial part is the selection of the ‘Approximate global size’ of the elements.

If the mesh size is too small, the model may become “stiff” to high distortions. But if it is too big, the mesh will not fit on well on the model face. Here a dimension of 2 mm is considered. The choice of mesh dimension is also conditioned by the computational power of the laptop used for the analysis.
Because we are dealing with hyperelastic material, a Hybrid element type for the mesh should be used. This option is related only for the part made of silicone. For the paper layer, this option is not requested. For both the mesh type the ‘Geometric Order’ must be ‘Quadratic’ to have a correct representation of the bending action. Once the mesh is set in the right way, the simulation is ready to be launched.

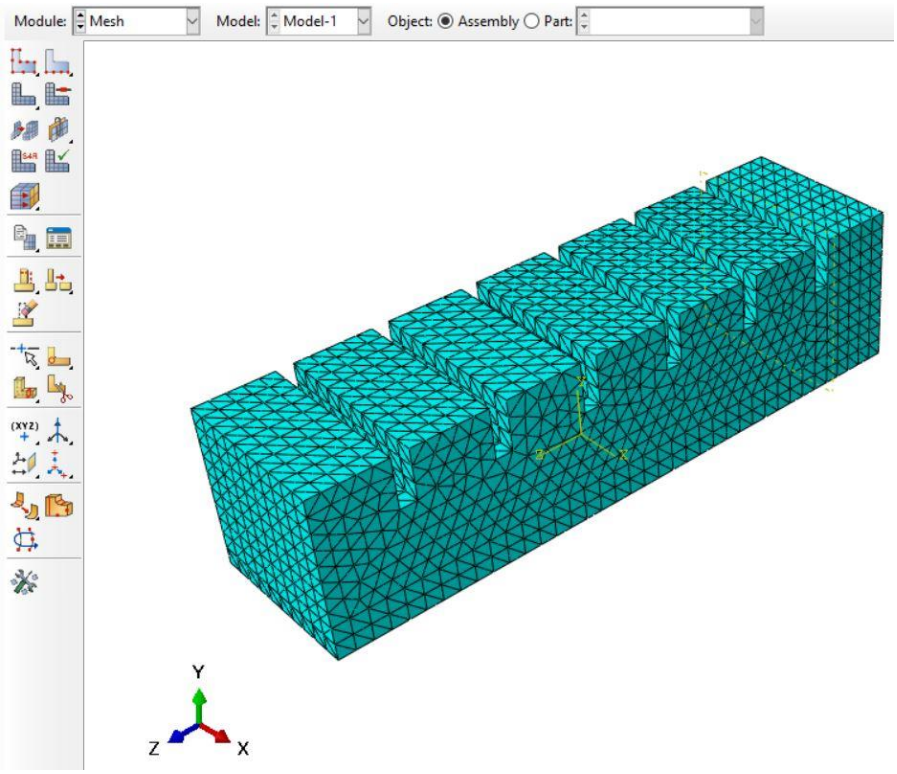

Figure 2-15 Mesh

FE ANALYSIS RESULTS

As said before the analysis has been performed for the four selected versions of the soft finger. To have the best knowledge of the soft robot behavior for each solution, ten different simulations with an increasing value of air pressure have been done, for a total of forty simulations.
In the comparison between the different solutions, two parameters able to describe the behavior of the soft robots are requested. These two parameters are the vertical displacement, indicated as 'U2', and the angle due to the deformation, called 'ϑ'.

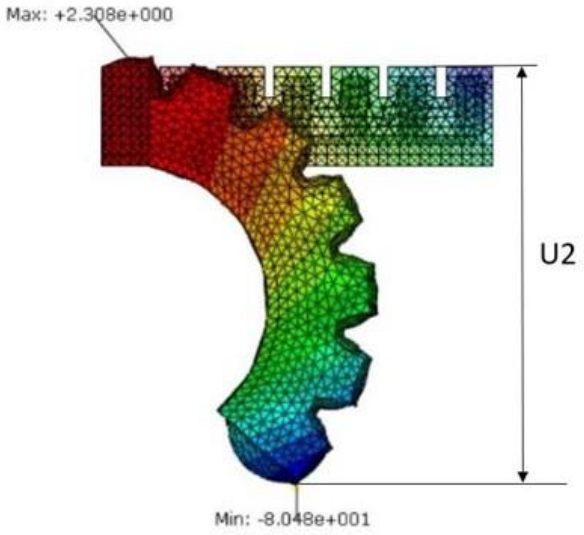

Figure 2-16 Vertical displacemet U2

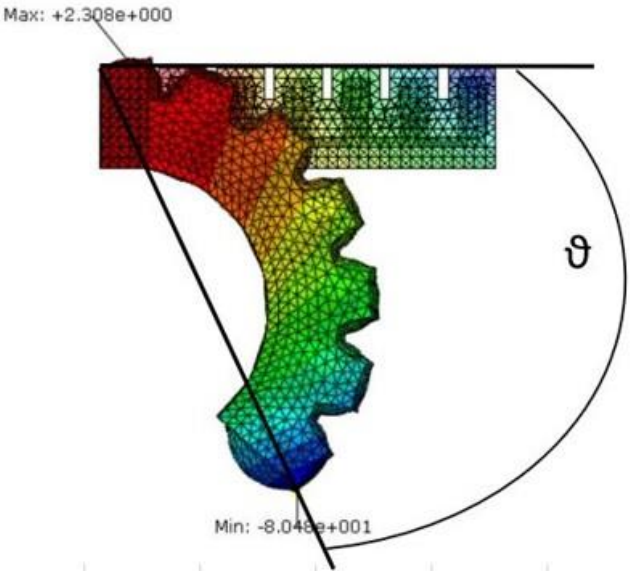

Figure 2-17 Curvature ϑ

In the following tables and graphs the results of the FE analysis are shown.

		U2 [mm]			
		R04	R05	R07	R08
		h=6 ; w=10	h=6; w=5	h=11; w=10	h=11; w=5
PRESSURE [kpa]	0	3.35	4.42	12.08	17.23
	10	11.01	17	27.21	42.56
	25	23.93	36.46	48.1	68.94
	40	38.19	54.53	66.06	80.56
	50	47.81	64.6	75.08	79.75
	60	57.14	72.73	81.72	71.99
	75	69.87	80.84	80.07	49.14
	80	73.56	81.91	77.2	39.57
	90	79.77	82.93	67.49	20.23
	100	84.07	81.46	53.59	4.64

Table 2-1 Vertical displacement U2

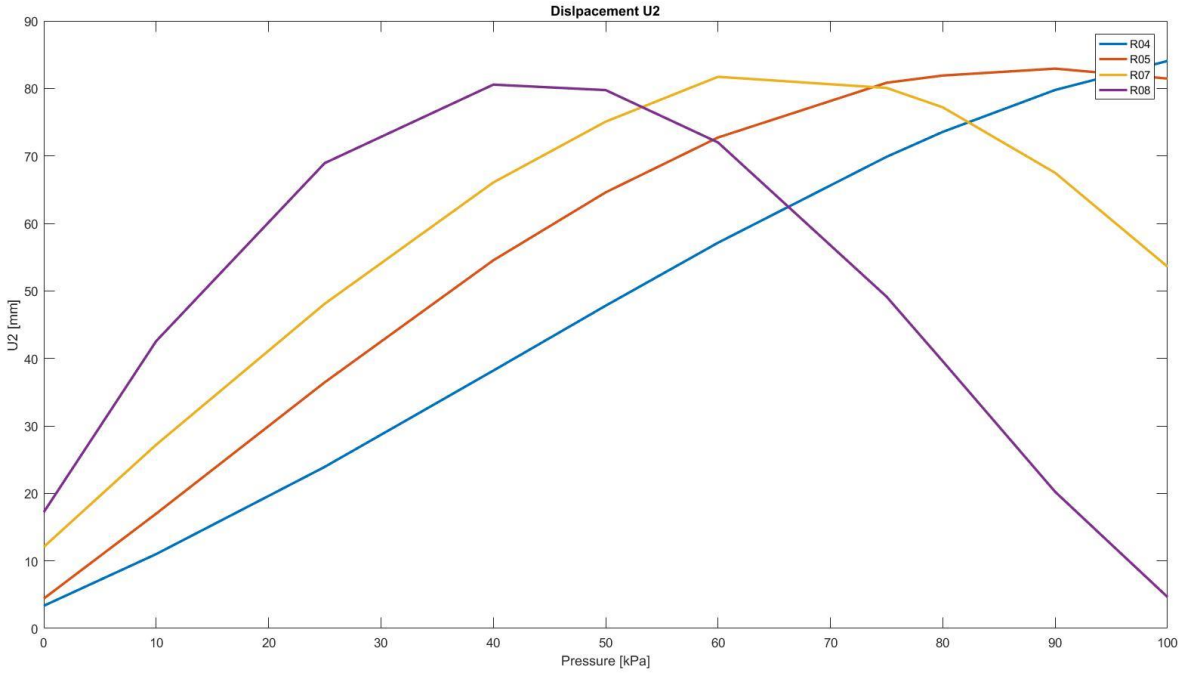

Figure 2-18 Vertical displacement comparison

		THETA [deg]			
		R04	R05	R07	R08
		h=6 ; w=10	h=6 ; w=5	h=11 ; w=10	h=11 ; w=5
PRESSURE [kpa]	0	2.46154	3.37859	8.90936	13.28137
	10	8.11462	13.10090	20.41676	34.57376
	25	17.86618	29.08675	38.07312	66.80941
	40	29.31531	46.64104	57.87857	90.14500
	50	37.80302	59.46662	74.27300	95.23111
	60	47.10178	70.01200	80.32500	106.28747
	75	63.60741	80.23600	93.21400	139.06518
	80	70.57488	85.03200	98.21310	148.15656
	90	81.17520	90.19960	120.08800	195.64840
	100	89.09600	95.31800	136.60286	266.45304

Table 2-2 Curvature ϑ

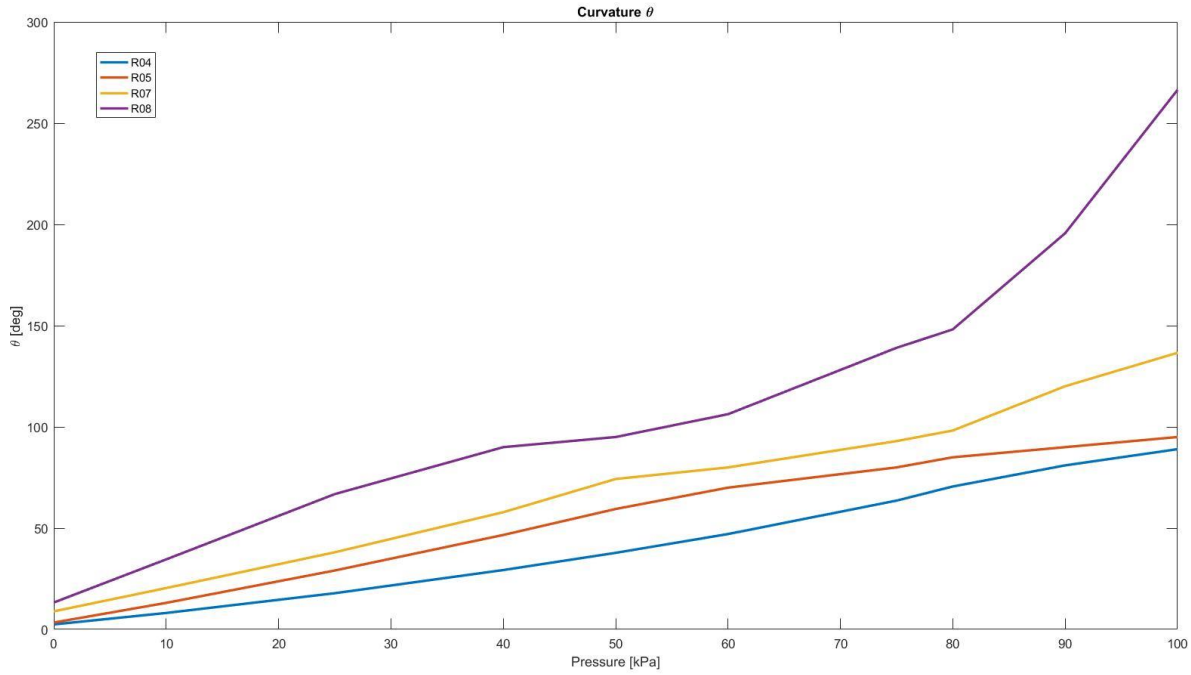

Figure 2-19 Curvature comparison

It's clear that as the pressure increases, consequently the displacement and the curvature increase too. When the pressure action is null (0 Kpa), only the gravity effect is acting on the soft finger. In this scenario, lower is the value of parameter h, the lower is the deflection of the soft robot. In some cases, it can be noticed that U2 start to decrease. This could seem wired but looking to the curvature it is possible to see that in this pressure range ϑ exceeds 90°. This means that the soft finger is rolling up on itself.

The two parameters previously declared influence the response of the soft robot. Keeping the parameter h constant and changing the length of the chamber and consequently the

number of them, it is possible to notice that both U2 and ϑ increase. Having a higher number of chambers means have more of joints, obtaining a higher dexterity of the soft robot. On the other side, keeping constant the chambers number and increasing the parameter h, the result is to obtain an higher flexibility of the finger. In fact, higher is the depth between two subsequent chambers, higher is the capability of the joint to bend more.

In conclusion, working on the two design parameters enables to obtain different behaviors of the soft robot. The dexterity of the finger is higher as the number of chambers increases and the parameter h increases too. As a drawback the finger will present an accentuated value of U2 and ϑ when the pressure value is null proving not suitable for some specific applications.

EXTRA ANALYSIS

To have a better knowledge of the soft robot behavior, some extra analysis can be done. Their aim is to investigate what happens when a negative pressure is applied to the silicone finger.
For what concern the FE analysis most of the analysis parameters remain the same and only few changes are mandatory. First, the pressure value has to be set negative. Second, additional contacts must be modeled. When negative pressure is applied, inner surfaces of the chambers could come in contact. As previously described a “self-contact” option must be set for these surfaces.
The result is the bending of the soft robots in the opposite direction. Looking at the results of analyses with different pressure values, it can be noticed that the behavior of the soft robot is quite similar for all the four proposed geometries of the finger. The design parameters do not affect in a consistent way the motion. Further, the displacement and the curvature are quite limited because the outer surfaces come in contact very soon.

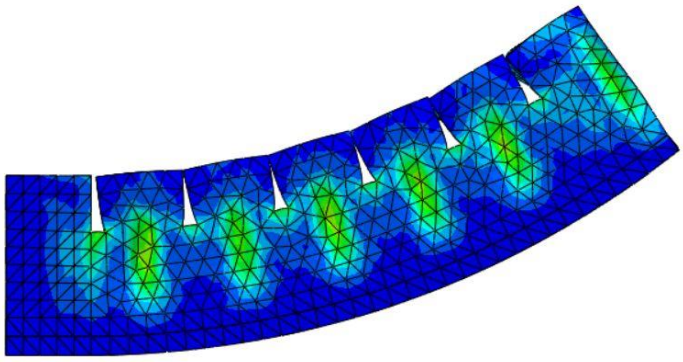

Figure 2-20 Negative pressure effects

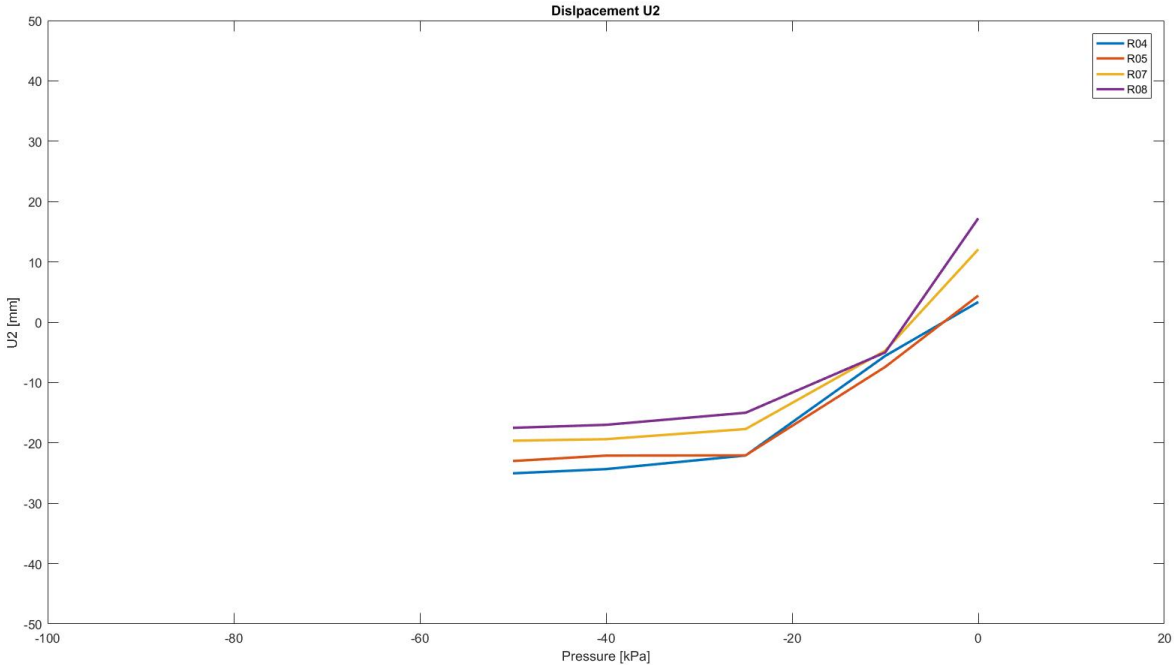

Figure 2-21 Vertical displacement comparison with negative pressure

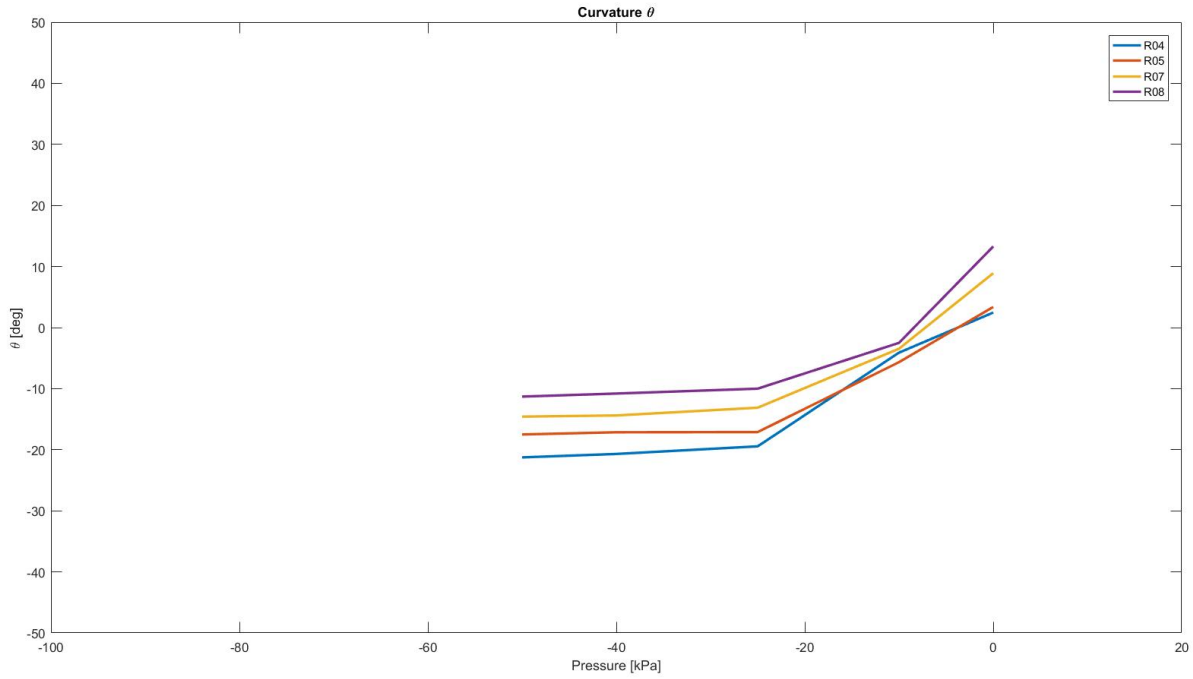

Figure 2-22 Curvature comparison with negative pressure

For pressure value higher than 50 kPa the analyses result useless, giving the same results of the 50 Kpa analysis. In fact, once the external walls of the chamber in contact, the bending action stops.

FE ANALYSIS CONCLUSIONS

Considering the application treated in this work, the four solutions can be considered suitable. The bending action and the dexterity of the robot are enough for all the proposed cases. Additional considerations can be made for other specific applications. For example, in some applications where a certain accuracy for null pressure is required, only the proposed solution that presents low value of U2 and ϑ for this specific pressure value can be considered suitable. On the other hand, if a high value of the curvature is requested, for instance a glove for hand rehabilitation, other proposed solutions can be taken into consideration

3. Design And Realization Of The Soft Robot

The first part of this chapter will discuss the construction of the robot, focusing on the design of the tools necessary for the process. The second part will instead describe the system used to move the robot and the integration of a sensor inside the robot itself. In particular, it will be briefly described the electrical circuit and the Arduino code required for the robot to move.

Molds design and prototyping

Once the behavior of the soft finger has been studied, a production process able to realize the component must be thought of. The simplest way to reproduce the designed geometry is to use different molds with certain geometry, in which the silicone rubber is injected. All the 3D drawing and prototyping have been made on Autodesk Inventor 2018. The molds have been produced by means of the 3D printing technology that allows to obtain objects with small details. A *Prusa i3 MK3* printer has been used for the realization of all the 3D printed components of this thesis work. Being the molds without particular details, the resolution of the printing process can be low, such as 0.3 mm, reducing the printing time. Moreover, thanks to the design of the components no supports are needed during the printing process.

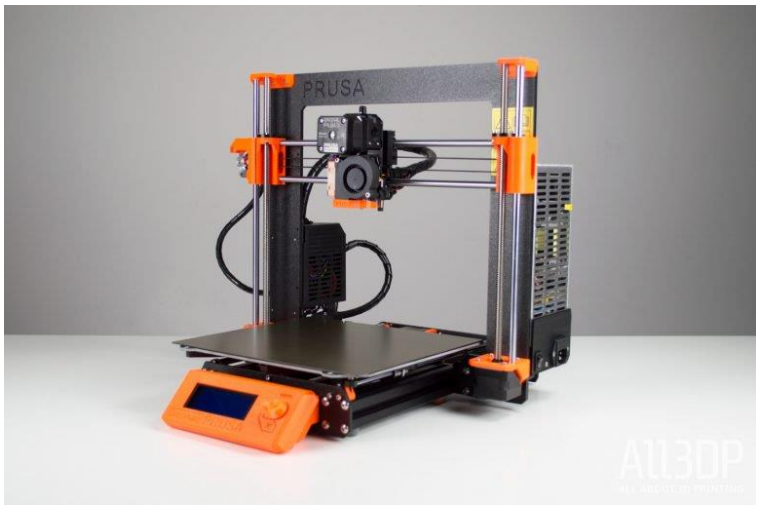

Figure 3-1 Prusa i3 MK3

To get to the final solution several attempts were made to solve problems arising during the inflation of the soft robot. All the designed solutions have a common principle to their base. A system of three molds is used, two of them for the realization of the upper part

and the remaining one for the bottom part. Considering the upper part, the two molds are assembled together. They are designed to reproduce the wanted geometry. Some specific geometries have been thought to appear as\ the negative of the soft robot shapes. Considering the first mold of the upper part, the biggest extruded parts are the ones related to the reproduction of the chambers, while the smallest ones reproduce the connection channels. The rectangular hollows are introduced to ease the production, as it will be explained later. The second mold of the upper part is simpler, and the extruded parts create the space between two consecutive chambers.

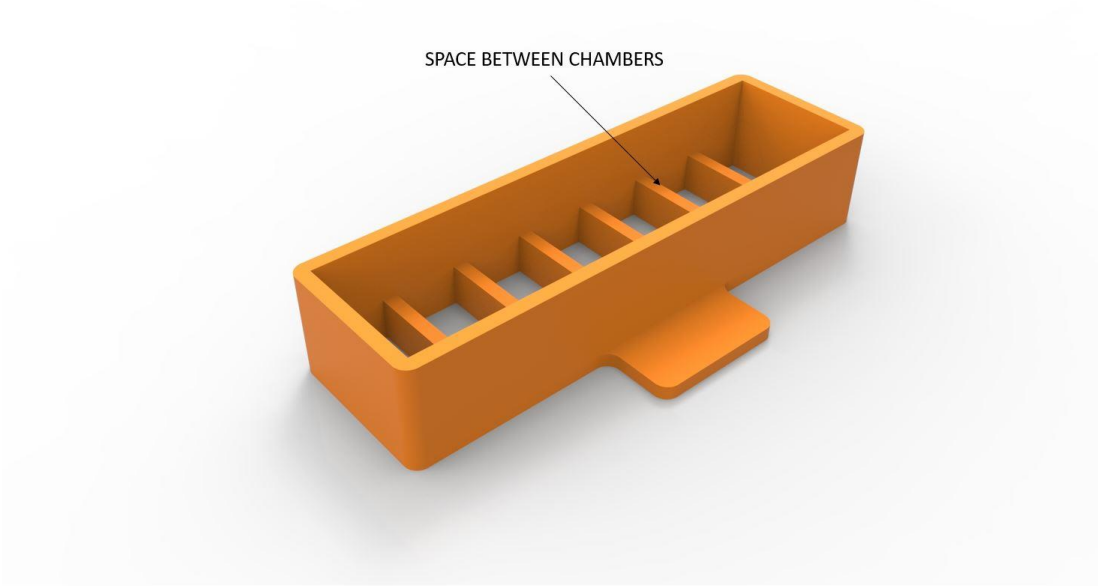

Figure 3-2 Solution 1, upper part

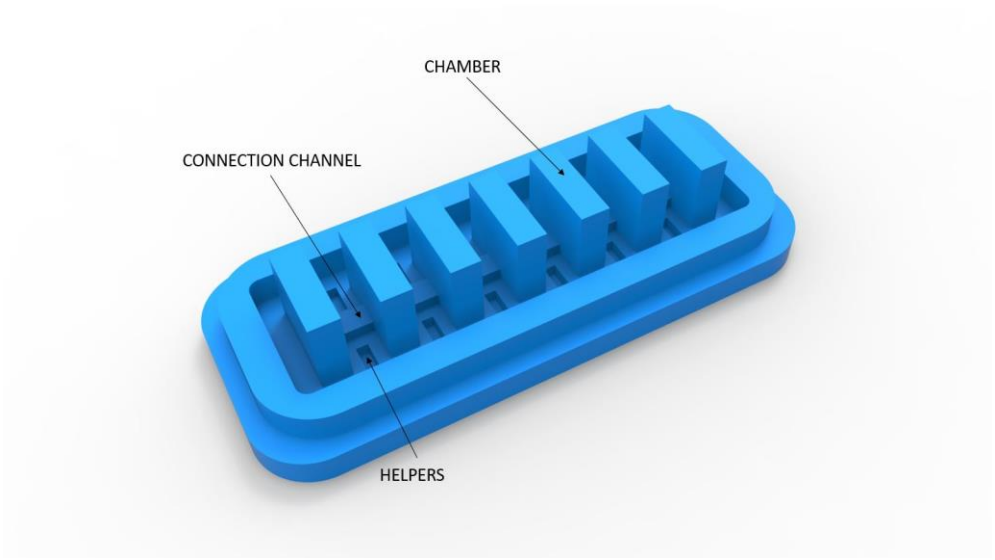

Figure 3-3 Solution 1, bottom part

Figure 3-4 Solution 1

Using solution 1 (*figure 3-4*) an air leak occurs because the tube is pulled out from the silicone by the air itself. The air leakage increases when the soft robot meets the resistance of an external object that tends to squeeze it. For this reason, the so made soft finger is not suitable for touching objects. Air leakage problem can be solved using some rubber bands, but this solution is not feasible. Furthermore, thinking about future applications, it is necessary to design a clamping system in such a way that the soft robot could be placed in specific spots, according to the final application.
Solution 2 (*figure 3-5*) expects the presence of a clamping system. For this reason, both the molds of the upper part have been modified so that this system can be placed inside the silicone. The new component is designed to have holes that can be used to mount the robot with screws. The pneumatic tube is inserted in an additional hole in the clamping system. Solution 2 expects an issue. The clamping system is made of PLA and is too stiff compared with the silicone rubber. When the soft robot bends the PLA part cannot follow the same deformation of the softer material, creating some small glimmers from which the pressurized air flows. Again, this phenomenon is accentuated when the robot impacts hard objects.

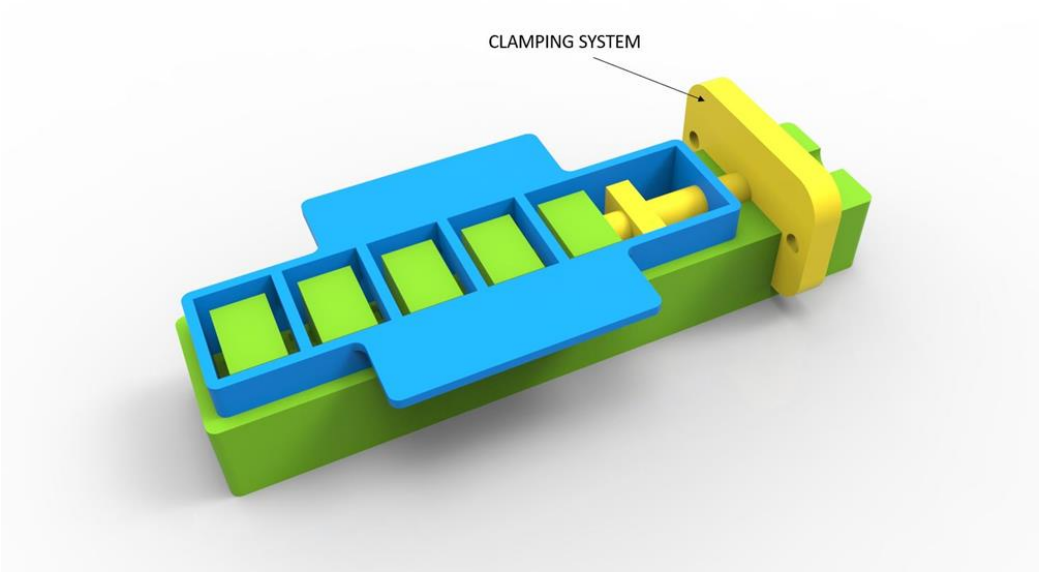

Figure 3-5 Solution 2

To try to reduce the air leaks, on the molds of the upper part some geometries are introduced creating a part of silicone around the cylindrical section of the clamping system (*figure 3-8*). Nevertheless, also with solution 3 the previously described problem has returned.

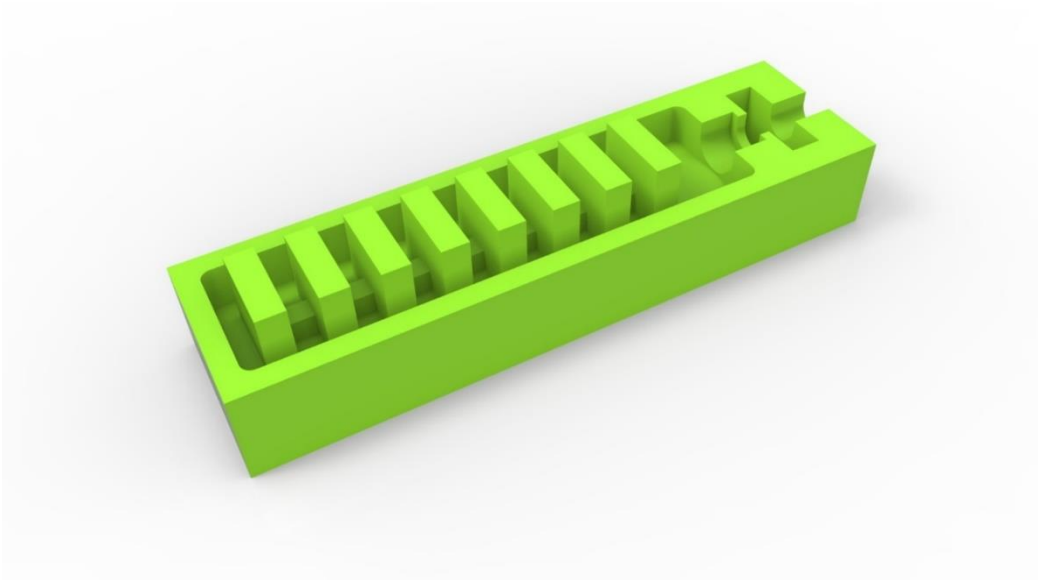

Figure 3-6 Solution 3, bottom part

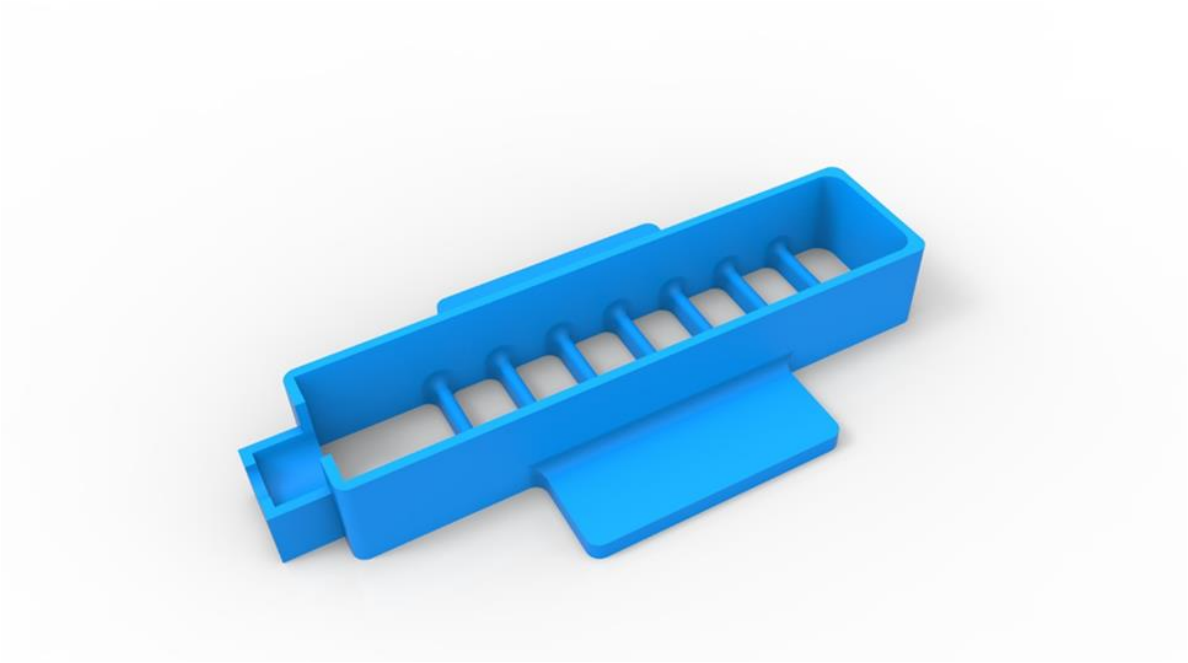

Figure 3-7 Solution 3, upper part

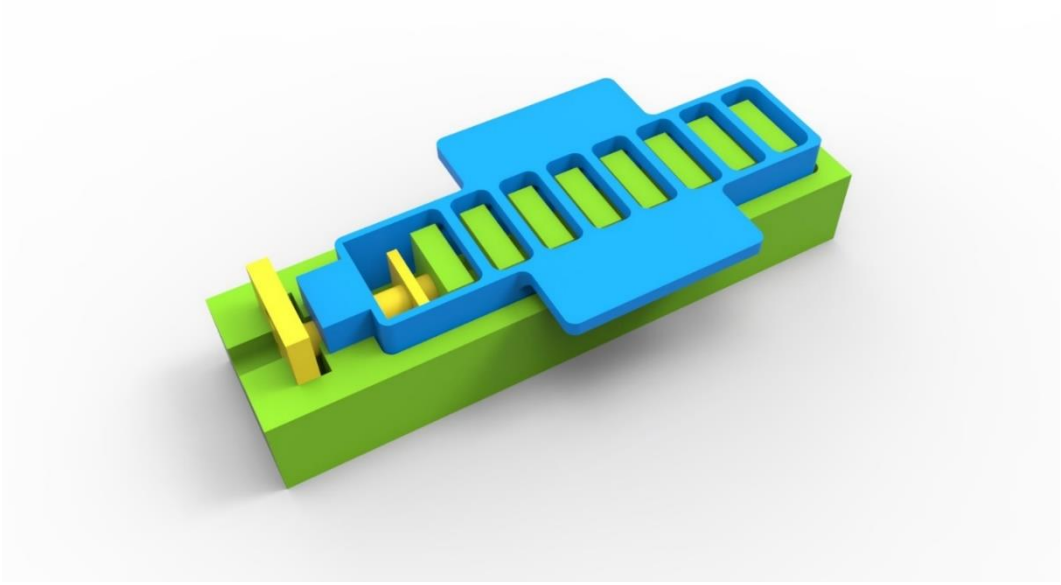

Figure 3-8 Solution 3

Solution 4 (*figure* 3-9) expects that the clamping system is no longer present preventing the air to leak out. On one side of both the molds, a cylindrical shape with a different diameter is introduced to create a silicone part able to keep directly in position the tube in which the pressurized air flows. For this purpose, an additional component representing the negative of the pneumatic tube is added. With this design the tube can follow the deformation of the soft robot, having a stiffness comparable to the one of the silicone

rubber. To make the system more safe a cable tie can be used. As it was expected, this solution works, solving the persistent problem of the previous designs. Since the clamping system is no longer present, the problem now is to think about something that can replace it. The thought design expects two different parts coupled to each other by means of screws, able to “entrap” the first part of the robot, reproducing the boundary condition used in the FE analysis. This system has also two holes so that the robot can be placed anywhere using two screws (*figure 3-11*).

For what concern the bottom part, a simple mold reproducing the overall shape is requested. The crucial dimension of this mold is the height. It must be high enough to guarantee two times the injection of the silicone rubber, as it will be explained later, but not too much to avoid the obstruction of the communicating channel. For this reason, its design influences the helpers dimensions. To increase the grip action of the soft finger, additional hollows with different shapes can be added to create a pattern. The idea is to recreate human fingerprints, making the robot as similar as possible to a human finger. The simplest solution is to reproduce small parallelepipeds, but other complex geometry can be thought (*figure 3-10*).

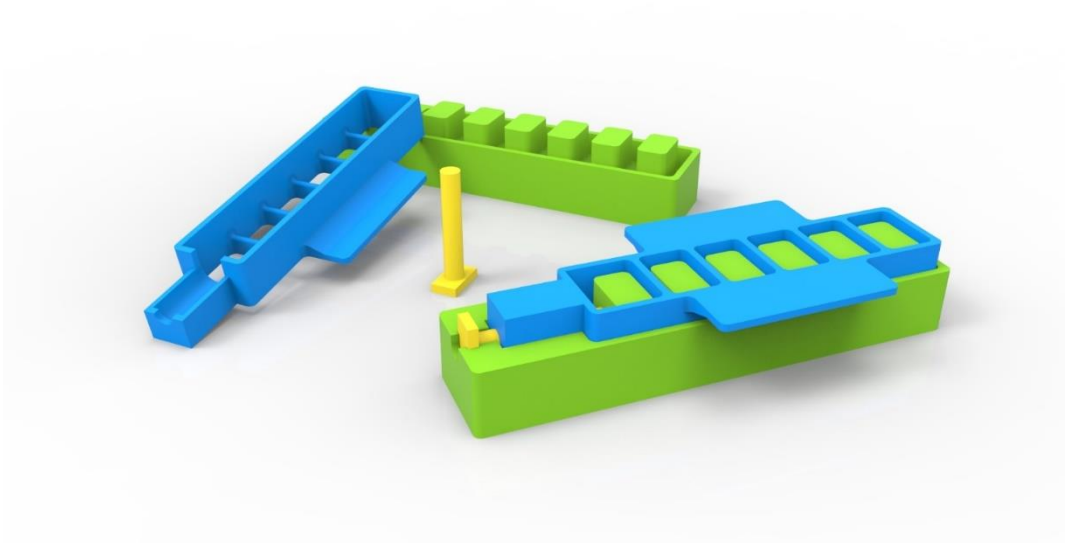

Figure 3-9 Solution 4

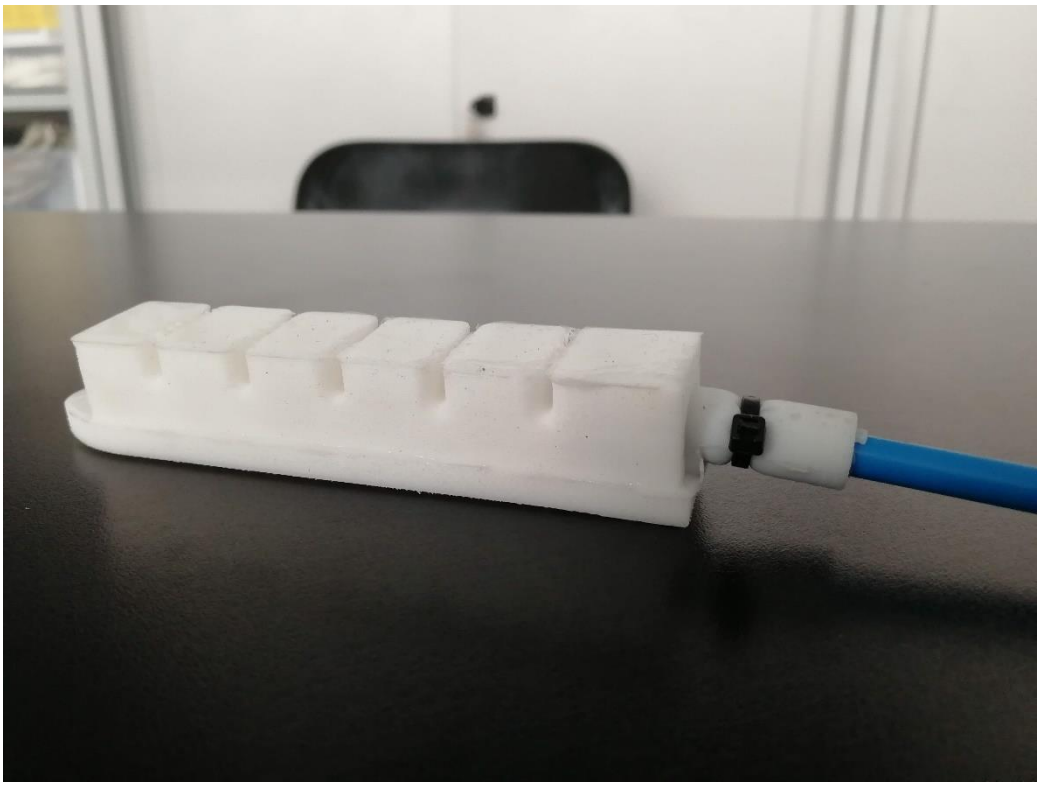

Figure 3-10 Solution 4, result

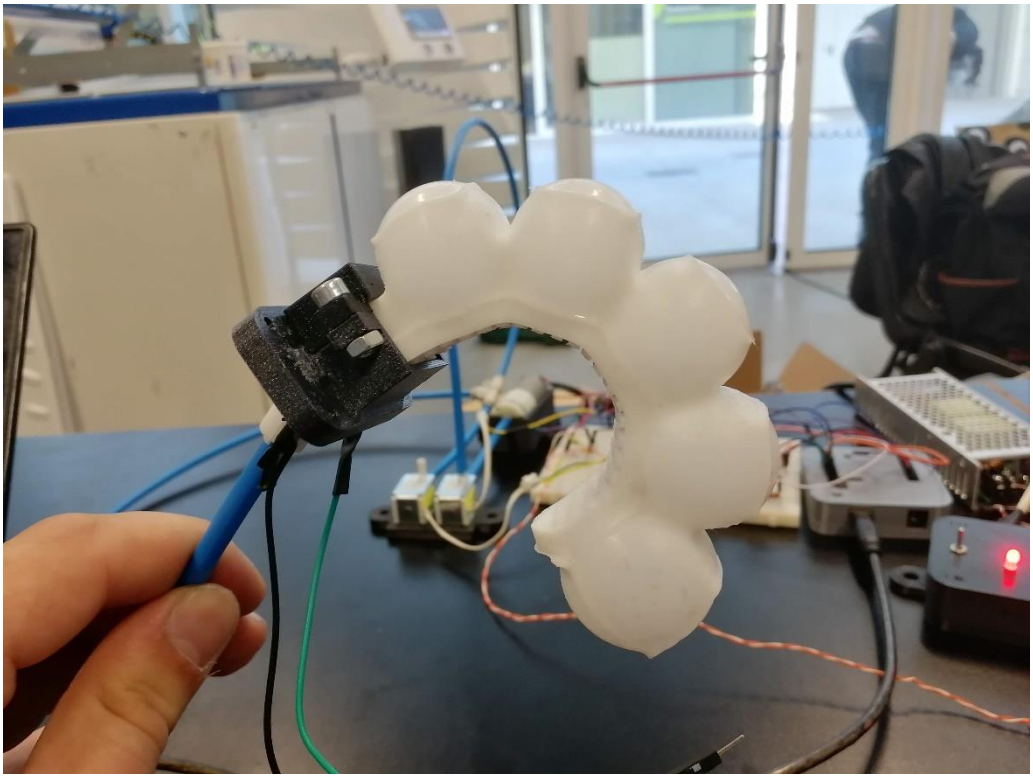

Figure 3-11 Solution 4, clamping system

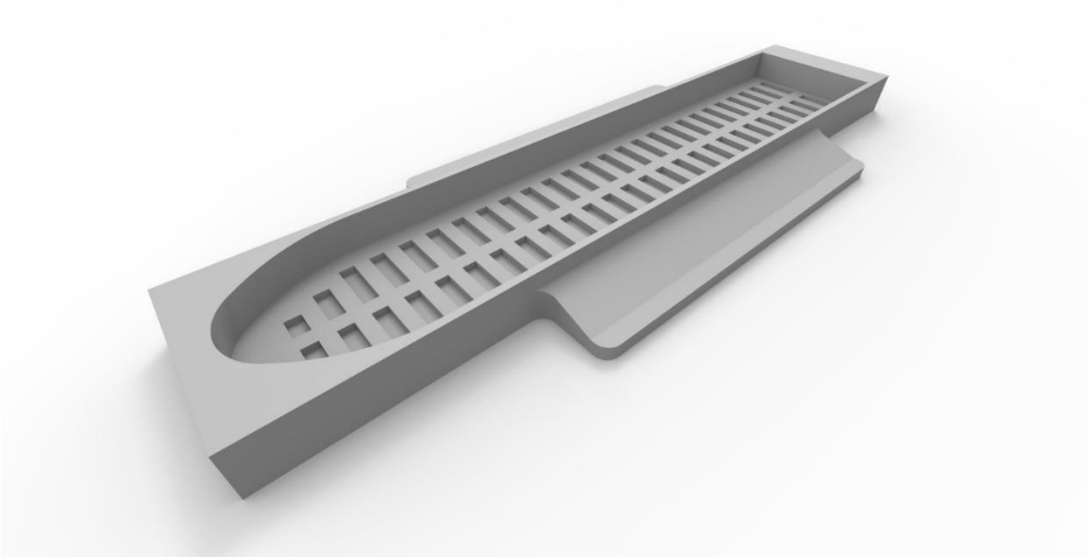

Figure 3-12 Second mold

Figure 3-13 and *figure 3-14* show all the described solutions.

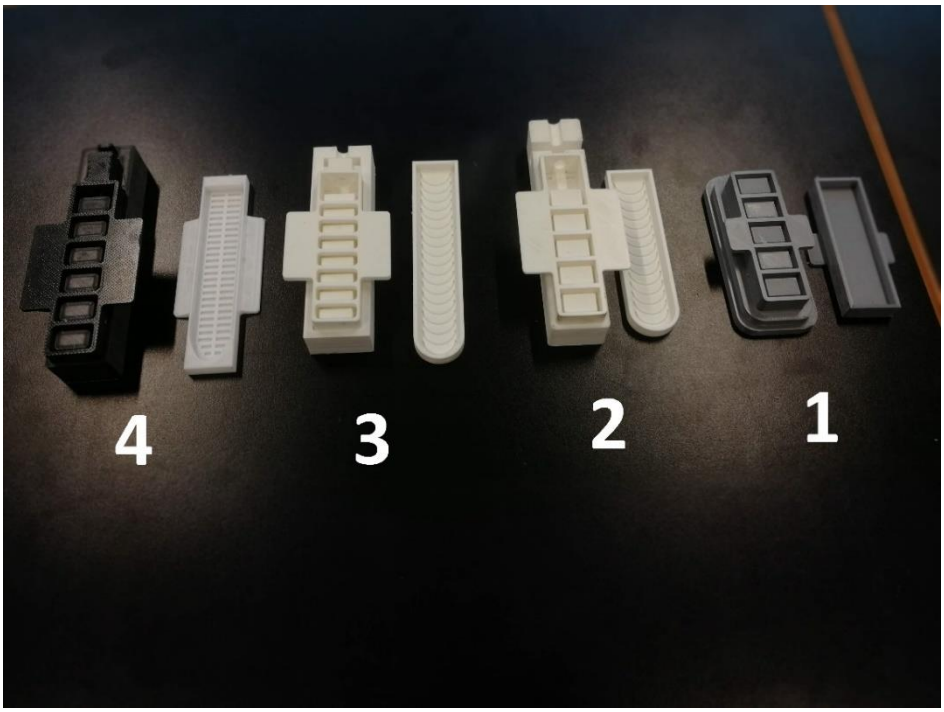

Figure 3-13 Proposed solutions

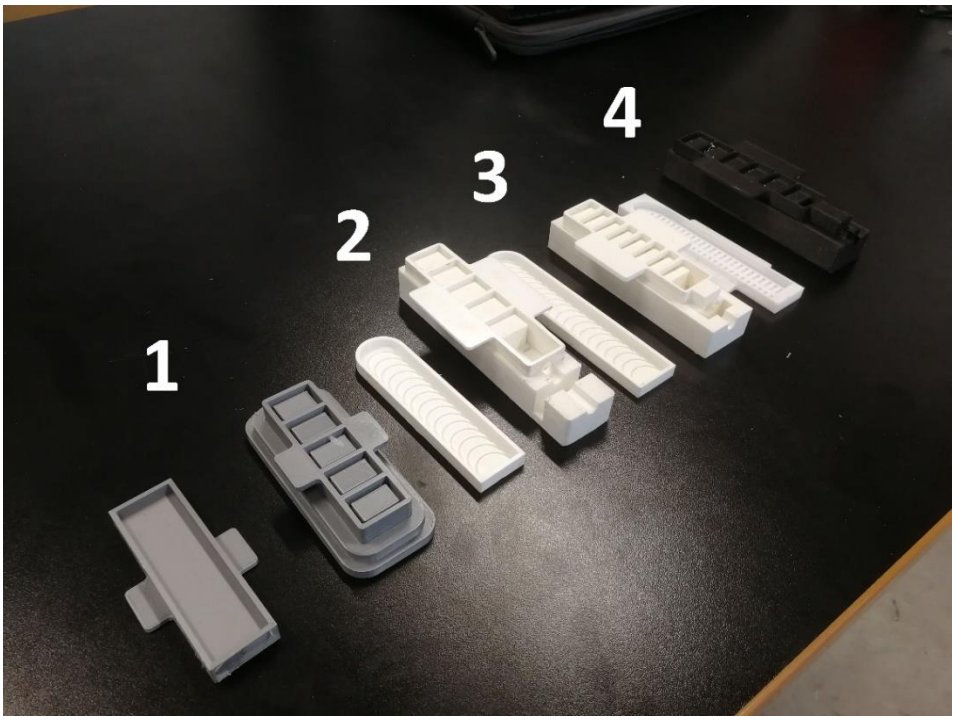

Figure 3-14 Proposed solutions

PRODUCTION METHOD

After producing the molds, the silicone rubber must be injected. As shown above, the molds of the upper part need to be coupled together with the negative of the pneumatic tube. The silicone rubber must be prepared previously mixing the silicone with a catalyst, respecting the specific proportion of 1 g of catalyst every 20 g of silicone. Then the mixture of silicone and catalyst must be mixed for at least 10 minutes. Once the molds are assembled the rubber is injected until it fills completely the molds system. For the bottom part, the production is divided into two stages. As explained before, for the robot to move a stiffer layer is mandatory, for instance a piece of paper. For this reason, the corresponding mold must be half filled allowing to put the paper layer on the top (*figure 3-16*).

Figure 3-15 Silicone rubber and catalyst

Once the silicone has been injected, it takes some hours to cure completely. Then the solidified silicone of the upper part can be extracted from the molds (*figure 3-17*). To conclude the production, some liquid rubber must me put inside the lower part of the mold until the mold itself is completely full, covering the already cured part. Then the upper part can be placed on it, using the helpers to prevent the liquid rubber to obstruct the connection channels (*figure 3-18*). When the silicone is completely cured, the soft finger is ready.

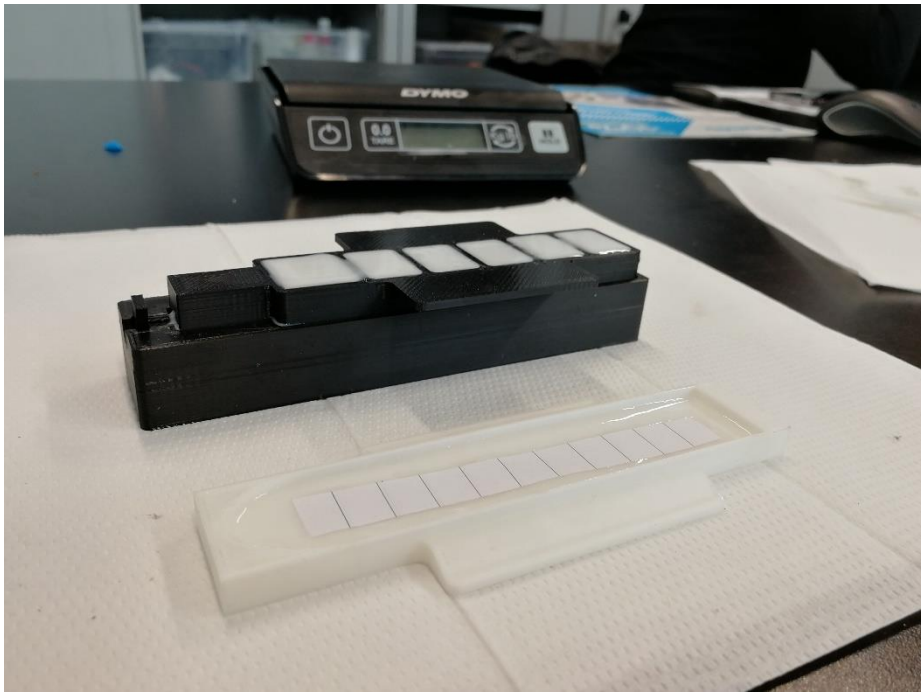

Figure 3-16 Silicone rubber injected inside the molds

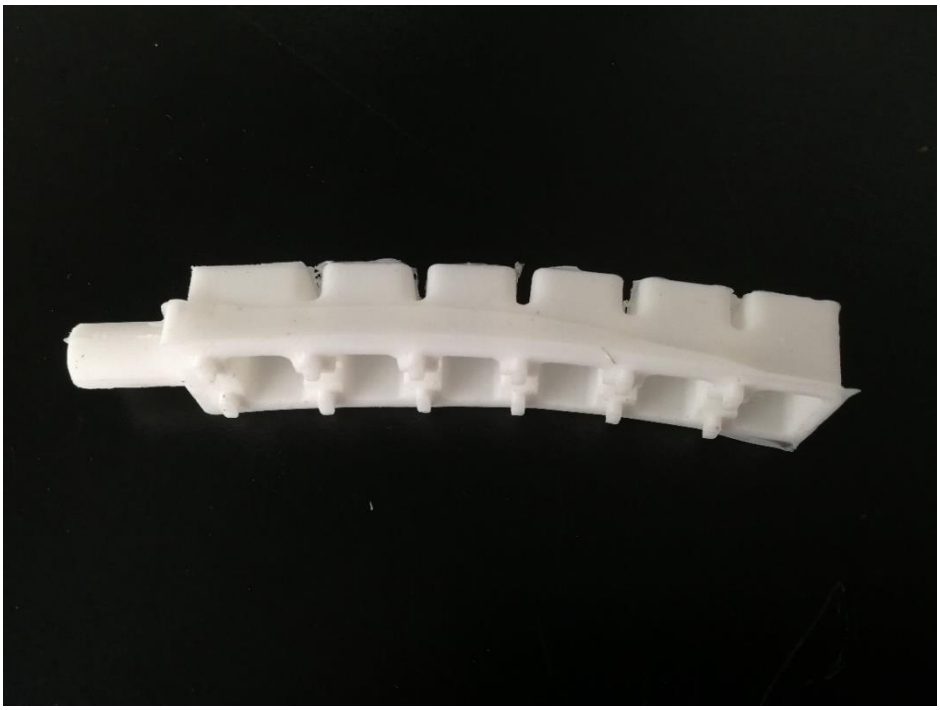

Figure 3-17 Soft robot's upper part

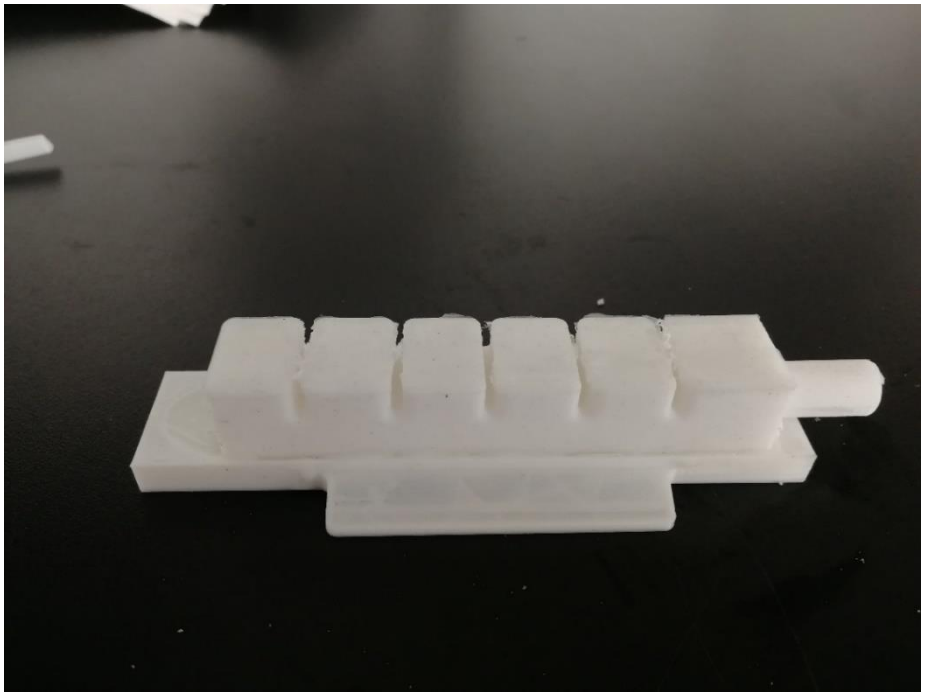

Figure 3-18 Last production step

To check that the soft robot is fully functional it’s requested to put a tube inside the hole and use a syringe to inflate the air. To prevent the air from coming out some cable ties can be useful to tighten the silicone part around the tube.

It can be seen that the soft robot respects the expected motion, previously analyzed with FEM technique.

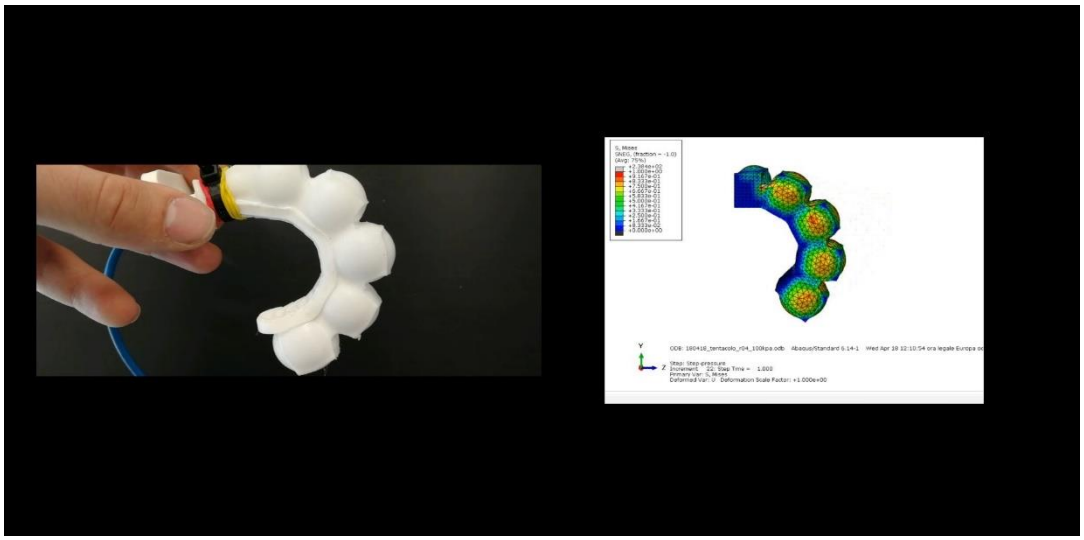

Figure 3-19 Comparison between FE model and reality

MOTION CONTROL WITH ARDUINO

The syringe is useful to check if the communication channels are not obstructed making the soft robot fully working but it's quite uncomfortable for automatized applications.
For this reason, to control the motion of the soft finger an electrical circuit controlled by Arduino can be used. In this way, with suitable scripts, different applications can be executed.

Electrical circuit design

The designed electrical circuit is thought to inflate the soft finger thanks to the actuation of a single 6 V air pump. To keep the air inside the silicone body two solenoid valve can be used. One of them is responsible for the intake phase, the other one for the air deflation. For other applications involving more than one soft robot, multiple pumps and solenoid valves can be used modifying the proposed solution.
To guarantee the integration of the mechanical parts and to avoid electrical issue some specific electrical components are mandatory. After some research on the web, some components have been selected.
The functions of each of the circuit components will be explained in the appendix.

Embedded sensors

Sensors can be useful to monitor the behavior of the soft robot for further applications. Working with rubber silicone makes it possible to directly integrate them inside the soft finger.

Many sensors work by converting the energy they read into a changing electrical resistance by using a variable resistive material at their heart. For example, force sensors and stretch sensors are made of a partially conductive rubber. When the rubber is stretched or compressed, the resistance changes. Flex sensors are resistors that change their resistance when they are bent. To read changes in resistance, these sensors are typically placed in a voltage divider circuit, which converts the resistance change into a changing voltage. Flex sensors are particularly suited for the studied application, making possible the measurement of the bending motion of the robot. These measurements will be useful later when machine learning techniques will be applied on the finger. On the market there is only one model of this sensor, sold with different lengths. For this application, the variant with a length similar to that of the robot is chosen.

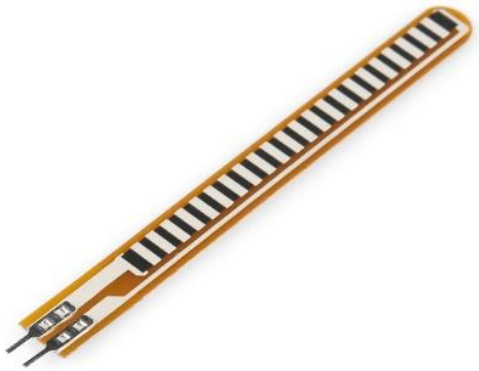

Figure 3-20 Flex sensor

Following this principle, flex sensors can be embedded in the silicone, placing the resistor on the lower part of the robot. In this way, when the robot moves it will bend the sensor making the measurements feasible. As the sensor is integrated inside the finger, the Flex sensor itself could become the stiffer layer instead of the piece of paper.

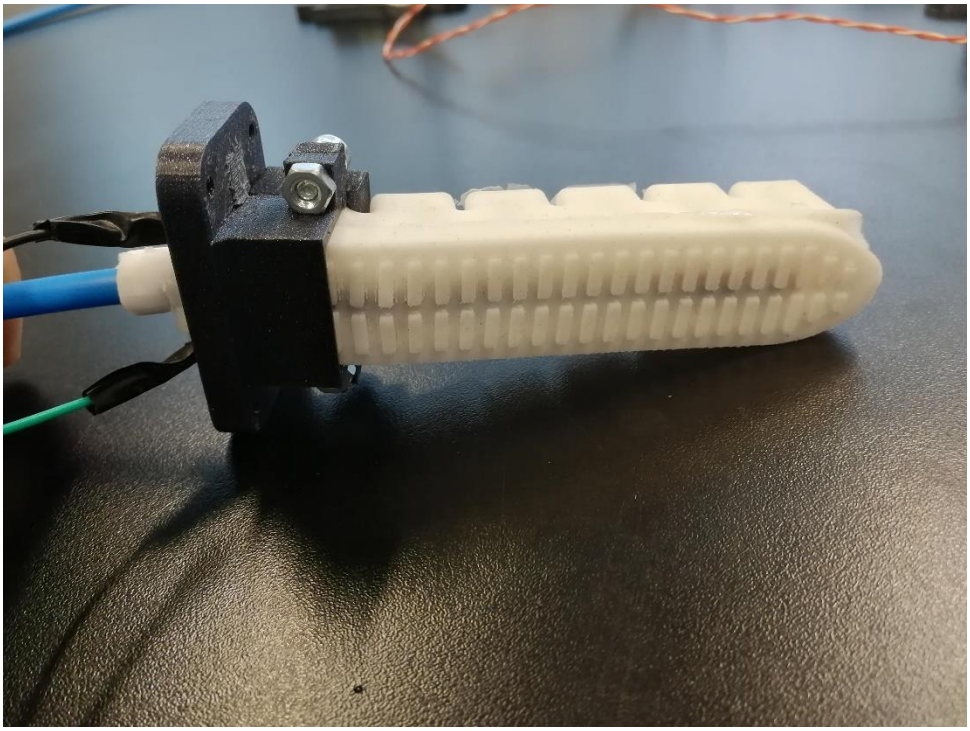

Figure 3-21 Flex sensor embedded inside the robot

Circuit wiring

Once the sensor is integrated it is possible to build the entire circuit. The schematic circuit arrangement for the actuation is shown below. The following pictures are done with a plug in of Arduino called *"Fritzing"*, that allows to obtain the scheme of an electrical circuit (*figure 3-22*) and a visual representation of the connection of the components with the Arduino board (*figure 3-23*).

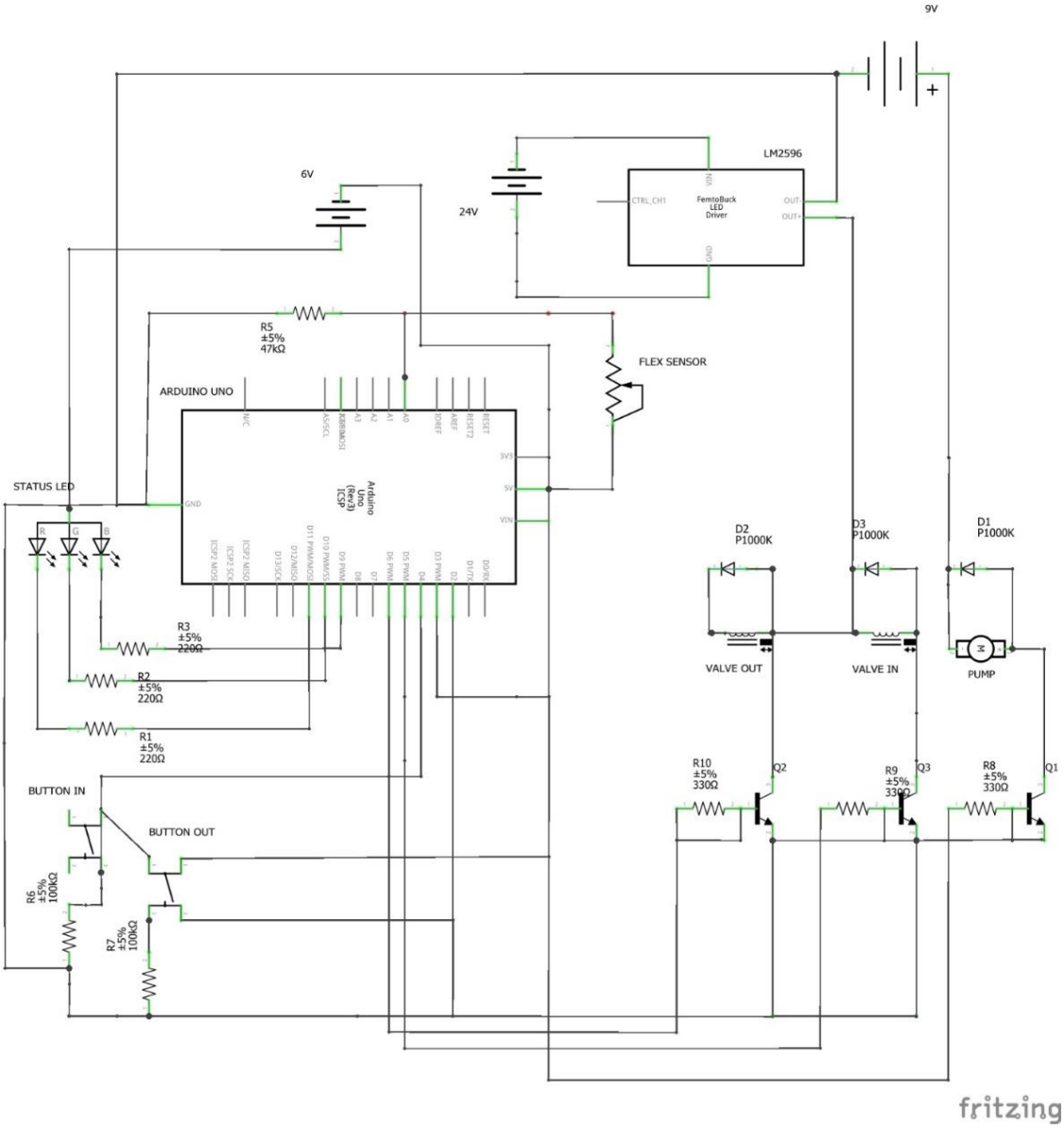

Figure 3-22 Circuit diagram

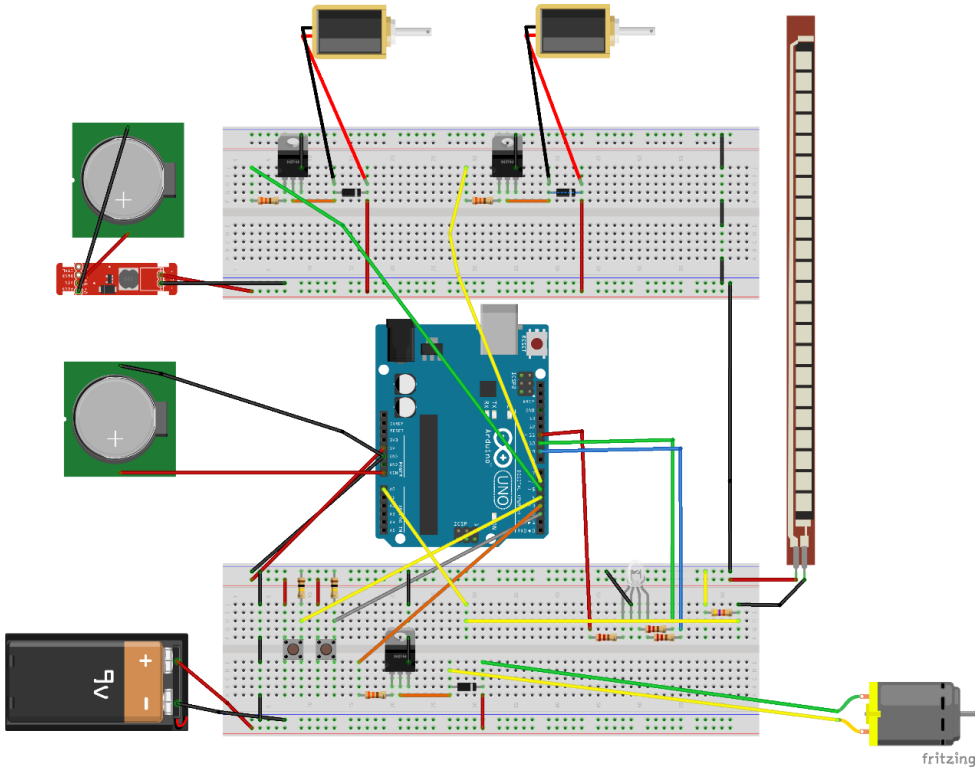

Figure 3-23 Arduino's connections scheme

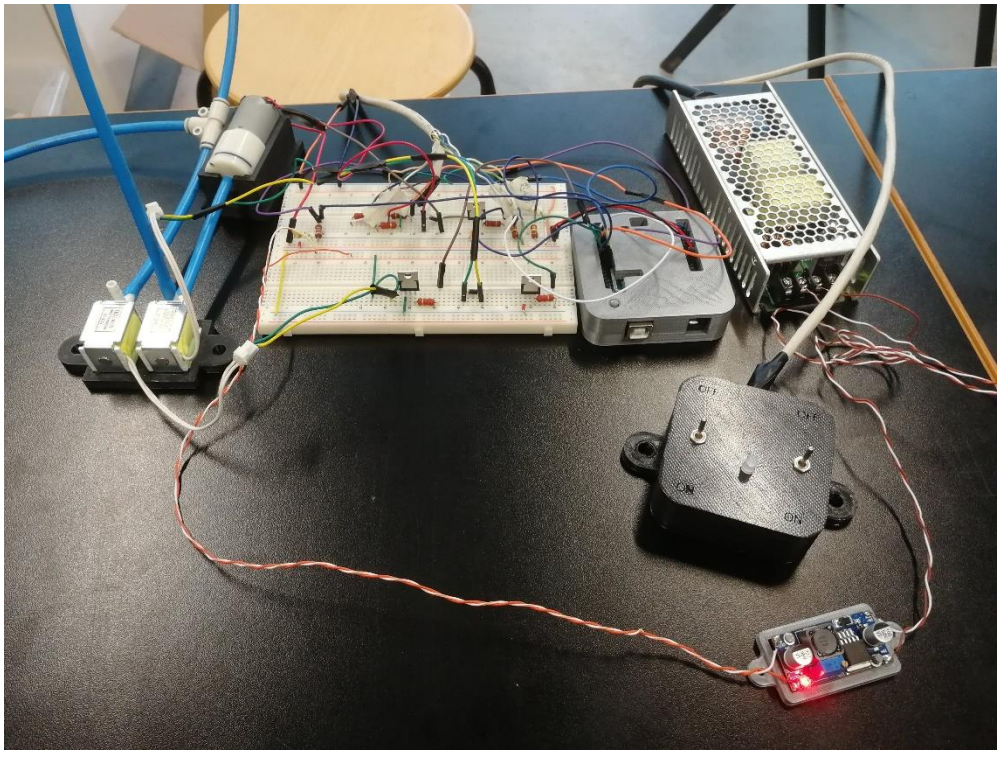

Figure 3-24 Realized circuit

Arduino's sketch

To make the robot to move the electrical circuit is not enough. It is necessary to write a software able to actuate the different components of the circuit. The code is written in the Arduino's development environment called IDE (Integrated Development Environment), where the codes are called "sketches". To collect the data from the sensors and show them in real time an open source program, called "*Telemetry Viewer*", is used. It allows to monitor the data in real time and to eventually save them in CSV files, useful for further analysis. In this way with a single code it is possible to actuate the soft finger and collect the data from the sensors.
The developed code is made by three different sections, that represent three different scenarios that can occur during the use of the soft robot. The code will be briefly described in the following lines.

The first part of the code expects the assignment of the different pins of the microcontroller to a specific variable used in the following line of the sketch.

Generally, an Arduino's sketch is always made by two big functions. The first is the "*setup*" function and the second one is called "*loop*" function. As the name suggests the "*setup*" function is invoked only once at the beginning of the program and it is used for the initial setting that will not change during the execution. For each pin it is decided if the pin itself woks as an output pin or as an input pin, that is a crucial part of the program. In this section it is also open the "*serial port*" with the command *"Serial.begin(9600)"* that allows to communicate with the Arduino boards by means of the USB port, making possible to print values or to send command by the serial monitor while the code is running.
On the other side the "*loop*" function is invoked repeatedly until Arduino is powered by electricity, executing every time all the commands inside itself. As said before, in the "*loop*" function three different scenarios are considered. Remember that the circuit have two different switches, one for the activation of the pump and the inlet valve, and the other one to control the outlet valve. Let's call the first "*intake switch*" and the second one "*outlet switch*". There is also a LED whose color is commanded by a dedicated function called "*RGB*". The three different scenarios that can occur are:

1. Intake switch OFF, outlet switch OFF: in this scenario no power is supplied to the pump and both the solenoid valves are closed. If the robot is inflated, the air cannot escape. The color of the status LED is set as red.
2. Intake switch ON, outlet switch OFF: in this scenario the power supply feeds the pump and the intake valve making the robot to move. The outlet valve is closed and the color of the status LED is set as green. Being that the robot moves, in this code section is also managed the acquisition of the data from the sensor. This data can be read directly from the serial monitor or can be transferred to the telemetry software for a better real time visualization.
3. Intake switch OFF, outlet switch ON: in this section no power is given to the pump and to the inlet valve, while the outlet valve is open. In this way the air inside the robot can go out allowing the soft finger to return to its original state. The color of the status LED is set as violet.

If it is considered necessary, it is possible to read the sensors value in all the three scenarios by adding few lines to the code as shown. This allows to have a better knowledge of what is happening to the robot. As said before, "*Telemetry viewer*" software allows to show the acquired data in real time. The data can be visualized in different ways, depending on their nature. In this specific case it's better to use a conventional plot, where the sensor value is put on the y-axis and the time is on the x-axis. Unfortunately, for what concern the data saving, this software is not very effective and intuitive. For this reason, in the next chapter, a new method to manage the acquisition must be taught.

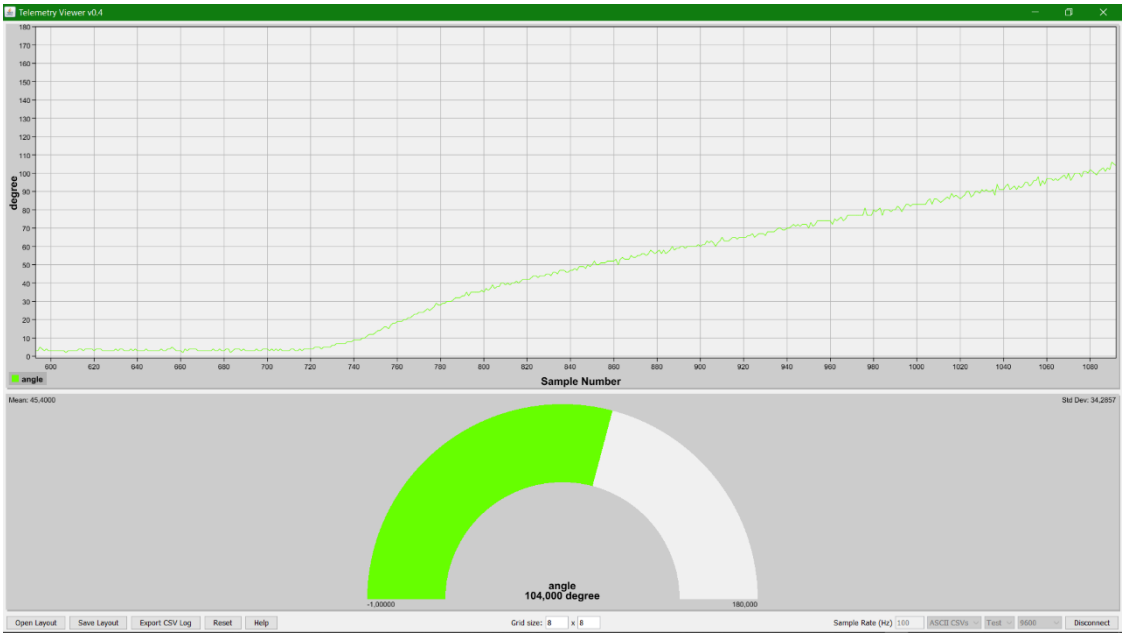

Figure 3-24 "Telemetry viewer"

4. MACHINE LEARNING

Once the smart soft robot has been realized and the electrical circuit is ready to move the robot itself, it is possible to proceed with a simple machine learning process. In this work, considering the use of the flex sensor, the aim of the machine learning is "to teach" to the soft robot if a touched object is soft or hard. Thinking about future application, this can be useful for smart pick and placed operation or to adjust the grip of an object depending on its own feature. If other sensors were used, different classifications could be made. For instance, with a temperature sensor it could be possible to distinguish between warm objects and cold objects.

In the studied case is possible to detect if an object is hard or soft by looking to the relative deflection occurring to the robot after the contact with a specific object. It is important that the acquisitions start only when the soft robot came in contact with the object to avoid errors due to the dimension of the touched samples. If the absolute deflection is considered, it could happen that a small hard object is treated as a soft object, and a larger soft one is detected as hard. To overcome this problem the simplest solution is to put the robot itself directly in contact with the object before the inflation of the silicone rubber. A more complex solution is to use some force sensors to detect when the soft finger touches something. Generally, this kind of sensors are piezoresistive sensors that change their resistance on the base of the force acting on them. The integration of an additional sensor on the soft robot might result complicated because the majority of the sensors on the market guarantee a small contact area, making them not suited because it is difficult to predict where the soft robot touches an object. In addition, these sensors are often very stiff, making them not feasible with the bending nature of this technology.

Acquired data are only the first step to realize a machine learning process. A specific algorithm is mandatory to manage the acquired data and make them ready for machine learning techniques. Then a code able to create a trained model on the base of some machine learning techniques using acquired data is also requested. As it will be explained later, for this purpose a MATLAB tool comes to help, making this part simpler. Finally, a code that is able to understand the nature of an unknown touched object is needed. This prediction takes advantage of an acquisition and use the previously trained model. In this thesis work, the prediction is not in real time because there's no direct connection between the acquisition system and the MATLAB environment. The unknown data must be imported later in MATLAB after the acquisition to be analyzed.

In the following sections it will be explained the code realized for this purpose, the setup used during the experimental tests and the obtained results.

MATLAB CODE

A MATLAB code is necessary in order to apply effectively a machine learning process to the studied case. To make everything simpler and more intuitive, a single code has been developed divided into several sections. Each of these sections represents a fundamental step, starting from the organization of data, passing through the training of a model up to the possibility of predictions based on that same model.

From the main menu it is possible to choose which part of the code to execute. In the following lines all the features of this program will be described, highlighting the purpose of the different parts of the code, without going into the details regarding the constructs and the functions used.

According to the operation to be performed, it is possible to choose between the following sections:

- **Data processing:** from this section it is possible to choose between two codes very similar to each other for what concern< the functions and the constructs used but with a different purpose.
  "Import train files" allows to work on acquired data that will be used to train a model. To execute these lines of code the acquired data and a file containing the labels are mandatory. Being this type of training a supervised one, the labels file represents a crucial part for machine learning because it creates a linking between the data and the feature of the acquisition. In this specific case the labels representing the characteristic of the object are "*hard*" and "*soft*". The code is developed in such a way that it is possible to select the file from a folder, specifying the extension of the file (.csv, .txt, …).
  At the end of this section it is possible to save the reworked data in a new "*.mat*" file. In the studied case the acquisitions come from a single sensor, but the developed code works also using *n* sensors.
  "Import test files" code is analog to the previously described one but works with unknown acquisition of objects of which the features have to be predicted. A label file useful to compare the predict characteristics with the real ones is also requested. The functions and constructs used are the same as well as the structure of the output file.
- **Training**: the training section is the core of the program. The output file coming from the *"Import train files"* section is mandatory to train a predictive model.
  Considering what has been said in the "*State of art*" section, machine learning algorithms depend on mathematics. In particular, some statistical quantities are requested in order to build a model. In this code, three statistical quantities are considered:
  1. Mean value: for each acquisition of each sensor the mean value is calculated
  2. Standard deviation: for each acquisition of each sensor the standard deviation is calculated

3. Principal component analysis (PCA): Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. If there are *n* observations with *p* variables, then the number of distinct principal components is *min(n-1,p)*. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, it accounts as much of the variability in the data), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. PCA is mostly used as a tool in exploratory data analysis and for making predictive models. It is often used to visualize genetic distance and relatedness between populations. For each acquisition of each sensor the PCA is calculated.

To evaluate this quantities predefined MATLAB function can be used, in particular for the PCA that it's more complicated compared to the other values.
Once the calculation of the statistical quantities is over, the code automatically runs the MATLAB tool called *Classification Learner*. This tool allows to use sets of data from the workspace to train a model. Choosing the data containing the values calculated previously, the statistical quantities are considered as *predictors* of the training while the labels are imported as *response* of the training. In this first step of the tool it is also possible to decide if a percentage of the training data can be used to validate the model or not. The validation function is strongly recommended, especially if a large set of data is available, to obtain a model as much precise as possible.
In the second step of the *Classification Learner* tool it's actually possible to train a prediction model. Here it is possible to apply algorithms, some of which have been explained in the "*State of art*" chapter. Very often some of these algorithms have multiple versions, making the amount of technique available very extensive. The tool gives the chance to create different models based on different algorithms, up to the possibility to build a model for each of them.

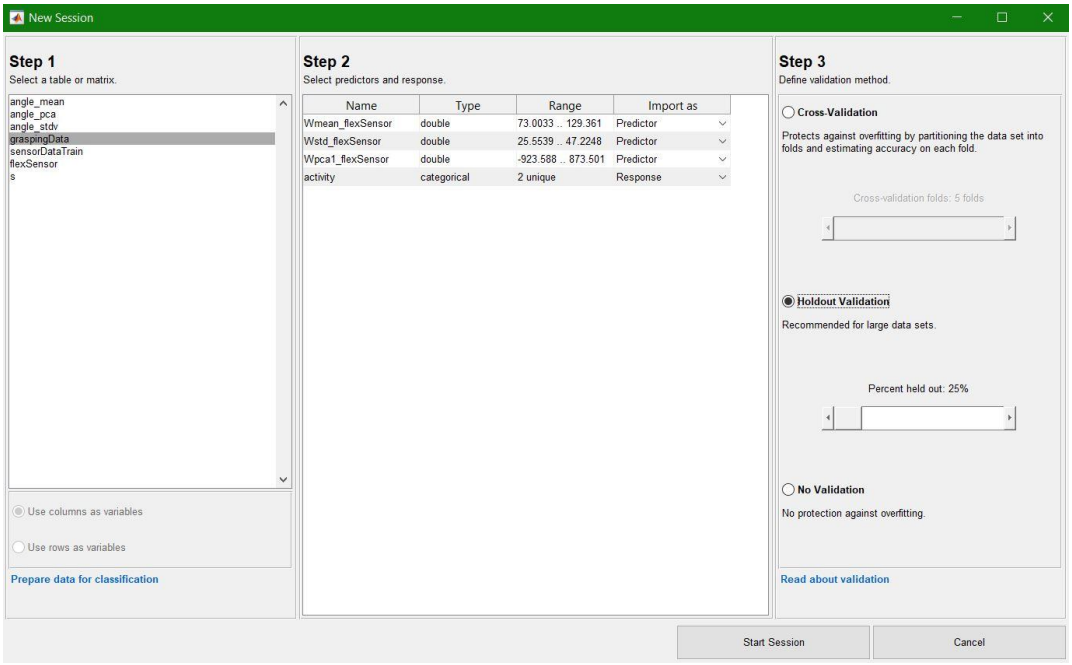

Figure 4-1 Classification Learner tool

The *Classification Learner* provides three tools to analyze the results of a single model and to compare it with another one. To show these tools two models with a 88.9% and 44,4% accuracy are used.

1. *Scatter plot*: a **scatter plot** is a two-dimensional data visualization that uses dots to represent the values obtained for two different variables - one plotted along the x-axis and the other one plotted along the y-axis. In this specific case the statistical quantities (predictors) are plotted on the two Cartesian axes. From this plot it is possible to see when the model has a good behavior and when it gives a wrong response.

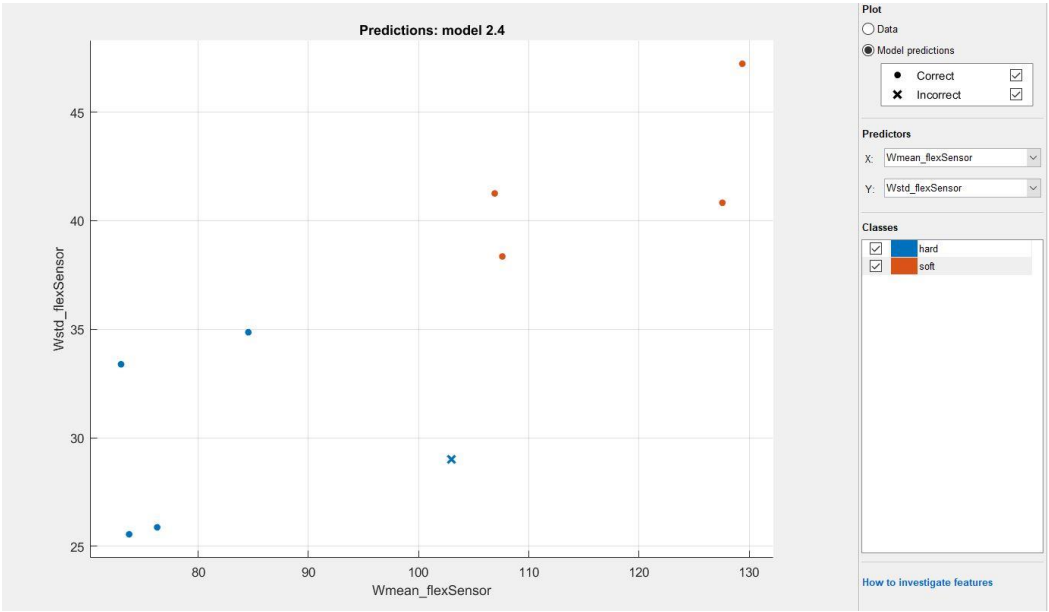

Figure 4-2 Scatter plot (88.9%)

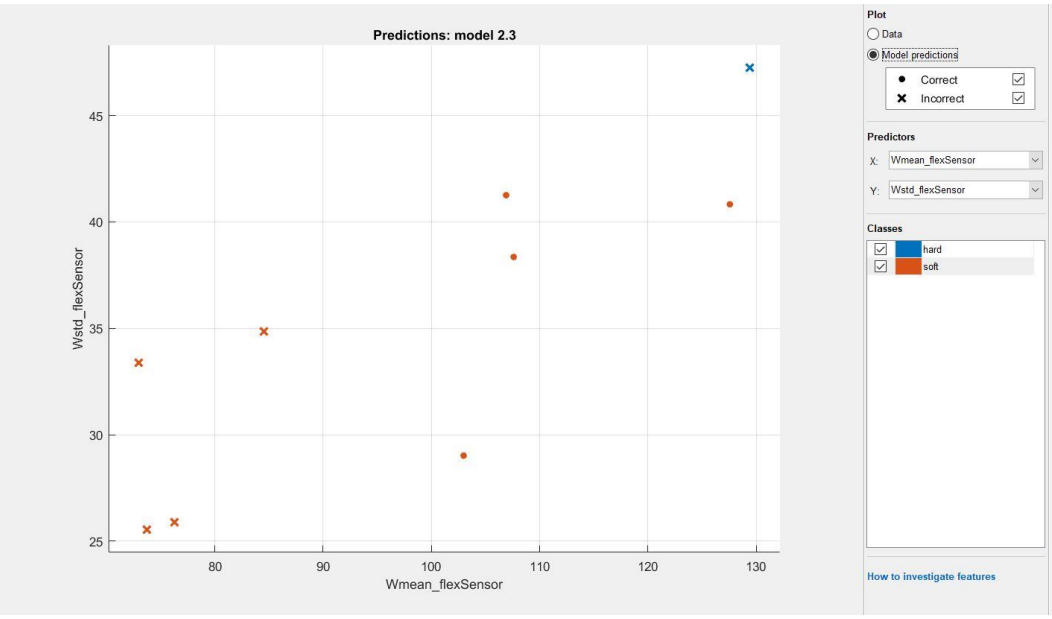

Figure 4-3 Scatter plot (44,4%)

2. *Confusion matrix*: a confusion matrix is a technique for summarizing the performance of a classification algorithm. Calculating a confusion matrix can give a better idea of what the classification model is getting right and what types of errors it is making.

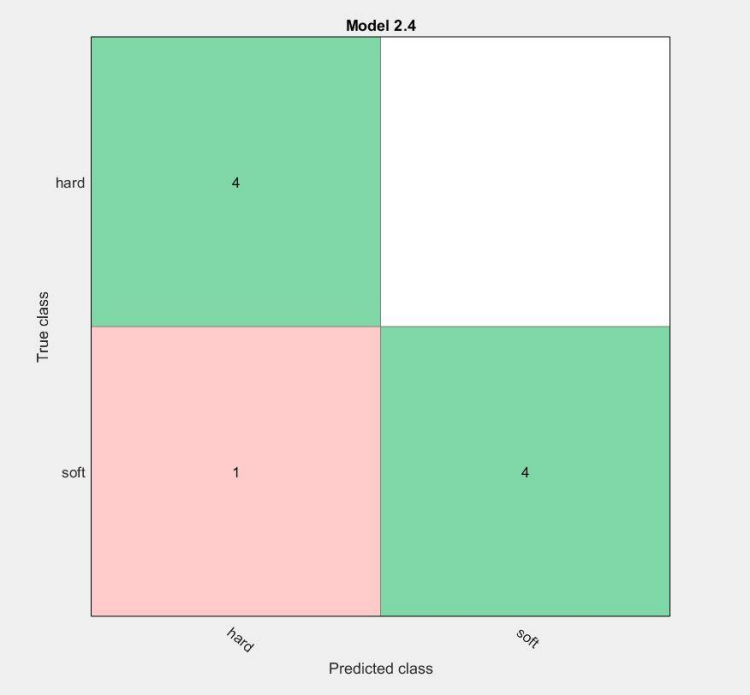

Figure 4-4 Confusion matrix (88,9%)

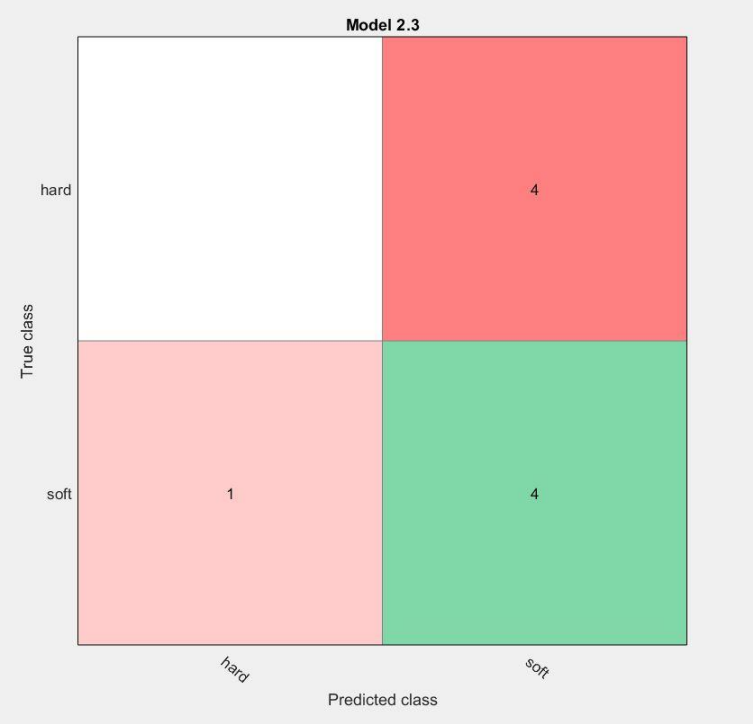

Figure 4-5 Confusion matrix (44,4%)

3. *Receiver operating characteristic curve (ROC curve)*: The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) for the currently selected classifier. For example, a false positive rate (FPR) of 0.2 indicates that the current classifier assigns 20% of the observations incorrectly to the positive class. A true positive rate of 0.9 indicates that the current classifier assigns 90% of the observations correctly to the positive class. ROC analysis provides tools to select possibly optimal models. Accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test; an area of 0.5 represents a worthless test.

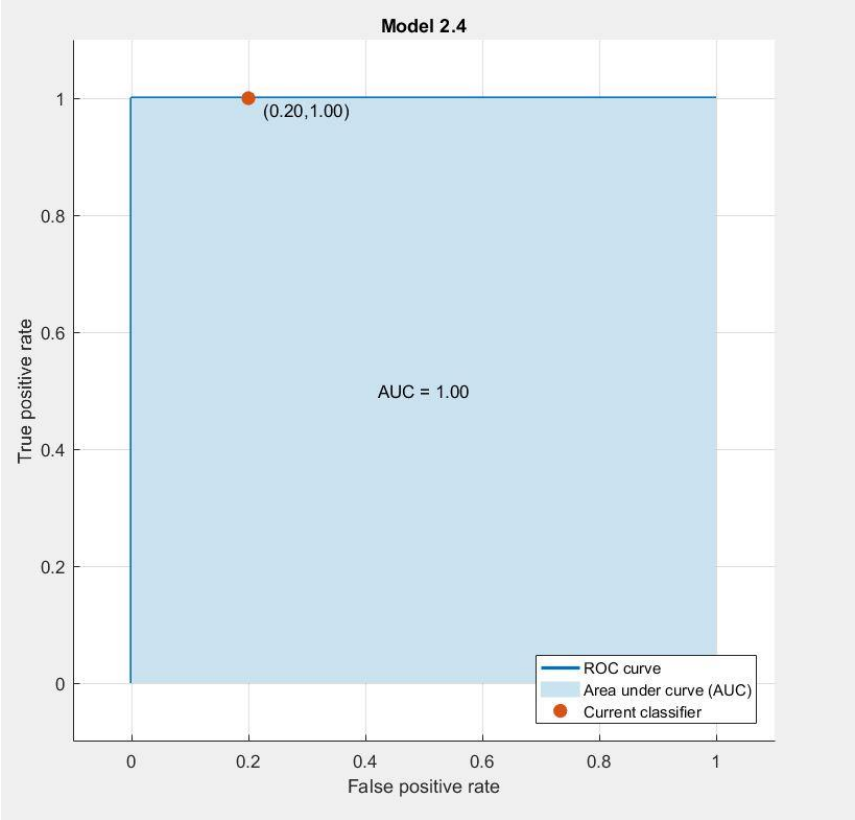

Figure 4-6 ROC curve (88.9%)

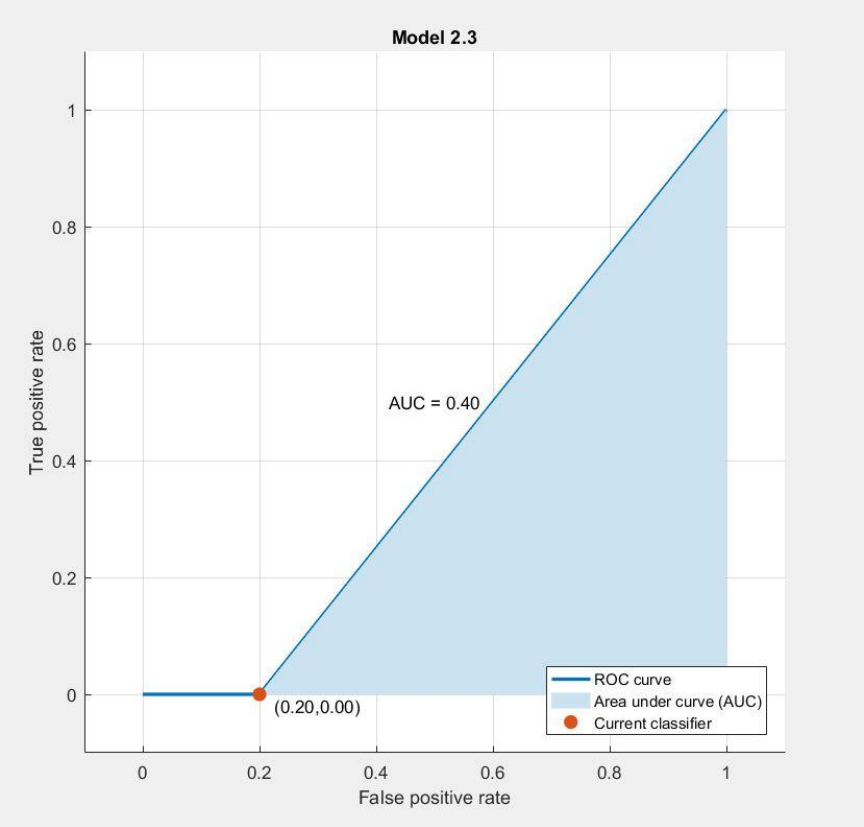

Figure 4-7 ROC curve (44.4%)

Once the most suitable and accurate model is chosen, it is possible to export it in the workspace and save it in a ".*mat"* file in order to be used for future predictions.

- **Prediction**: the aim of this last part of the main program is to use the trained model to make forecasts on unknown set of data. The unknown data set used in this operation is from the *"Import test files"* section. This structure of data could contain any number of acquisitions, provided that the number of sensors and the same data structure coincide with those used for model training. To test the model and to have results on the accuracy, the data are accompanied by a .txt file containing the labels representing the true nature for each acquisition. As for the "train data", also in this part of the code the same statistical quantities calculated in the train section are requested. By means of a MATLAB function called "*predict*" it is possible to compare for each acquisition the statistical quantities of the unknown data to the ones of the training data on the base of the trained model. The code is then structured to show the prediction results and the model reliability in a simple and clear way. For each acquisition the prediction result is compared with the true nature label of the same acquisition. A histogram and a table are used to show the results through the help of colors.

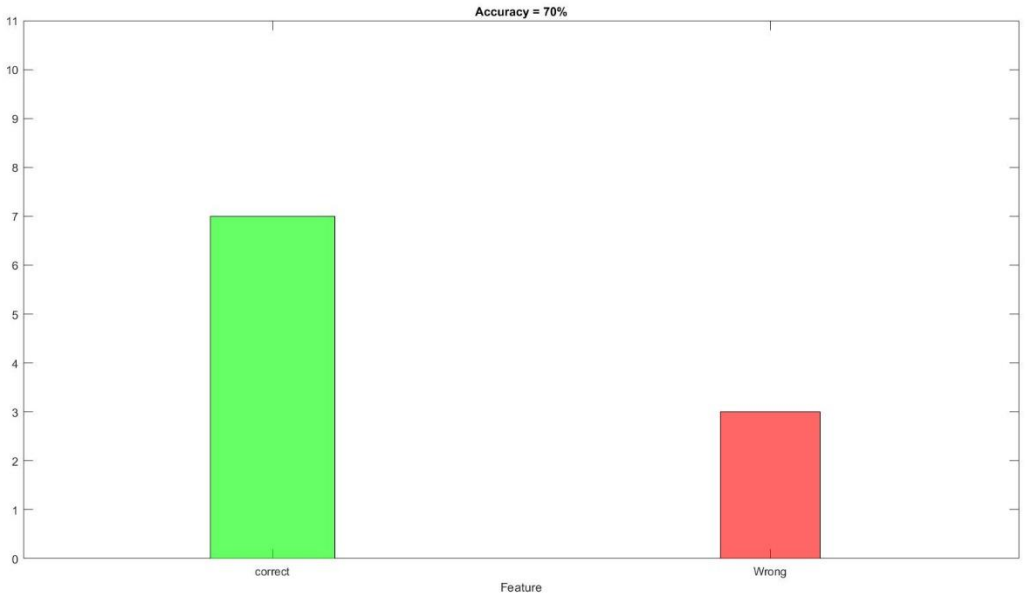

Figure 4-9 Results histogram

	PredictedFeature	RealFeature	Results
1	soft	soft	Correct
2	soft	soft	Correct
3	soft	hard	Wrong
4	hard	hard	Correct
5	soft	soft	Correct
6	soft	soft	Correct
7	soft	hard	Wrong
8	soft	soft	Correct
9	hard	hard	Correct
10	soft	hard	Wrong

Figure 4-10 Results table

This section ends with the possibility to save the results in a *.mat* file or to export them into an Excel file or other format for further analysis.

EXPERIMENTAL TESTS

As previously said, machine learning requires a large amount of data to work effectively. Larger is the database available, more accurate will be the trained model. The most important companies in the world that work with machine learning applications, such as Google or Amazon, have continuously updated databases managed by large servers. In this work it's impossible to reach such a large amount of data, first of all for the available time and second for the computational power. Nevertheless, an acquisition method and an acquisition set up are thought to make this process as effective as possible.

Software

Firstly, it is necessary to make some changes in the Arduino's sketch. The original one expects to acquire the data by means of the open source program "*Telemetry Viewer*". As it was shown, this software is very efficient in showing the acquired data in real time, but at the same time is not very effective in saving them. Thinking to a process in which a large number of acquisitions are requested, this solution is not the best one. Secondly, as said in the "MATLAB code" section, a set of acquisition of the same length is mandatory for the code to work properly. The original Arduino's sketch doesn't provide the possibility to collect a fixed number of samples or to acquire data for a specified time. What is needed is a code able to acquire data in a controlled way and save each acquisition in a dedicated text file. In the following line the solution for these two problems will be shown.

First of all, to make file saving faster and more intuitive it is possible to use the "*Micro SD module*" for Arduino. This component allows to save the acquired data in a file directly on a Micro SD card, without the help of a secondary software.

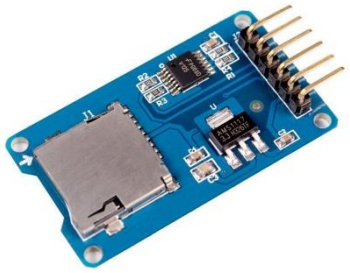

Figure 4-11 Mirco SD module for Arduino

To integrate this component it is necessary to upgrade the sketch with some lines that enable the communication between the Arduino board and the component itself. To do this in the first part of the code is mandatory to include two Arduino's libraries able to manage the Micro SD card module.

To make sure that the acquisitions are all of the same length it is possible to add a *for* cycle to manage the acquisitions themselves. Once the number of samples is chosen, the

statements declared in the cycle will be repeated for that number of times. To make it clear to the user that the acquisition is finished, some lines are introduced in the code which will cause the status LED to flash for a few seconds. Finally, the code is modified allowing to decide the file name of each acquisition by writing it directly in the serial monitor.
Adding few lines to the code it is possible to calculate the execution time of each acquisition cycle, thus making the sampling frequency known. In particular the cycle time is 0.012 sec, therefore the sapling frequency is about 83.3 Hz. The code expects the acquisition of 300 samples, consequently the acquisition time is about 3 seconds. This is enough to guarantee the correct inflation of the soft finger and the acquisition of a reasonable amount of data.

All the acquisition files can be obtained by removing the micro SD card from the reader and putting it into a PC.

Hardware

Once the software part is ready, it's necessary to design an experimental setup. This setup has to be realized to ensure reliable acquisition, avoiding inaccuracies and errors that will be reflected into the reliability of the trained model. In particular there are two requirements that need to be satisfied:

1. The end of the robot from which the air is inserted must remain as steady as possible, so that the values read from the sensor are not affected by this motion. Furthermore, this guarantees continuity and comparability between the various acquisitions.
2. As described at the beginning of this chapter, traditional force sensors are too stiff so they cannot be used to detect when the robot touches an object. To obtain reliable data is necessary that the robot is in contact with the object before the acquisition starts. For this reason, it is useful to have a system that allows the robot to be physically positioned at different heights, depending on the object to be touched.

Following these two requirements the structure shown in the following images has been designed and realized by 3D printing technology. As for the molds, this component is realized with PLA.

Because it hasn't small details and due to its big dimension, the resolution used during the printing is low (0,3 mm).

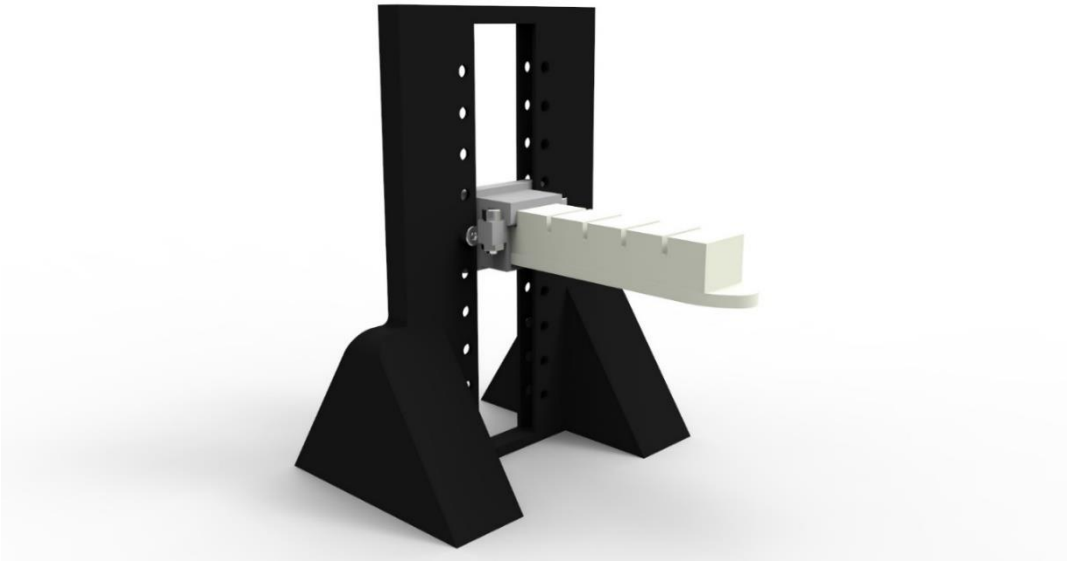

Figure 4-12 Experimental structure 3D model

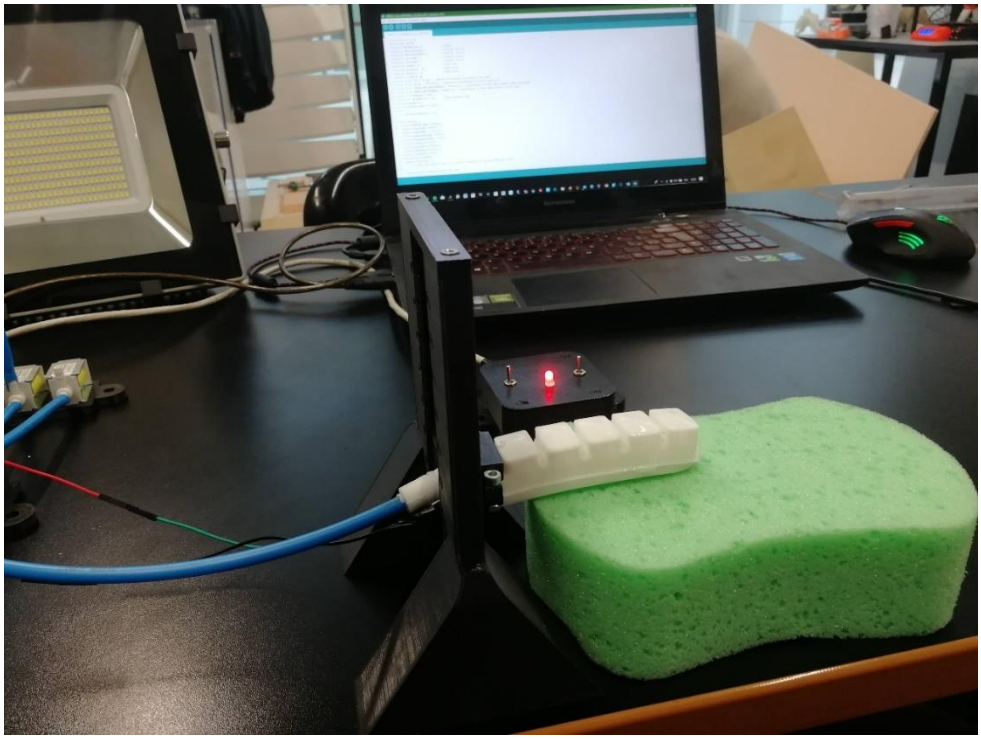

Figure 4-13 Experimental structure

This solution takes advantage of the already implemented clamping system and allows to place the soft finger at different height thanks to the various holes.

Results

Despite the optimization of the code and the hardware aimed at making the acquisition process simple and fluid, the process itself turns out to be quite slow and intricate having to request the presence of an operator. The time to get a large and complete database is very long and it is therefore difficult to think of being able to obtain a large amount of data capable of operating a reliable training process. Nevertheless, the results obtained will give an indication of the goodness of the developed technology and process.

Considering the thesis work carried out, the database related to the training of the model contains 112 acquisitions obtained by taking into consideration various objects of different materials, both hard and soft. The following table summarizes the measurements made for the training process.

OBJECT	NUMBER OF ACQUISITION	FEATURE
SPONGE 1	24	SOFT
PLASTIC BOX	16	HARD
FOAM RUBBER	8	SOFT
CARTON BOX	8	HARD
PACKAGING POLYSTYRENE	16	HARD
SPONGE 2	8	SOFT
CARTON BOX 2	16	HARD
SYNTHETIC COTTON	16	SOFT

Table 4-4-1 Acquisitions features

Once these data are available, it is possible to run the program and to analyze the training results. As said before, the *Classification Learner* tool provides different results on the base of different training algorithms. Using the previously described model it's possible to investigate which model is the best one for a specific application. In this specific training session models from 100% to 33.7% of accuracy are obtained. Both the limit cases are not acceptable for a different reason. The one with the lowest accuracy is maybe obtained with an algorithm that is not suitable with the available data. The excellence of the model with the highest accuracy cannot be considered reliable, as said before, for the low number of acquired data. The following results are referred to two different models obtained with different algorithms with almost the same accuracy. These two models will be also considered during the prediction phase. The first one is obtained with the *Support vector machines* (SVM) algorithm and it has an accuracy of 95.5% during the training phase. The second one, coming from the *k-nearest neighbors'* (KNN) algorithm, has instead an accuracy of 93.8% referred to the training database. Since the key of the work is not the development of the training algorithm, the two ones mentioned before are briefly described in the "*State of the art"* section.

The first picture shows the original data set, while the other ones show the results of the training itself by means of the analysis tools. In particular, for each tool the result of the SVM model is shown first and the result of the KNN model is shown per second.

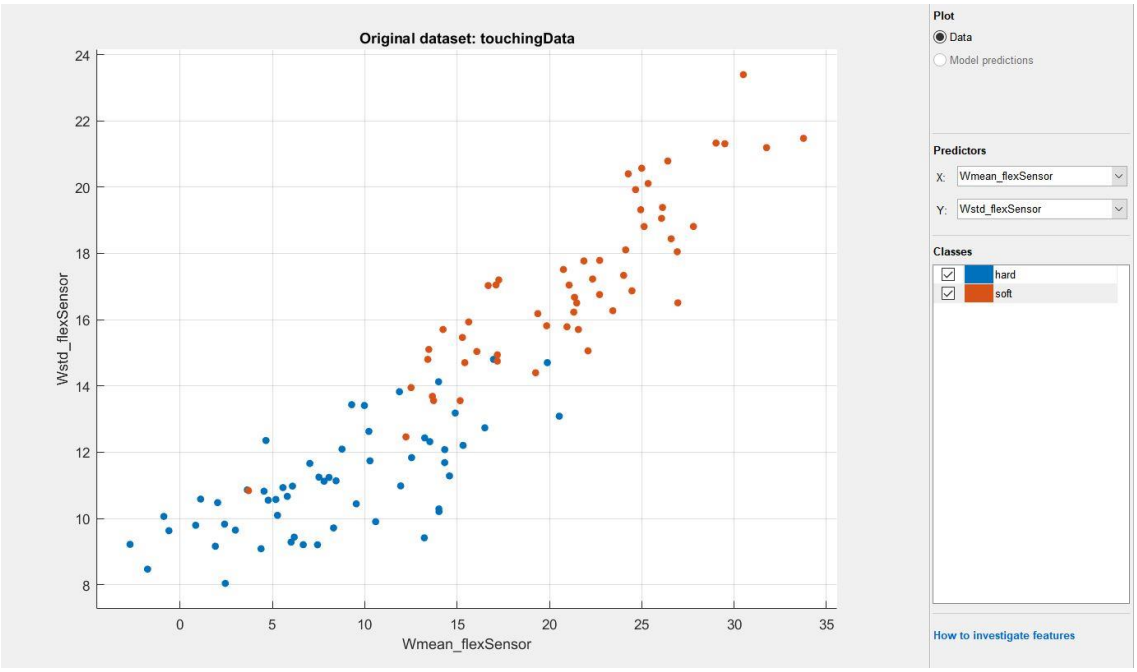

Figure 4-14 Original dataset

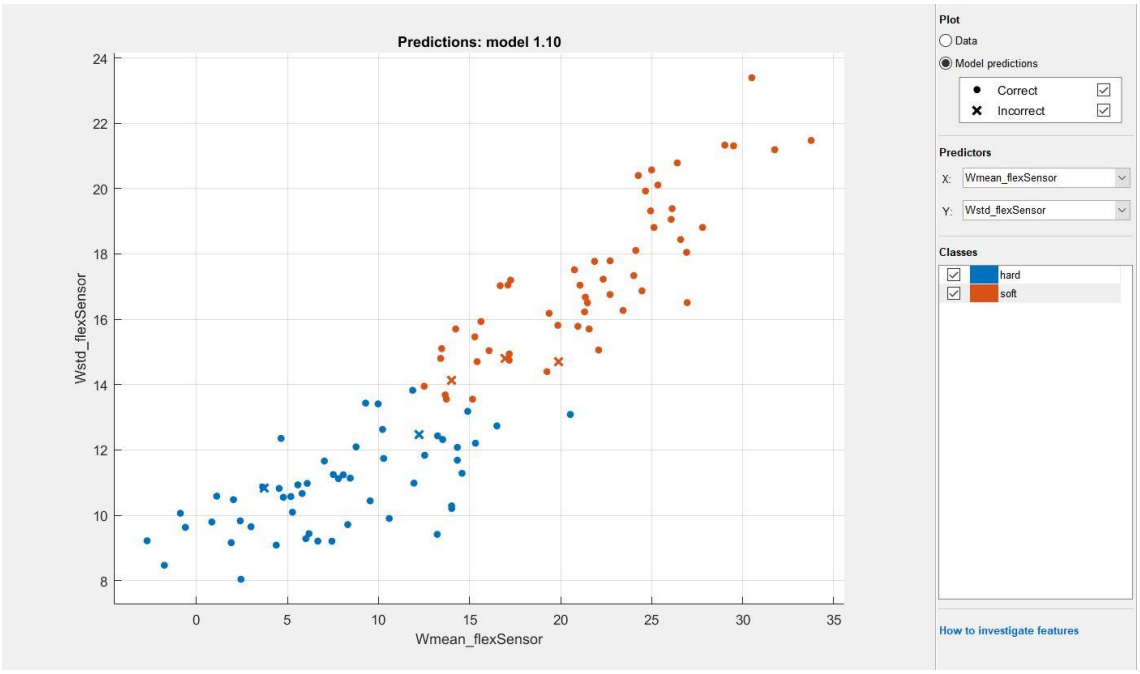

Figure 4-15 SVM model, scatter plot

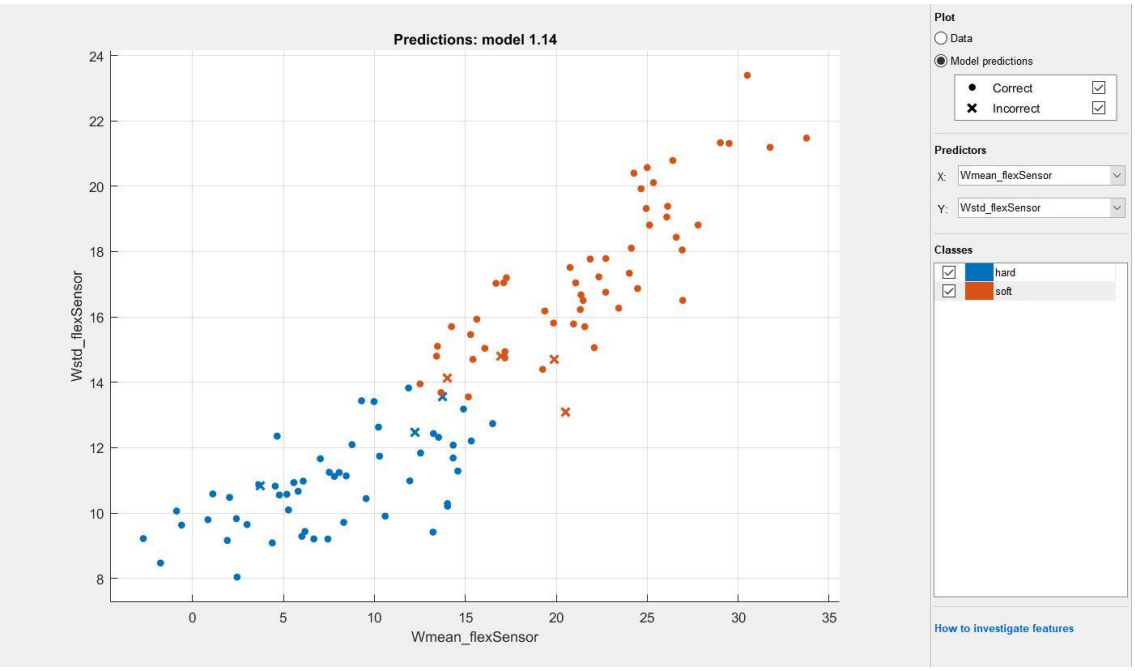

Figure 4-17 KNN model, scatter plot

Figure 4-16 SVM model, confusion matrix

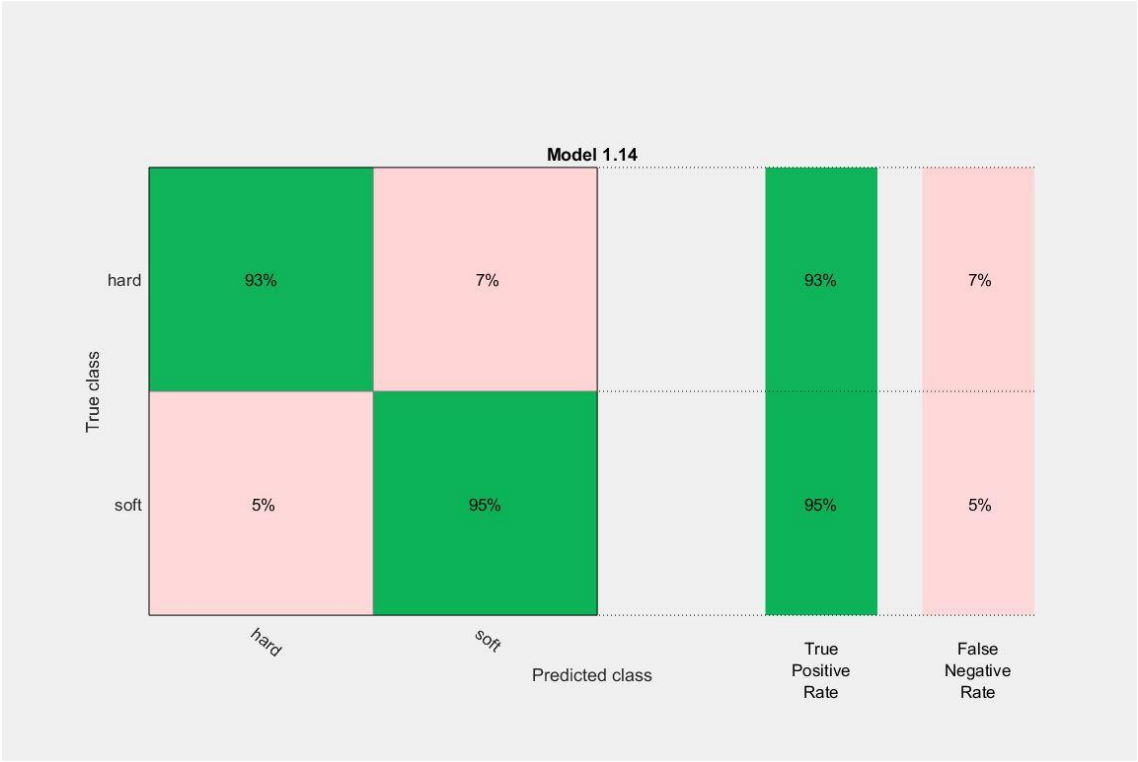

Figure 4-19 KNN model, confusion matrix

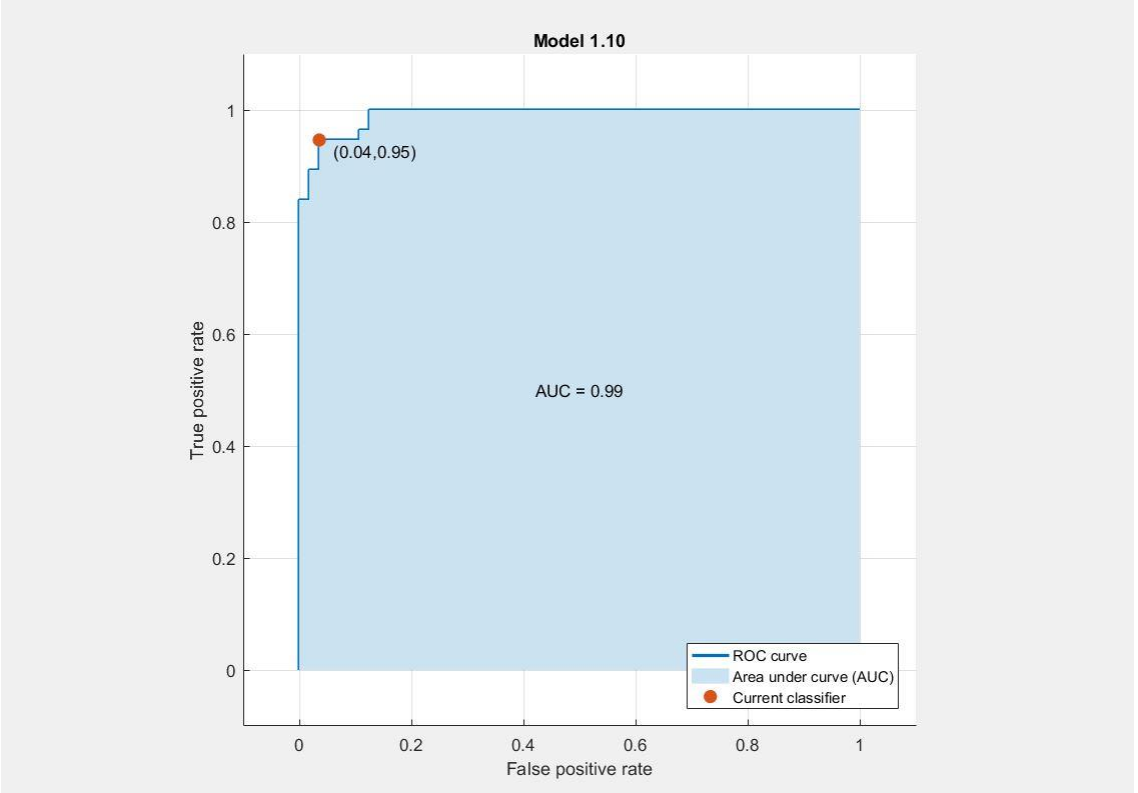

Figure 4-18 SVM model, ROC curve

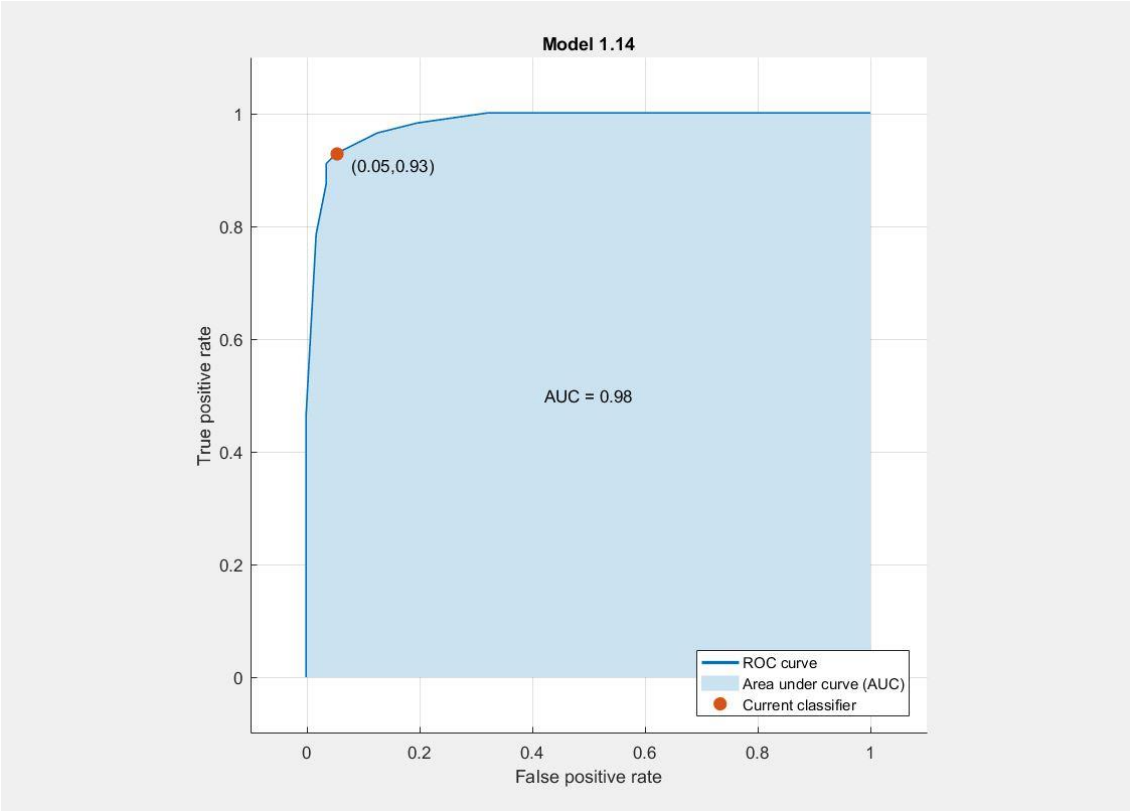

Figure 4-20 KNN model, ROC curve

Once the models are trained and stored, it's possible to predict the feature of an unknown object starting from the model itself. For this last part or the work other acquisitions are requested. To put the created models even more to the test, these acquisitions must also be made on objects not previously used during the training phase. Moreover, the program expects the presence of a *.txt* file containing the real feature of each acquisition. This text file is required in order to evaluate whether the model prediction is correct or not. For the prediction phase 30 acquisitions are taken into account.

The following images show the results of the prediction based on the two previously mentioned models. As before, for each kind of figure the first one is always refereed to the SVM model while the second one to the KNN model.

Figure 4-21 SVM model, results histogram

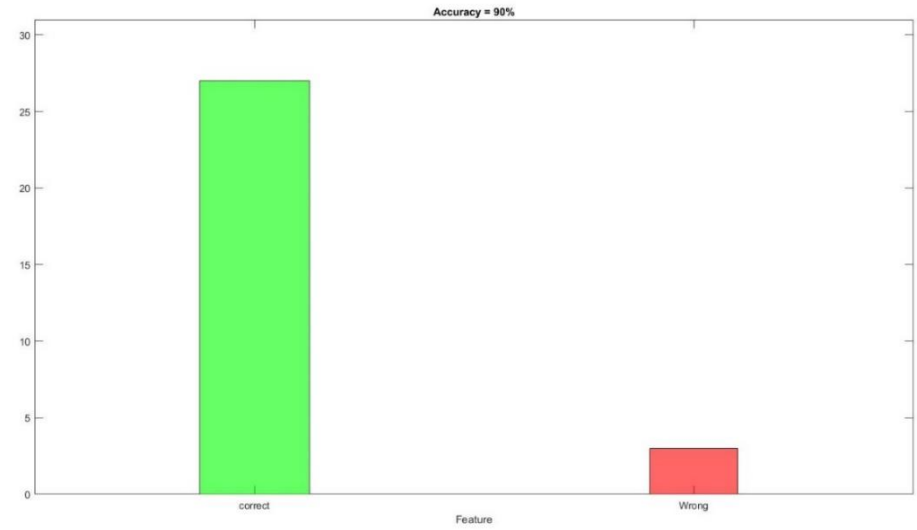

Figure 4-20 SVM model, results histogram

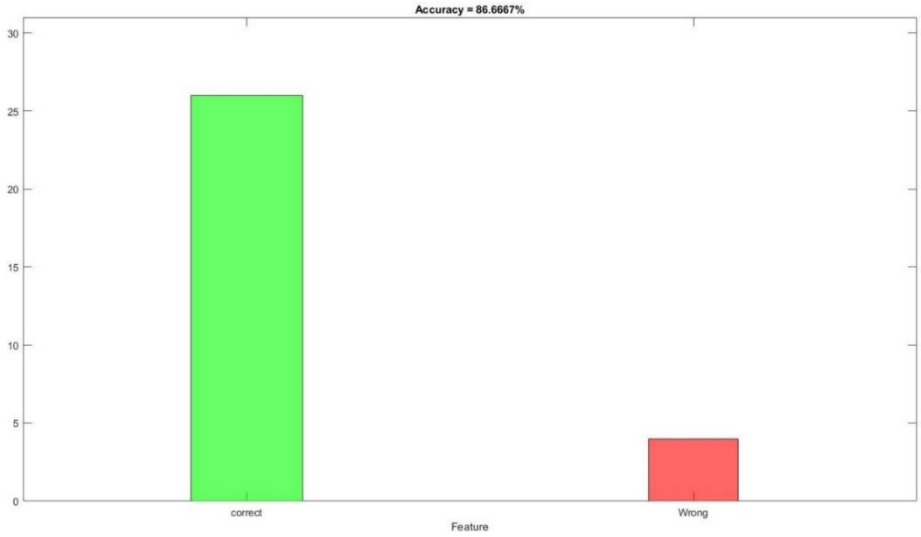

Figure 4-21 KNN model, results histogram

The results show that the overall accuracy of both the model is less than the accuracy estimated by the *Classification Learner* tool. This can be attributed to the fact that objects different to ones used in the training phase have been considered. Also, only 30 acquisitions have been used to test the models, so even just a mistake, for statistical reasons, can drastically reduce the overall accuracy. Moreover, it must be considered the low reliability of the models due to the low number of data used for the training phase. The majority of the errors are in the recognition of hard objects that are confused with soft ones. In fact, some objects that can be considered hard by human touch are instead recognized as soft by the soft robot. Anyway, this experimental session shows that the software and hardware designed work properly and that the developed technology can be effectively applied.

SVM model

PredictedFeature	RealFeature	Results
soft	soft	Correct
soft	soft	Correct
soft	soft	Correct
hard	soft	Wrong
soft	soft	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
soft	soft	Correct
soft	soft	Correct
soft	soft	Correct
soft	soft	Correct
soft	soft	Correct
soft	hard	Wrong
hard	hard	Correct
hard	hard	Correct
soft	hard	Wrong
hard	hard	Correct
hard	hard	Correct
soft	soft	Correct
soft	soft	Correct
soft	soft	Correct

Table 4-2 SVM model, results

KNN model

PredictedFeature	RealFeature	Results
soft	soft	Correct
soft	soft	Correct
soft	soft	Correct
hard	soft	Wrong
soft	soft	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
hard	hard	Correct
soft	soft	Correct
soft	soft	Correct
soft	soft	Correct
soft	soft	Correct
soft	soft	Correct
soft	hard	Wrong
hard	hard	Correct
hard	hard	Correct
soft	hard	Wrong
hard	hard	Correct
soft	hard	Wrong
soft	soft	Correct
soft	soft	Correct
soft	soft	Correct

Table 4-3 KNN model, results

5. CONCLUSIONS

Soft robotic technology can be very useful in the food industries, making possible the grasp of both "hard" foods and "soft" foods. This technology could also be used in warehouses that require the presence of very different products, making useless the use of specific grippers for each type of object. Obviously, the potential of this technology increases with the integration of sensors and artificial intelligence as developed in this work.

As a conclusion of the thesis work, I wanted to realize a conventional soft robotic gripper able to grab objects. For this purpose, the choice is to adopt three soft fingers realized as explained in the previous chapters. The electrical circuit and the Arduino's sketch able to guarantee the proper behavior of the gripper are the same explained in the "*Electrical circuit design*" and "*Arduino's sketch*" section. A single pump is able to completely inflate three soft fingers. The gripper is made by two main bodies. On the upper case the fingers are fixed at 120° from each other thanks to the clamping system. With this arrangement it is possible to have a uniform grip of the objects. The second case works as a simple cover on the bottom. The clamping system is modified and now has four screws instead of two. The main parts are assembled together by means of three screws. The grasp of the objects can be manually controlled from the handle of the gripper, equipped with two switches and a status LED. The realized tool revealed itself able to grab objects of different shape, dimensions and hardness. Unfortunately, it is not possible to grasp objects whose width is greater than the space present between the three fingers. To overcome this problem, the gripper can be improved using a system able to create also vacuum. In such a way the fingers can bend in the opposite direction making the grip of bigger objects possible. This gripper could also be modified to become an end-effector to be placed on a robot by adding a proper system able to mount the gripper on the robot itself. In this way the electric circuit, and therefore the pump and the valves, would be controlled by the robot itself.
Many problems subsist instead in the realization of a smart gripper and for this reason it's not addressed in this work. The main issue is the contact between the soft fingers and the object to be taken. First of all, as shown in the "Sensors" section, the majority of the force sensors on the market detect punctual contact and are too stiff to accomplish the bend motion. The second issue is how to integrate the contact sensors of each finger to the other. It could happen that a finger touches the object before another one, creating a discrepancy between the acquisitions. In this way it is impossible to use these data inside the created MATLAB code.

What has been done during this work makes it possible to state that soft robotic technology can be evolved with the introduction of artificial intelligence. The experimental tests conducted show that the integration of the sensor into the soft robot is possible and that the developed MATLAB code works correctly for the training of

predictive models. As previously stated, the results of this work can be refined by increasing the size of the original database, increasing so the reliability of the results.

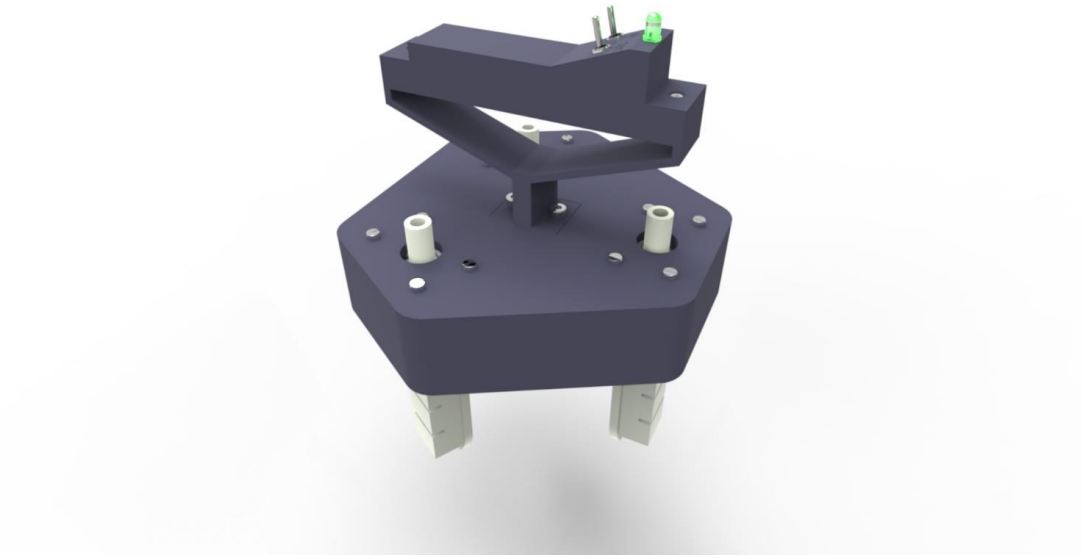

Figure 5-1 Soft robotic gripper 3D model

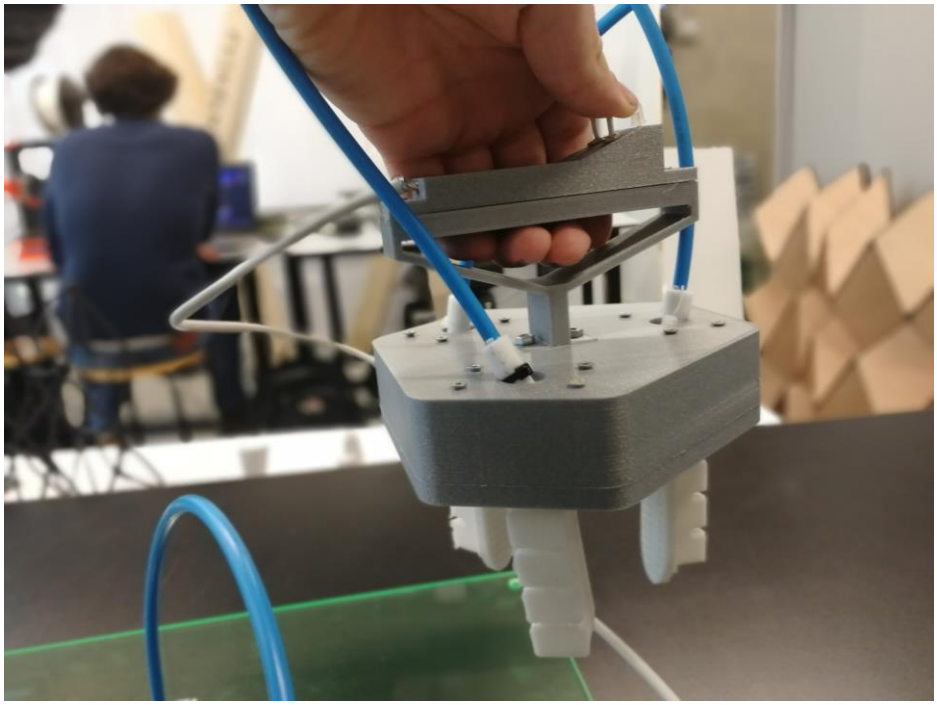

Figure 5-2 3D printed soft Robotics gripper

6. APPENDIX

The appendix will first describe the components used to construct the electric circuit and then it will show the codes developed for Arduino, both for the simple robot motion and for data acquisition.

ELECTRICAL CIRCUIT COMPONENTS

As explained in Chapter 3, a pump and two solenoids valves are mandatory for the robot to move.

Figure 6-1 Electrical pump

Figure 6-2 Solenoide valve

To guarantee the integration of the mechanical parts and to avoid electrical issue some specific electrical components are mandatory. After some research on the web some components have been selected.
The functions of each component present in the circuit are explained below:

Arduino: The Arduino microcontroller is the brain of the regulator system. Here, control commands are written to inflate/deflate the robot and read the sensors value.

Figure 6-3 Arduino board

Transistors: TIP120 which is an NPN Darlington transistor is required to interface the high current external devices to the microcontroller. The pump and solenoid valves make use of high current and cannot be directly connected to the digital I/O pins of the microcontroller. One transistor is required for each of the two the solenoid valves and air pump to drive them at their appropriate current while ensuring that they are controlled by Arduino. When the output pin of the controller to which the valve/pump is connected to is high (5V), the transistor is active and current is able to flow into the base turning on the TIP120 transistor. When the output pin is low (0V), the collector current, IC, will be zero therefore the transistor is off. In general, the transistor is used to switch current on and off to the solenoid valves and pump; through the transistor, a small current supplied by the microcontroller will switch on a large current that drives the valves and pump. A 1Kohms resistor can be placed between the microcontroller output and the base. This resistor controls the maximum current that can flow from collector to emitter. The circuit is designed to make sure that large currents needed by the dc pump and valves are not supplied by the microcontroller but by the external circuitry powered by an external power supply.

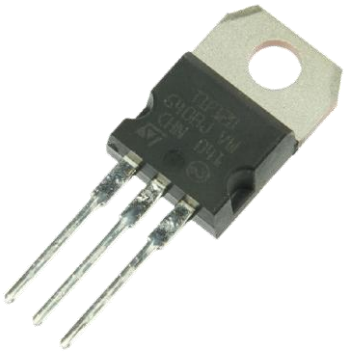

Figure 6-4 TIP120 transistor

Diodes: The 1N4001 diodes are used as flyback diodes for the valves and pump. Three diodes are required for the pump and valves. When the pump or valve is subjected to a large change in current, such as when the transistor switches, the inductor in the pump and valves presents a large back-emf (back-electromotive force or voltage). This large voltage spike could be harmful, so flyback diodes were required to dissipate this spike.

Figure 6-5 1N4001 diode

Power supply: Arduino can supply current to feed the mechanical components, but it's not enough. The current drawn by the pump is about 150mA and the current drawn by the each of the valves are about 180mA each. Therefore, an external power supply should be able to supply at least 800mA of current. A 12 V or 24 V power supply can be used for this purpose.

Figure 6-6 Power supply

Switching Voltage Regulator: The switching voltage regulator is needed to produce a 6V DC output from the power supply to power up the valves and pump at the appropriate voltage. The LM2596 DC-DC Switching Adjustable Step-Down Voltage Regulator Buck Converter is used to achieve this. It is more efficient to use a switching voltage regulator compared to using a linear voltage regulator because a current of about 800mA is required for the pumps and valves when the circuit is in operation which results in significant amount of heat.

Figure 6-7 LM2596 switching voltage regulator

In addition to these mandatory components, others can be used to improve the quality of the circuit.

ON/OFF toggle switch: to actuate the pump and the valve ON/OFF toggle switch are useful. Contrary to a simple button that required to be pushed to allow the current passage, with this component is possible to maintain the ON status permanently. Two of this toggle switch are requested to control the intake phase and the outlet phase.

Figure 6-8 ON/OFF toggle switch

RGB status LED: to monitor which part of the script is running, an RGB LED can be programmed to behave as a status LED

Figure 6-9 RGB LED

A little gangplank is designed and 3D printed to keep these additional component, making a sort of control board from which it's possible to actuate the system and monitor its status.

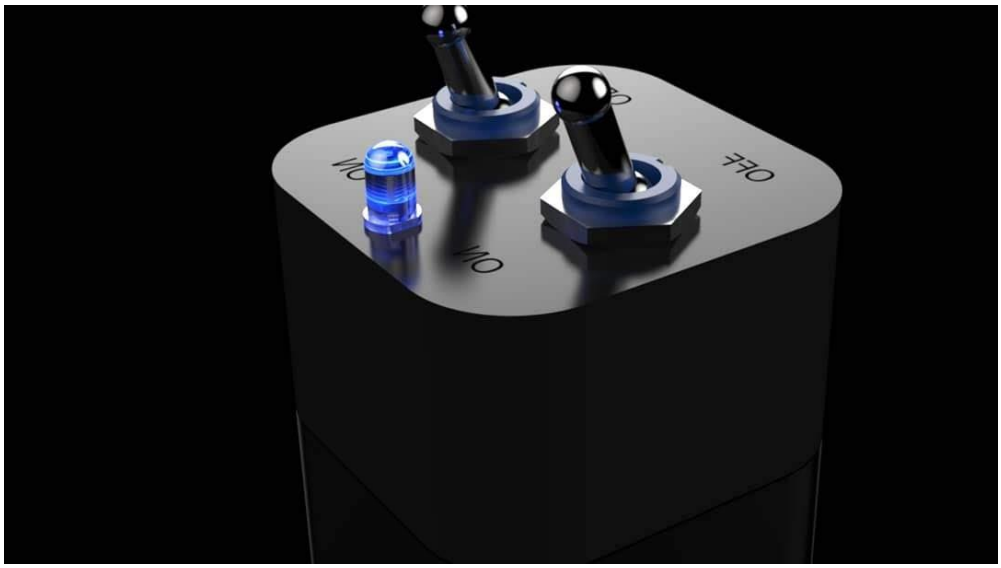

Figure 6-10 Gangplank 3D model

Figure 6-11 Gangplank

ARDUINO CODE

In the following pages it will be shown first the code used to generate the simple motion of the soft robot.
The second part shows instead the modifications needed to allow the data acquisition by means of the Micro SD card module. Some notes are present in the code to explain better what each section does.

Robot simple motion sketch

```
#define MOTOR_PIN 3             //Pump
#define buttonPinIN 8           //Intake switch
#define buttonPinOUT 2          //Outlet switch
#define valveIN 5               //Intake valve
#define valveOUT 7              //Outlet valve
#define redPin  6              //LED red
#define greenPin  10            //LED gren
#define bluePin  9              //LED blue
#define flexPin A0
const float VCC = 4.98; // Measured voltage of Ardunio 5V line
const float R_DIV = 47700.0; // Measured resistance of 4.7k resistor
const float STRAIGHT_RESISTANCE = 37300.0; // resistance of the flex sensor when straight
const float BEND_RESISTANCE = 90000.0; // resistance of the flex sensor at 90 deg

void setup() {
 pinMode(MOTOR_PIN, OUTPUT);
 pinMode(valveIN, OUTPUT) ;
 pinMode(valveOUT , OUTPUT) ;
 pinMode(buttonPinIN, INPUT);
 pinMode(buttonPinOUT, INPUT);
 pinMode(redPin,OUTPUT) ;
 pinMode(greenPin,OUTPUT) ;
 pinMode(bluePin,OUTPUT) ;
 pinMode(flexPin,INPUT) ;
 Serial.begin(9600);
}

void rgb (int red, int green, int blue){   //LED color management
analogWrite(redPin,red) ;
analogWrite(greenPin,green) ;
analogWrite(bluePin,blue) ;
}

void loop() {

    while(digitalRead(buttonPinIN)==HIGH && digitalRead(buttonPinOUT)==LOW){
    //Itake switch HIGH, outlet switch LOW ===> pump ON , valveIN ON , valveOUT OFF
    rgb(0,255,0) ;      //green LED
    analogWrite(MOTOR_PIN,255) ;
    analogWrite(valveIN , 255) ;
    analogWrite(valveOUT , 0) ;
    int flexADC = analogRead(flexPin);  //sensor data acquisition
    float flexV = flexADC * VCC / 1023.0;
    float flexR = R_DIV * (VCC / flexV - 1.0);
    float angle = map(flexR, STRAIGHT_RESISTANCE, BEND_RESISTANCE,
                  0, 90.0);
```

```
    //angle = angle-146 ;
    char angle_text[30];
    dtostrf(angle, 10, 10, angle_text);
    char text[31];
    snprintf(text, 31, "%s", angle_text);
    Serial.println(text);
    if (digitalRead(buttonPinIN)==LOW){
      return 0  ;
      }
    }

    while(digitalRead(buttonPinIN)==LOW && digitalRead(buttonPinOUT)==LOW){
    //Itake switch LOW, outlet switch LOW ===> pump OFF , valveIN OFF , valveOUT OFF
    rgb(255,0,0) ;      //red LED
    analogWrite(MOTOR_PIN,0);
    analogWrite(valveIN , 0) ;
    analogWrite(valveOUT , 0) ;
    int flexADC = analogRead(flexPin);
    float flexV = flexADC * VCC / 1023.0;
    float flexR = R_DIV * (VCC / flexV - 1.0);
    float angle = map(flexR, STRAIGHT_RESISTANCE, BEND_RESISTANCE,
                  0, 90.0);
    //angle = angle-146;
    char angle_text[30];
    dtostrf(angle, 10, 10, angle_text);
    char text[31];
    snprintf(text, 31, "%s", angle_text);
    Serial.println(text);
    if (digitalRead(buttonPinIN)==HIGH||digitalRead(buttonPinOUT)==HIGH){
    return 0 ;
      }
    }

    while( digitalRead(buttonPinOUT)==HIGH && digitalRead(buttonPinIN)==LOW){
    //Itake switch LOW, outlet switch HIGH ===> pump OFF , valveIN OFF , valveOUT IN
    rgb(255,0,255) ;   //violet LED
    delay(200) ;
    analogWrite(MOTOR_PIN,0);
    analogWrite(valveIN , 0) ;
    analogWrite(valveOUT , 250) ;
    int flexADC = analogRead(flexPin);
    float flexV = flexADC * VCC / 1023.0;
    float flexR = R_DIV * (VCC / flexV - 1.0);
    float angle = map(flexR, STRAIGHT_RESISTANCE, BEND_RESISTANCE,
                  0, 90.0);
    //angle = angle-146 ;
    char angle_text[30];
    dtostrf(angle, 10, 10, angle_text);
    char text[31];
    snprintf(text, 31, "%s", angle_text);
    Serial.println(text);
  }
  delay(50) ;
}
```

Data acquisition sketch

```
#include <SPI.h>
#include <SD.h>
#define MOTOR_PIN 3             //Pump
#define buttonPinIN 8           //Intake switch
#define buttonPinOUT 2          //Outlet switch
#define valveIN 5               //Intake valve
#define valveOUT 7              //Outlet valve
#define redPin  6               //LED red
#define greenPin  10            //LED gren
#define bluePin  9              //LED blue
#define flexPin A0
const float VCC = 4.98; // Measured voltage of Ardunio 5V line
const float R_DIV = 47700.0; // Measured resistance of 4.7k resistor
const float STRAIGHT_RESISTANCE = 37300.0; // resistance of the flex sensor when straight
const float BEND_RESISTANCE = 90000.0; // resistance of the flex sensor at 90 deg
const int nSample = 300 ;
const int chipSelect = 4;     //Chip select PIN
String nameFile ; |
String estensione = ".txt" ;
char c ;
int correctionFactor = 0 ;

void setup() {
 pinMode(MOTOR_PIN, OUTPUT);
 pinMode(valveIN, OUTPUT) ;
 pinMode(valveOUT , OUTPUT) ;
 pinMode(buttonPinIN, INPUT);
 pinMode(buttonPinOUT, INPUT);
 pinMode(redPin,OUTPUT) ;
 pinMode(greenPin,OUTPUT) ;
 pinMode(bluePin,OUTPUT) ;
 pinMode(flexPin,INPUT) ;
 Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  Serial.print("Initializing SD card...");
  // see if the card is present and can be initialized:
   if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  Serial.println("card initialized.");
}

void rgb (int red, int green, int blue){   //LED color management
analogWrite(redPin,red) ;
analogWrite(greenPin,green) ;
analogWrite(bluePin,blue) ;
}

```

```
void loop() {

    while(digitalRead(buttonPinIN)==HIGH && digitalRead(buttonPinOUT)==LOW){
    //Itake switch HIGH, outlet switch LOW ===> pump ON , valveIN ON , valveOUT OFF
    rgb(0,255,0) ;      //green LED
    analogWrite(MOTOR_PIN,255) ;
    analogWrite(valveIN , 255) ;
    analogWrite(valveOUT , 0) ;
    File dataFile = SD.open(nameFile, FILE_WRITE);
     for(int i=0;i<nSample;i++){ //for any acquisition repeat the following lines
      float cycleStart = millis() ;
      int flexADC = analogRead(flexPin);  //sensor data acquisition
      float flexV = flexADC * VCC / 1023.0;
      float flexR = R_DIV * (VCC / flexV - 1.0);
      float angle = map(flexR, STRAIGHT_RESISTANCE, BEND_RESISTANCE,
                  0, 90.0);
      angle = angle - correctionFactor  ;
       if (dataFile) {// if the file is available, write to it:
        dataFile.println(angle);

        delay (10) ;
        if(i==0){
         float cycleTime = (millis() - cycleStart)/1000 ;
         float freq =1/cycleTime  ;
         Serial.print("Cycle time: ");
         Serial.print(cycleTime) ;
         Serial.println("sec");
      }
       Serial.println(angle);// print to the serial port too:
       }
      else {
       Serial.println("error opening datalog.txt");// if the file isn't open, pop up an error:
      }
      if (digitalRead(buttonPinIN)==LOW){
       return 0  ;
      }

     }
    dataFile.close();

    analogWrite(MOTOR_PIN,0) ;
    analogWrite(valveIN , 0) ;
    analogWrite(valveOUT , 0) ;
     for(int jj=0;jj<7;jj++){
      rgb(255,200,0) ;
      delay (500) ;
      rgb(0,0,0) ;
      delay (500) ;
     }
    delay (1000);
   }

    while(digitalRead(buttonPinIN)==LOW && digitalRead(buttonPinOUT)==LOW){
     //Itake switch LOW, outlet switch LOW ===> pump OFF , valveIN OFF , valveOUT OFF
      if(Serial.available()){
       nameFile="" ;
        do{
         if(Serial.available())
          c=Serial.read();
          if(c != '\n')
          nameFile+=c ;
         }
        while(c != '\n') ;
         nameFile = nameFile + estensione ;
         Serial.print("File name: ") ;
         Serial.print(nameFile) ;
         Serial.println("") ;
        }
```

```
    rgb(255,0,0) ;      //red LED
    analogWrite(MOTOR_PIN,0);
    analogWrite(valveIN , 0) ;
    analogWrite(valveOUT , 0) ;
     if (digitalRead(buttonPinIN)==HIGH||digitalRead(buttonPinOUT)==HIGH){
      return 0 ;
      }
     delay (200) ;
   }

   while( digitalRead(buttonPinOUT)==HIGH && digitalRead(buttonPinIN)==LOW){
    //Itake switch LOW, outlet switch HIGH ===> pump OFF , valveIN OFF , valveOUT IN
    rgb(255,0,255) ;   //violet LED
    delay(200) ;
    analogWrite(MOTOR_PIN,0);
    analogWrite(valveIN , 0) ;
    analogWrite(valveOUT , 250) ;
   }
   //delay(50) ;
}
```