POLITECNICO DI TORINO

Master's Degree in Aerospace Engineering

Master's Degree Thesis

Design and testing of an experimental setup for gust alleviation through active flow control

Supervisors

Prof. Gioacchino CAFIERO

Ing. Enrico AMICO

Candidate

Alessio CARNEMOLLA

March 2025

Abstract

This study aims to design and evaluate an efficient setup for a flapping wing operating in artificially generated gusts created by a Windshaper (WS) with dimensions 2×1.5 m. The WS allows flow speeds up to 20 m/s and enables the generation of various time-dependent wind profiles. The setup features a NACA0018 airfoil equipped with a dynamically actuated flap, controlled in real time by a servo motor. The flapping wing structure and its lateral support are 3D-printed and attached to a vertical beam. Two pressure sensors are embedded in the airfoil's upper surface: one positioned before the maximum thickness point and the other after it. Additionally, a hot wire anemometer (HW) is placed in the wake region along the airfoil's symmetry plane to measure velocity fluctuations. The primary objective of this experiment is to develop an optimized setup capable of acquiring and processing high frequency data efficiently. Once the system is fully tested and its properties verified, it will serve as a reliable tool for future algorithm development and testing.

Contents

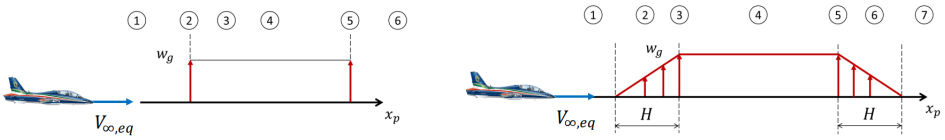
Chapter 1

Introduction

1.1 The Gust Problem

The term *gust* refers to sudden and temporary variations in wind speed. This phenomenon is observed as a rapid change in wind direction or force, which can affect aircraft performances, stability, and control during flight. Gusts can be particularly risky during the takeoff and landing phases when the aircraft is closer to the ground and has less time to adapt to changes in wind speed and direction. However, this phenomenon should not be underestimated during any flight phase.

The *gusts* occurrence can be attributed to various factors such as atmospheric turbulence, air movements, thermal changes in the atmosphere or specific cases of *downslope wind*, such as the *Föhn*.



(a) Instantaneous Gust[1]. (b) Gradual Gust[1].

Figure 1.1: Types of gusts.

A *gust* can be *instantaneous* (without a front), with a speed of w_g or *gradual* (for example linear), with a speed of w_g and a gust front length H. To better understand the effects of this phenomenon on aircraft performance, we now introduce the *Maneuvering diagram* of an aircraft.

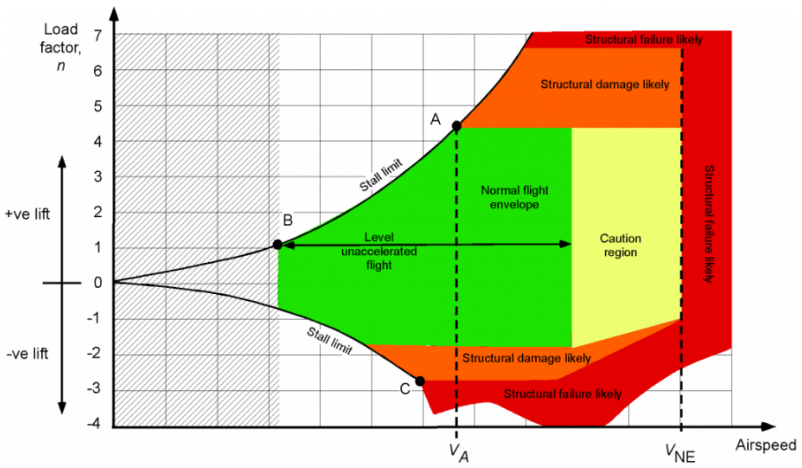


Figure 1.2: *Maneuvering Diagram*, specific for each aircraft[2].

This diagram, derived from experimental data, shows the airspeed (sometimes the *equivalent* speed[1] or Mach number) on the x-axis and the *load factor* on the y-axis. The load factor is defined as:

$$n = \frac{L}{W} = \frac{\rho_0 V_{EAS}^2 C_L}{2\frac{W}{S}}$$

There are two main uses for a $V - n$ diagram, like the one mentioned above: as a *Maneuvering diagram*, as previously considered, and as *Gust diagram*, considering the gust impact on the airplane. The diagrams are basically the same, but the information presented has different interpretations. Gust can affect the aircraft from any direction, however, upward (vertical) gusts have the most pronounced effects on the aircraft in terms of aerodynamic response and induced load factor.

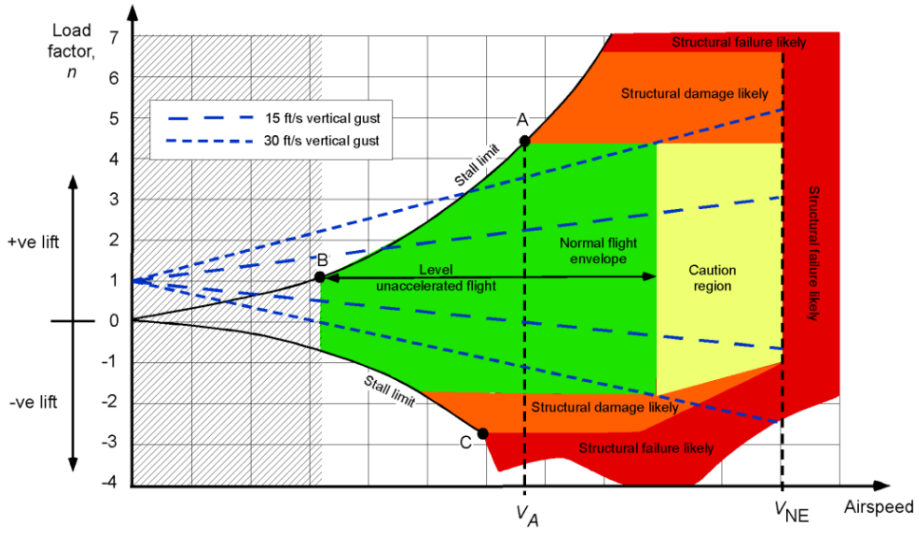


Figure 1.3: *Gust Diagram.*[2]

It is noted that the load factor, for a given gust intensity, decreases with increasing wing loading, $\frac{W}{S}$. This means that smaller, lighter aircraft (generally with lower wing loadings) tend to respond much more to gusts than larger aircraft.

Additionally the load factor, for a given gust intensity, varies linearly with airspeed. Therefore, it is possible to plot lines with different values of W_g on the $V - n$ diagram, as shown in figure. These sets of straight lines show the aircraft's load factor at a given airspeed due to the effects of gusts. When one of these lines crosses the maximum (or minimum) load factor limit, the corresponding airspeed is the maximum speed that can be maintained without stalling the wing or exceeding the maximum structural load factor allowed. Each type of aircraft will have its own specific diagram regarding gust intensity and airspeeds[2].

The resulting load factor of the aircraft can exceed the allowable load limit if the gust is sufficiently severe. However, it cannot exceed the maximum load factor without causing significant structural damages or failures. Even if the load factor limit is exceeded in flight, there may still be non-fatal damages, and the aircraft must be carefully inspected before flight[2].

It is therefore necessary to control this phenomenon as much as possible to avoid problems, which could range from minor to catastrophic. For this purpose, various methods can be employed to control the parameters.

[1] V_{EAS}: Equivalent Airspeed. This value refers to the velocity adjusted for air density variations with altitude.

1.2 Control Methods

1.2.1 Control Methods: PID

Among the solutions that can be employed to better control variations in quantities there are *PID Controllers*. Proportional-Integral-Derivative (PID) control is a negative feedback systems widely used in automatic control systems. This controller was developed to offer more precise control compared to *On-Off* control, which leads to significant fluctuations in the variables, unacceptable in more specific applications. They are the most common feedback control system in industry, especially in the *PI* version (without the derivative action)[3].

The controller receives an input value from a process and compares it with a reference value. The difference, known as the error signal, is then used to determine the output variable of the controller, which is the variable calculated in the process.

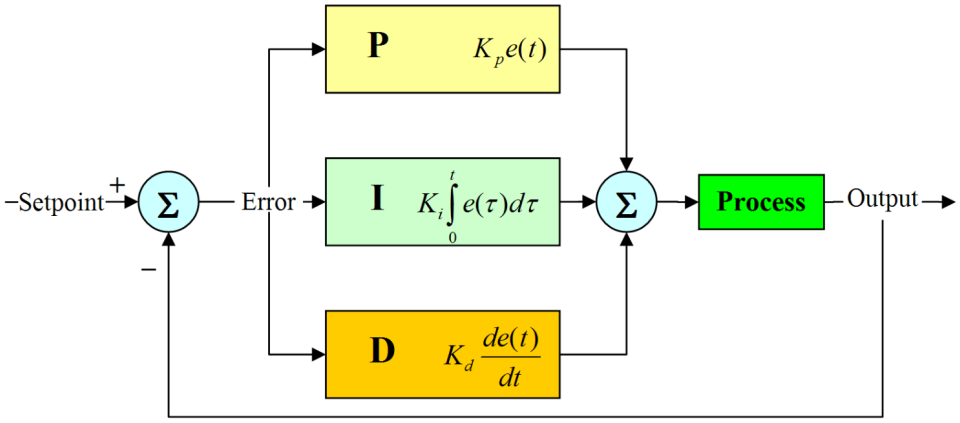


Figure 1.4: PID: *Flow Chart*.

In a *PID* controller, the output variable u_t is generated based on the contribution of three terms:

- the first is proportional to the error between the reference r_t and the variable to be controlled y_t;
- the second is proportional to the integral of the error e_t. In fact, it depends on the average value of the error;
- the third is proportional to the derivative of the error. This value, therefore, depends on the rate of change of the error.

The control law for a *PID* controller can be written as[4]:

$$u_t(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt}$$

where:

- K_P is the *proportional gain*;
- K_I is the *integral gain*;
- K_D is the *derivative gain*;

The three actions of a PID controller are calculated separately and then simply summed algebraically:

$$u(t) = u_P(t) + u_I(t) + u_D(t)$$

Proportional Action *P*

The proportional action is obtained by multiplying the error signal e by an appropriate constant value:

$$u_P(t) = K_P e(t)$$

The control action is simply proportional to the error between the reference value and the variable to be controlled. Thus, the greater the error e_t entering the controller, the greater the control action performed by the *PID*. Using the single proportional action, a difference will be created between the requested value and the actual obtained value. This difference can be reduced by increasing the proportional gain of the controller. However, increasing the proportional gain may produce an increase in oscillations caused by rapid transients[4]. Since proportional action does not guarantee elimination of the error at steady-state, for constant reference signals, it is possible to eliminate the error by adding an appropriate value U to the control variable u_t such that: $u_P(t) = K_P e(t) + U$.

Integral Action *I*

The integral action is proportional to the time integral of the error signal e, multiplied by the constant K_I:

$$u_I(t) = K_I \int_0^t e(\tau) d\tau$$

This definition for the integral action ensures that the controller *remembers* past values of the error signal. Specifically, the value of the integral action is not necessarily zero if the error signal is zero. This property allows the PID to bring the process exactly to the required reference point, where proportional action alone would not be enough. This action is visible after a time interval τ_I, defined as the *latency time*.

Derivative Action *D*

To improve controller performance, the derivative action can be added:

$$u_D(t) = K_D \frac{de(t)}{dt}$$

The idea is to quickly compensate the changes in the error signal: if e is increasing, the derivative action tries to compensate this deviation based on its rate of change, without waiting for the error to become significant (proportional action) or persist for a certain amount of time (integral action). Derivative action is often omitted in PID implementations because it makes them too sensitive: a PID with derivative action, for example, would experience a sharp variation if the reference suddenly changes from one value to another, resulting in a derivative *de* tending to infinity, or being very high. This does not encourage the application of derivative action in cases where the physical actuator should not be subjected to excessive forces.

If well tuned and if the process is sufficiently *tolerant*, however, derivative action can make a significant contribution to the controller performances[3].

PID regulation requires appropriate *tuning* of the parameters used for the various actions. These parameters differ from each other depending on the application and require calibration through experimental data. Tuning can be *manual* or *automatic*.

1.2.2 Control Methods: FLC

The *FLC* methods are control methods based on *fuzzy logic*, an extension of classical logic that deals with uncertainty and imprecision. A *fuzzy set* is an extension of a *crisp set*[2]. *Crisp sets* only allow total or no membership, while *fuzzy sets* allow partial membership. In other words, an element can partially belong to a set[5]. Unlike traditional logic, which operates with binary values (true or false, 1 or 0), fuzzy logic works with values between 0 and 1, allowing for the expression of concepts that are intermediate.

[2]Also called a classical or traditional set, it is a set in which each element can either belong or not belong, with no possibility of intermediate membership.

Classical binary logic assumes only *True* or *False* assertions, using such logic as a control system (for example thermostat *Cold* or *Hot*, corresponding to 0 or 1). *Fuzzy logic* assumes the existence of intermediate values such as *partially true* or *partially false* in all its features. Reminding the thermostat example again, it is possible to obtain a spectrum of values from *High Temperature: Hot, 1* to *Low Temperature: Cold, 0*.

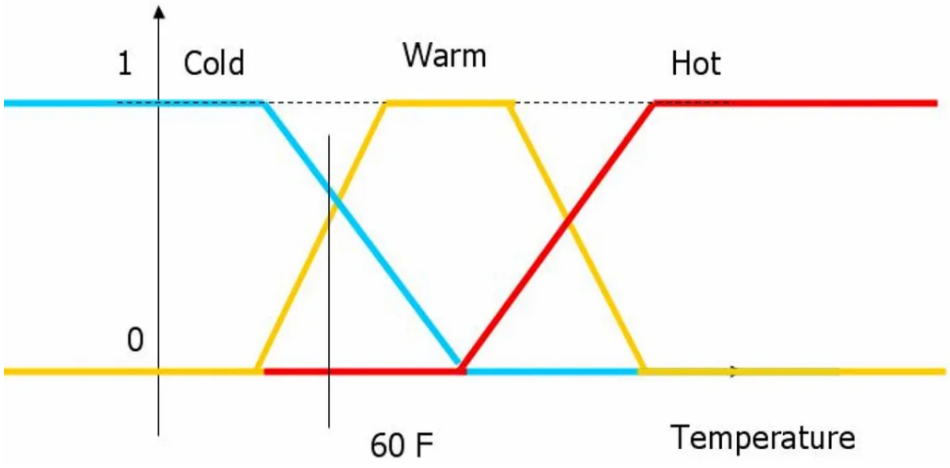


Figure 1.5: Degree of Membership.

This logic follows several steps:

- *Input variables*: input of the variables;
- *Fuzzification*: the process through which variables are converted into specific Degrees of Membership to a particular declared *Fuzzy Set*. This process essentially associates numerical values to specific *linguistic* values declared in the *Fuzzy Set*. Referring to the thermostat example, an input variable, after the fuzzification process, will be associated to an adjective describing *how hot, warm, or cold* it is.
- *IF-THEN rules*;
- *Defuzzification*: the process by which linguistic variables are converted back into numerical variables, similar to the input variables;
- *Output Variables*: output variables.

This control method finds various applications in industrial settings, such as automated control of sluice gates in hydroelectric power plants (*Tokyo Electric Power*), cruise control for vehicles (*Nissan, Subaru*), and temperature control for industrial ovens (*Fuji Electric*)[5].

1.2.3 Control Methods: MPC

MPC (*Model Predictive Controllers*) are an advanced class of control systems used to manage dynamic processes. The main idea is to use a mathematical model to simulate the system being controlled to predict its future behavior and optimize control in real-time. The first MPC control method was introduced in industrial settings in 1978 with an article by Richalet et al., and in 1979 with works published by Shell.

These methods involve the use of:

- a *Model* (**M**): a mathematical model of the dynamic system that describes how the output (for example temperature, speed, position) changes over time in response to inputs (such as commands or forces). This model can be linear or nonlinear;
- a *Prediction* (**P**): a prediction of the system's future behavior based on the current inputs and states, calculated at each time step;

- a *Control* action (**C**). *MPC* solves an optimization problem aiming to minimize an objective function (for example the error between the system's prediction and the desired behavior), while also considering constraints on variables such as inputs and states. After obtaining the optimal solution, *MPC* applies only the first control action (for example the first value of the optimal input) and repeats the process at the next time step[6].

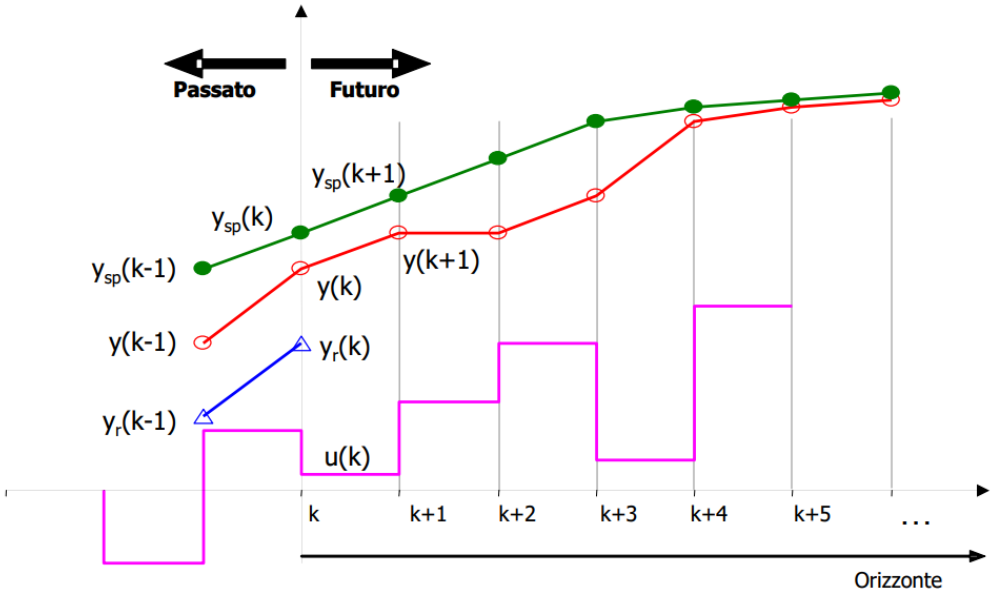


Figure 1.6: Applicability of *MPC*

Where:

- y_{sp}: reference trajectory of the system, the optimal time-varying set point;
- y: the response of the predictive model;
- y_r: the actual value of the variable.

The model M is used to calculate a sequence of control actions that satisfy an appropriate objective function: minimizing the system's response y relative to the optimal trajectory y_{sp} (set point) and minimizing control effort[6].

The core idea behind this technique is to transform the classical control problem into a mathematical optimization problem, making it easy to incorporate constraints and limitations in the real system. Particularly, the model of the entire system is used to predict, over a *finite horizon*, the evolution of the state variables, starting from their current values and based on future control input sequences. The strategy of *MPC* is explained for discrete-time systems, but it is almost equivalent for continuous-time systems. Predictive control is particularly used when constraints are present in the process. These constraints are related to the maximum and minimum values and the rate of change of input variables (manipulated or control variables) and output variables (controlled variables)[7].

The main advantages of *MPC* are[7]:

- it allows to consider constraints in the optimization problem;
- it can be used in multivariable systems;
- it can be used to solve a wide variety of problems if the model is sufficiently accurate.

However, this method also shows some disadvantages[7]:

- building the model could be very complicated;
- deriving the control laws can be quite complex;
- sometimes the cost can be extremely high, leading to a slowdown in the computation of solutions.

1.2.4 Control Methods: DDPC

Data Driven models are based on the use of data. To build such models, algorithms from *machine learning* and *statistical learning* are employed. For this purpose, it is necessary to explore the data, often considering multiple models, and then build a model by applying a specific algorithm. This model is typically used for tasks such as classification, regression, or prediction. In most cases, these models are used to assist decision-makers or to make automated decisions. Naturally, data is a key component in building *Data Driven* models. The quality of the data directly influences the final model, and only good quality data can produce good quality models[8].

Data Driven Predictive Control (DDPC) has been recently proposed as an effective alternative to model-based predictive control (MPC) due to its unique characteristics of being time-efficient and free from *bias*[9]. However, it has also been observed in many examples that noise in the output data can severely compromise the final performance of the system in feedback control, as it affects both the data-based system representation and the control update calculated from the online measurements.

In the field of automatic controls, *ML* (*Machine Learning*) methods have been widely employed and developed for their ability to bypass the need for model descriptions of complex systems. Instead of using state models or physical laws, this description is made through large amounts of collected data, evolving existing techniques such as *MPC* and leading to a significant reduction in design times for control systems. In traditional techniques, modeling usually takes up to 75% of the project time, and it requires various skills[9].

1.2.5 Machine Learning

Machine Learning is a particular area of Computer Science in which the efficiency of a system improves itself by repeating the tasks using data instead of being explicitly programmed. By using Machine Learning (ML) it is possible to *classify* or *predict* patterns: based on some input data, which can be labeled or unlabeled, the algorithm will produce an estimate about a pattern in the data[10]. Generally, ML algorithms involve three main elements: a *Decision Process*, an *Error Function* and a *Model Optimization Process*.

The decision process passes through some input data, which can be labeled or unlabeled. By using this data the algorithm will produce an estimate about a pattern in the data. Then, an error function evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model. If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. This process produces the Model Optimization. The algorithm will repeat this iterative *evaluate and optimize* process, updating weights autonomously until a threshold of accuracy has been met.

Generally, ML is divided in three main branches as: *Supervised Learning*, *Unsupervised Learning* and *Reinforcement Learning*. Below an explicative image shows the differences between these branches which will be separately discussed later on.

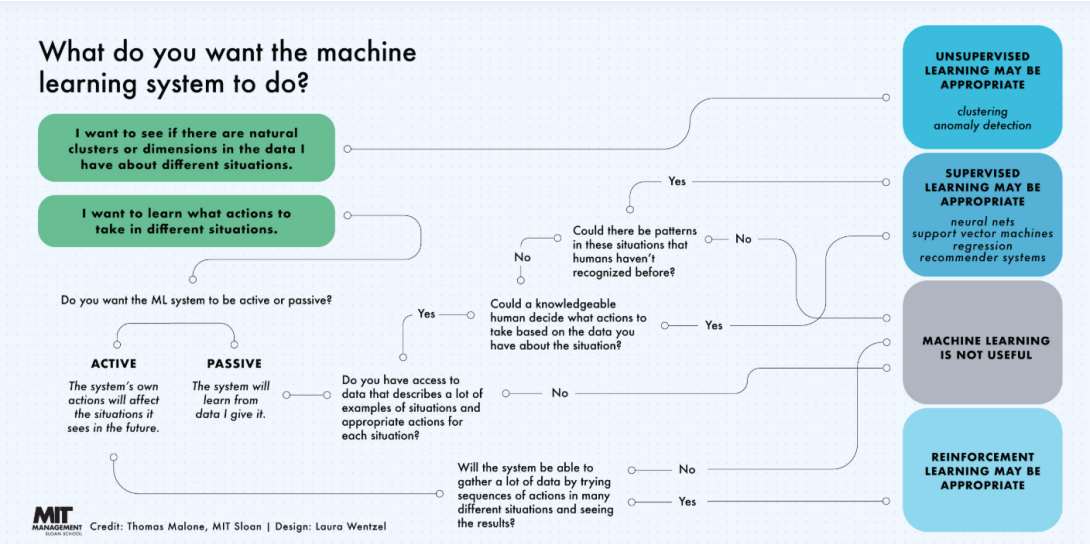


Figure 1.7: Machine Learning areas[11].

Supervised Learning

Supervised learning, also known as supervised machine learning, involves the use of labeled datasets to train algorithms to classify data or predict outcomes accurately. It works as the input data feeds the model itself, which optimizes the weights to make the data fit better in it. This occurs as part of the cross validation process to ensure that the model avoids overfitting or underfitting[10]. Supervised learning helps organizations solve a variety of real-world problems by giving large amount of labeled data to a model which will be used to classify or predict patterns.

Supervised Learning deals with two types of problems: classification problems and regression problems. This algorithm helps to predict a discrete value. For instance, processing the photos of the fruit dataset, each photo has been labeled as a mango, an apple, etc. Here, the algorithm has to classify the new images into any of these categories[12]. Examples of classification problems are: *Naive Bayes Classifier*, *Support Vector Machines* and *Logistic Regression*.

Unsupervised Learning

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets (subsets called clusters). These algorithms discover hidden patterns or data groupings without the need for human intervention. Unsupervised learning's ability to discover similarities and differences in information make it ideal for exploratory data analysis, cross-selling strategies, customer segmentation, and image and pattern recognition. It's also used to reduce the number of features in a model through the process of dimensionality reduction. *Principal component analysis* (PCA) and *singular value decomposition* (SVD) are two common approaches for this. Other algorithms used in unsupervised learning include neural networks, k-means clustering, and probabilistic clustering methods[10].

This learning algorithm is completely opposite to Supervised Learning. In short, there is no complete and clean labeled dataset in unsupervised learning. Unsupervised learning is self-organized learning. Its main aim is to explore the underlying patterns and predicts the output.

Reinforcement Learning

Reinforcement learning (RL), also known as reinforcement machine learning, does not require a vast amount of labeled data. In an RL model, the agent learns by interacting with the environment and receiving rewards for its actions. By recording each action and the corresponding reward, the agent can either *exploit* previously learned actions or *explore* alternative ones. A sequence of successful outcomes is reinforced to develop the optimal policy for a given problem. RL is different from both supervised and unsupervised learning, as its algorithms learn to respond to an environment autonomously. This field is rapidly evolving and continuously generating a wide range of learning algorithms.

1.3 Setup Testing

The use of *DDPC* and *Machine learning* is particularly reliable in fluid dynamics for applications such as drag reduction, control of movable surfaces, and disturbance reduction.

The current experiment aims to create and test an efficient *setup* that acquires data under specific conditions. Once all the setup, the acquisition and action time has been checked, it could be used as a tool for control method testing. In this specific case it will be used as a tool to validate a DDPC method to mitigate gusts based on a Reinforcement Learning algorithm. The setup involves a *Flapping Wing* which, subjected to *gusts*, acquires data for the *Shedding* phenomenon visualization in the wake. The main objective is to create an efficient and reliable structure, starting from the design through the printing and the assembly. This setup will also involve the electronics area, with the choice of all the sensors and boards to reach the mentioned purpose. Great importance will be given to the possibility of acquiring and processing vast amount of data while moving the flap using a *Servo Motor*. The final test will take place in an artificially generated gust situation, trying to minimize the *Shedding* phenomenon by *teaching* the flap how to move.

Over the years, several studies have been conducted on the use of *DDPC* methods in fluid dynamics, particularly to regulate movable surfaces of *flapping wings* under gust conditions[13]. The current experiment draws inspiration from a similar study, where an algorithm based on *Machine Learning* and an appropriate *setup* is used to maintain a constant value of *Lift*, measured through *Load Cells*.

In fluid dynamics, *Shedding* refers to an oscillating flow that occurs when a fluid, such as air or water, flows around a bluff body at certain velocities, which depend on the size and shape of the body. This phenomenon results in the formation of vortices behind the body that periodically *detach* from its surface. It is characterized by a dimensionless parameter known as the *Strouhal number* ($St = \frac{f_v D}{U_\infty}$)[14].

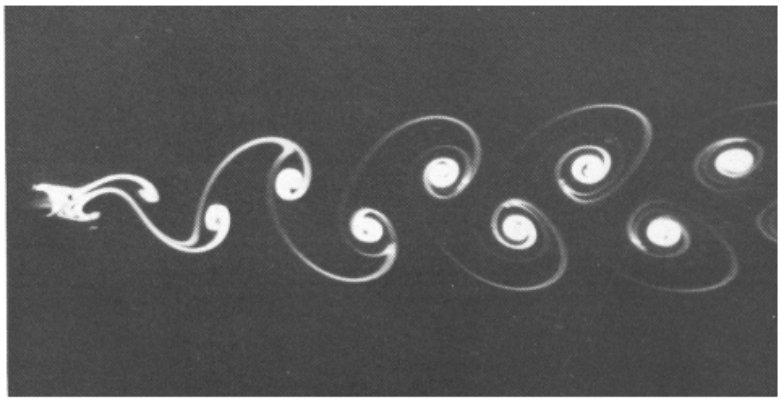


Figure 1.8: Vortex Shedding.

The *Vortex Shedding* phenomenon can lead to complications. If the frequency of vortex detachment matches the resonant frequency of the structure exposed to the airflow, resonance may occur. The structure will begin to vibrate with harmonic oscillations induced by the energy from the flow. Therefore, it is necessary to minimize the effects of this phenomenon, starting from damping the detachment of vortices themselves.

The goal of the current experiment is to build and test a suitable *setup* to acquire data as quickly as possible and, eventually, apply control to the movable surface, controlled by a *Servo Motor*. By putting in motion this surface, whose movement is determined by the flow conditions, the goal is to minimize vortex detachment in the wake. It must be said that this setup could be used for testing several control methods and this is only a particular case. In fact, the final setup is thought to measure pressure data on the profile surface and velocity in the wake region but gives the possibility to measure also frequencies on the surface by using microphones capsule and it allows sensors to be added due to its efficient external and internal design.

Chapter 2

Design & CAD

2.1 Introduction and Requirements

The *setup* process first involves the design of the *Flapping Wing* and the entire fixing and support system that will allow to conduct the experiment in gust conditions, ensuring its correct working.

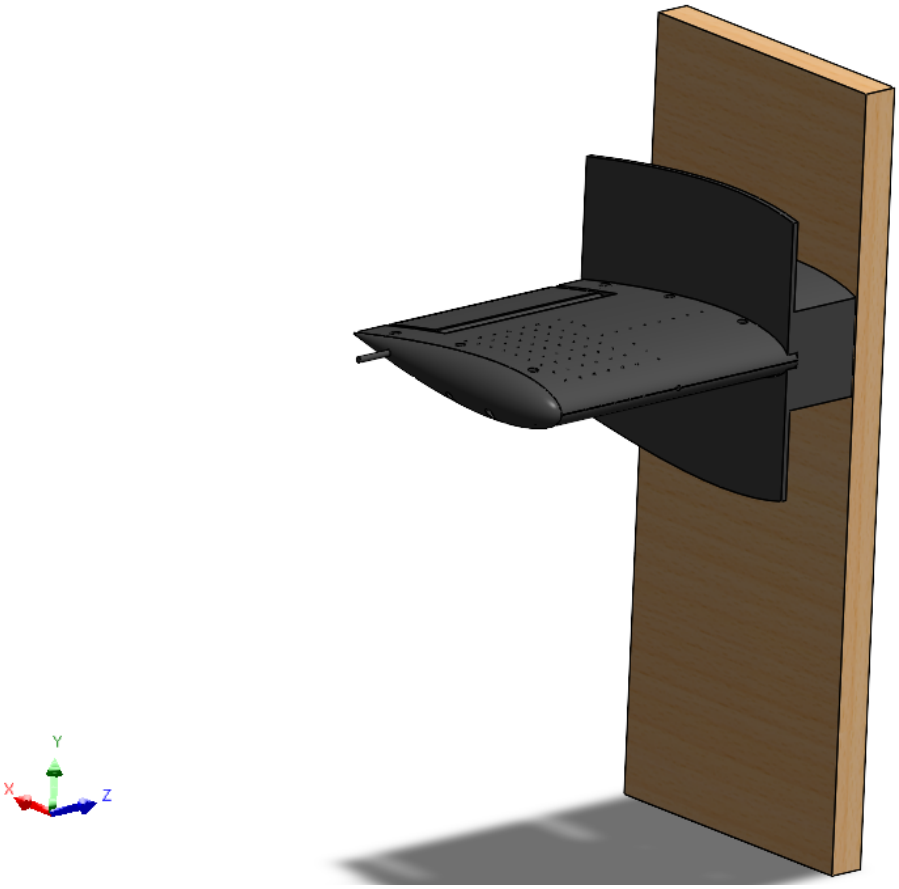


Figure 2.1: Rendering of the setup.

The structure is designed so that the wing is connected to to a support that guarantees a correct interface with a wooden *Beam*, which is fixed to a vertical column by a pin with adjustable inclination and vertical movement.
The design, and then the realization, of the *Flapping Wing* has to fulfill some requirements and specifications given by multiple factors, such as printing limitations on the component's dimensions, ease in the assemblage of the parts, adequate space and accommodation for sensors and electronic components, ensuring easy cable exit and the overall stability of the *setup*.

To fulfill the previously listed requirements, the structure was designed as a set of multiple parts:

- the top surface of the wing (**Sensing Side**);
- the bottom surface of the wing (**Non-Sensing Side**);
- the lateral component (**Side Part**);
- the moving flap (**Flap**);
- the lateral support (**Support**).

Through this approach, it is possible to build the entire *setup* in a reversible way, allowing an easy disassembly and reassembly.

Despite its printing limitations in terms of dimensions, a *Bambu Lab* 3D printer has been chosen. This 3D printer has a print bed size and height that do not allow the creation of very large parts, but it has a high printing speed. The print plate measures 25x25 cm, and the transverse dimension is also 25 cm. Regarding these constraints, the wing was designed with the following specifications:

***Flapping Wing* dimensions**	
Chord (c) [mm]	250
Wing Span ($\frac{b}{2}$) [mm]	247

Table 2.1: Wing Dimensions.

Therefore the structure design will involve the separate creation of each mentioned component. Then all the components will be printed. It is anticipated that some specifications will be changed during the process to optimize the structure as much as possible for the experiment.

The wing structure will initially be created as a single component, called the *Profile*. This component will then be split into two new parts. However, it is preferred to use the name *Profile* to refer to the entire portion of the wing (*Sensing Side* and *Non-Sensing Side*), which will remain fixed during the experiment. The *Flap*, which will change his incidence angle α, will be treated as a separated part from the *Profile*, so it's preferred to refer to it by the name *Flap*.

The term *Flapping Wing* will refer to the structure made up of *Sensing Side*, *Non-Sensing Side*, *Flap*, and *Side Part*.

The term *Complete* will refer to the structure made up of *Flapping Wing* and *Support*.

2.2 Profile

The design of the wing first requires the selection of a specific airfoil that best respects all the requirements. For the ultimate purpose of the experiment it is necessary to create a suitable internal space for the housing of sensors used to acquire data. Because of this necessity, the best choice would be that of an airfoil with a large trasversal dimension, such as the *NACA0018* airfoil. This symmetrical airfoil (the first digits being 00) has a maximum thickness of 18% of the chord length, a specification that makes it possible to create the internal insert without excessive space limitations. It is anticipated that, for this experiment, there is no need to develop a high lift value. The fixed part of the structure (*Sensing Side* and *Non-Sensing Side*), unlike the *Flap*, will not be subjected to changes in incidence α. Therefore, a symmetrical airfoil was preferred, which is suitable for this experiment due to its simplicity.

For the realization of the experiment, constant environmental conditions are expected, specifically a pressure close to atmospheric pressure at *sea level* and a temperature around $10°C$. The maximum flow velocity expected is 8 $\frac{m}{s}$. Below, the aerodynamic polar curves for a *NACA0018* airfoil are shown at a Reynolds number Re, whose value is calculated based on the given conditions:

$$Re = \frac{\rho_0 U L}{\mu} \approx 10^6$$

where:

- Density $\rho_0 = 1.225\ \frac{kg}{m^3}$;
- Dynamic viscosity $\mu = 1.770 \cdot 10^{-5}\ \frac{kg}{ms}$;
- Characteristic length (chord) $L = 0.247\ m$;
- Velocity $U \approx 8\ \frac{m}{s}$.

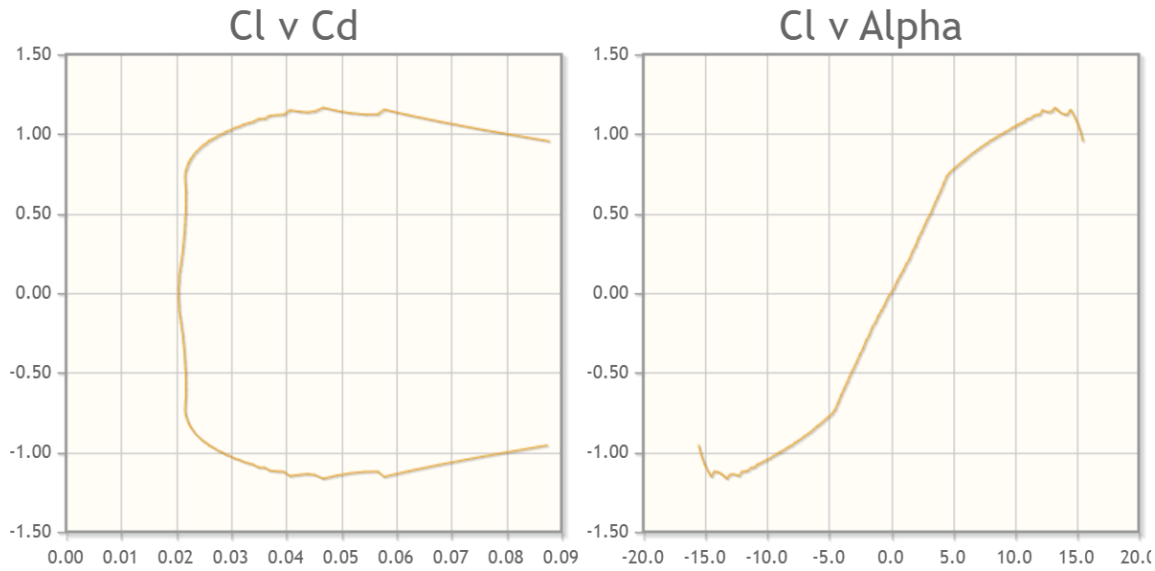


Figure 2.2: Aerodynamic Polars of $NACA0018$ at $Re10^6$[15].

Initially, the same length was set for the transverse dimension (*Span Wing*) and the longitudinal dimension (*Chord*), both equal to 250 mm, leading to a maximum thickness of 18% (45 mm). It is anticipated that, to achieve better printing results, the transverse dimension will be reduced by 3 mm.

Since two separate components have to be created starting from a single one (*Profile*),it is necessary to consider how to put together these parts (*Sensing Side* and *Non-Sensing Side*). It was decided to use threaded screws embedded in the structure in order to change the aerodynamic performance of the wing as little as possible. The same treatment will later be applied to the *Profile-Support* and *Support-Beam* interfaces, fixing these parts with threaded screws. This choice, therefore, implies creating holes for screw

insertion.

It is anticipated that the *Profile-Flap* interface is handled by the shaft around which the *Flap* is allowed to move. This shaft (*Shaft*), passing through the *Profile* structure, keeps the rotation axis fixed .

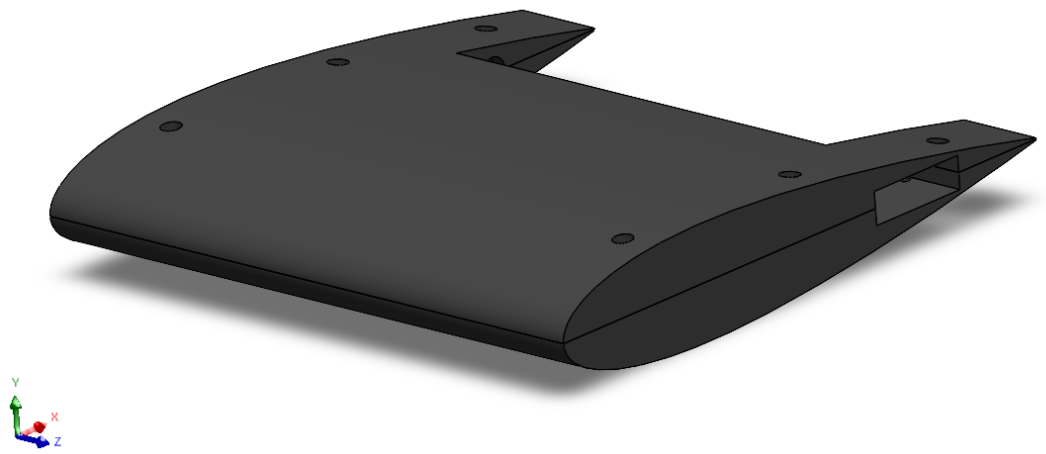


Figure 2.3: Profile.

The entire wing is created by extruding a *NACA0018* airfoil, created by importing the points that define its curve[15]. Then, a portion of the structure is cut from the extruded solid to insert the movable flap, rounding the edges to allow smooth rotation of the component without interference with the fixed part. An insert for the *Servo Motor* is also created to perfectly align its axis with the *Flap*'s axis, which is located precisely in its aerodynamic center.

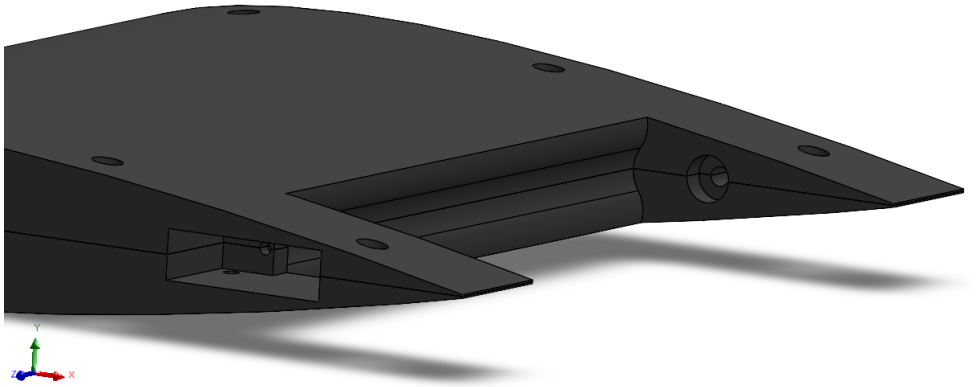


Figure 2.4: *Detail*: Servo Motor Insert.

Once the structure is complete, the wing is divided into two halves to allow to work on the top and bottom parts separately. It is specified that the two components composing the fixed part of the wing (*Profile*) are designed asymmetrically, due to the presence of inserts for microphones, the need for an eventual *Breadboard* housing, and various pressure taps on the top surface of the airfoil. However, to allow for proper operation of each part, the *Profile* is *cut* in its symmetry plane. It is anticipated that, in addition to the screw and threaded insert, a centering system will be required for the two components. This system will allow a maximum tolerance of 0.5 mm between the parts.

Below the *Profile* and its related dimensions:

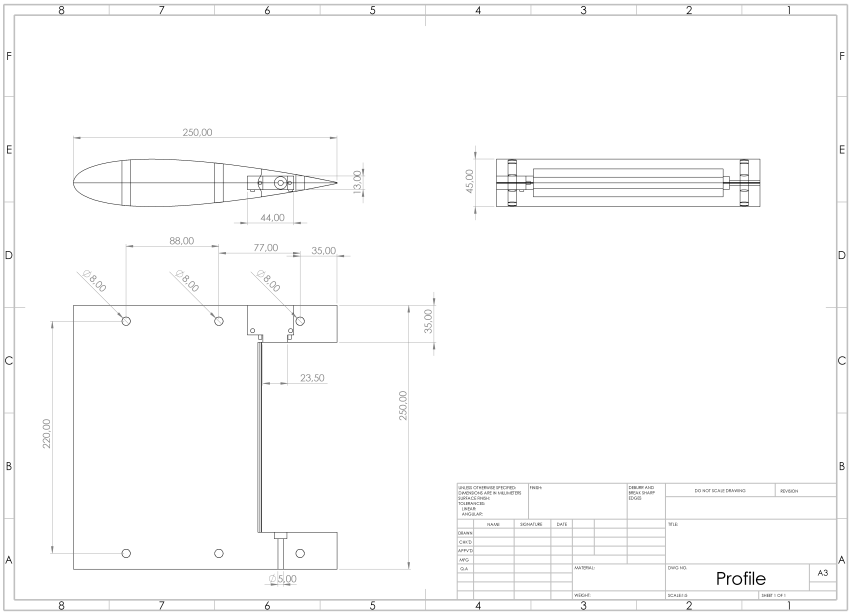


Figure 2.5: *Profile.*

2.3 Sensing Side

Once the upper part of the *Profile* is obtained, the position and available space for the cable compartment, where the *Breadboard* and its associated sensors will be housed, must be analyzed. Additionally, inserts need to be created for potential microphone capsules, from which further data can be acquired. Furthermore, it is essential to position the various static pressure taps on the wing surface, which will allow for the selection of the most suitable location from which to gather data.

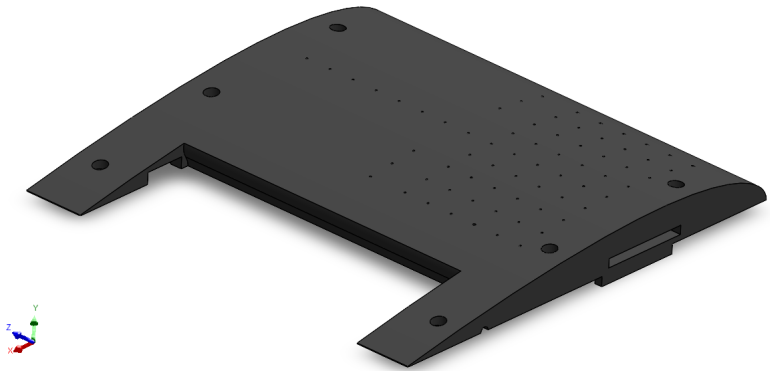


Figure 2.6: Surface of *Sensing Side*.

Initially, four pressure taps are created and positioned along the symmetry axis of the surface, along the chord. These taps, through four respective ducts, allow the pressure signal to be acquired from outside the structure, particularly on the lateral side of the *Flapping Wing*. All the other pressure taps end their respective ducts inside the wing insert, where the *BreadBoard* and all data acquisition system will be positioned. Each static pressure tap is made as a surface hole with a diameter of 1 mm, ensuring minimal interference with the airflow. From these holes, ducts with a diameter of 2.4 mm extend. These ducts are deliberately enlarged compared to the surface holes because they need to accommodate small, rigid stainless steel tubes, to which plastic tubes of similar diameter will be connected, transmitting the pressure signal to the corresponding sensors.

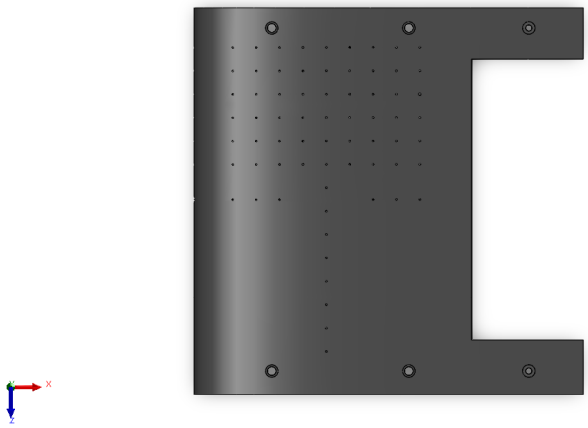


Figure 2.7: *Detail*: Arrangement of pressure taps.

Regarding the microphones, two inserts are created with dimensions matching the capsules (*CMA-4544PF-W*), ensuring minimal tolerance and preventing vertical sliding. These housings are positioned upstream and downstream the *Profile* maximum thickness position, towards the *Leading Edge* and *Trailing Edge*. Again, the holes on the surface have the same diameter as the static pressure taps (1 mm). It should be noted that, to obtain a correct measurement of static pressure and frequency using the microphones, it is necessary to create holes with an axis that is normal to the local surface of the profile. Therefore, the previously mentioned 1 mm holes will be made regarding this consideration. The two inserts for the

microphone capsules also include two ducts of sufficient diameter to house the cables that will carry the signal outside the wing for eventual data acquisition. These ducts are made exactly on the symmetry plane of the *Profile* so, once the parts (*Sensing Side* and *Non-Sensing Side*) will be fixed, they will act as a circular duct.

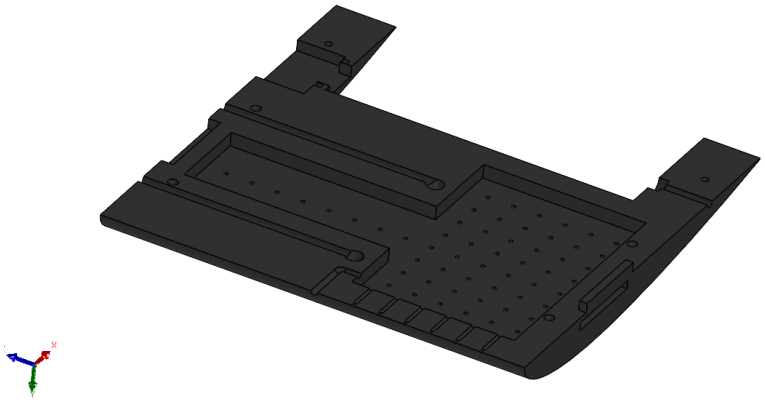


Figure 2.8: Interior of *Sensing Side*.

Considering the necessity to fix the component to the *Non-Sensing Side* part, there are 4 lateral holes, two toward the *Leading Edge* and two halfway along the chord. These holes are designed as far from the center of the wing as possible to minimize impact on aerodynamic performance. These holes have a sufficiently large diameter to accommodate *M4* screws, which will be embedded into the surface. The two lateral holes near the *Trailing Edge* present smaller diameters, due to the limited space in the thickness direction, allowing easy attachment with *M3* screws and their embedding.

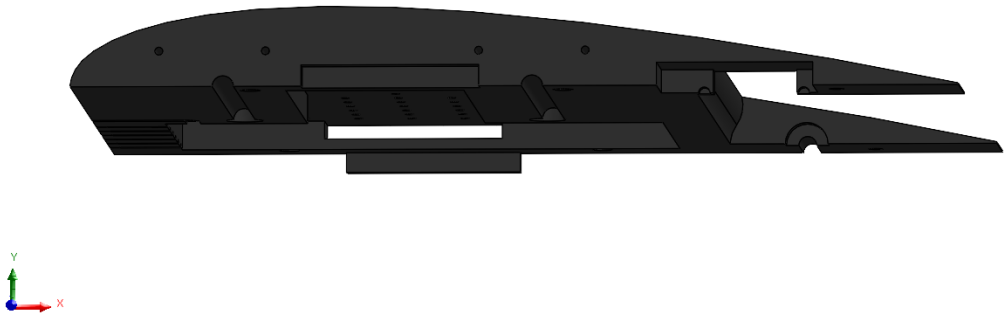


Figure 2.9: *Detail*: Centering system.

It is taken into account the previously mentioned centering system, consisting in an insert and an extension on both sides of the profile, which will match with corresponding extension and insert on the *Non-Sensing Side* component. This centering system guarantees a tolerance of 0.5 mm.

Lastly, designed for potential use, there are several total pressure taps positioned exactly on the front plane of the profile. These taps are created as semicircular ducts, and once the upper and lower parts are fixed and aligned, they will allow for their use and ensure accurate measurements. All total pressure ports terminate their respective ducts in the internal chamber of the structure.

Below, the *Sensing Side* part and its relevant dimensions:

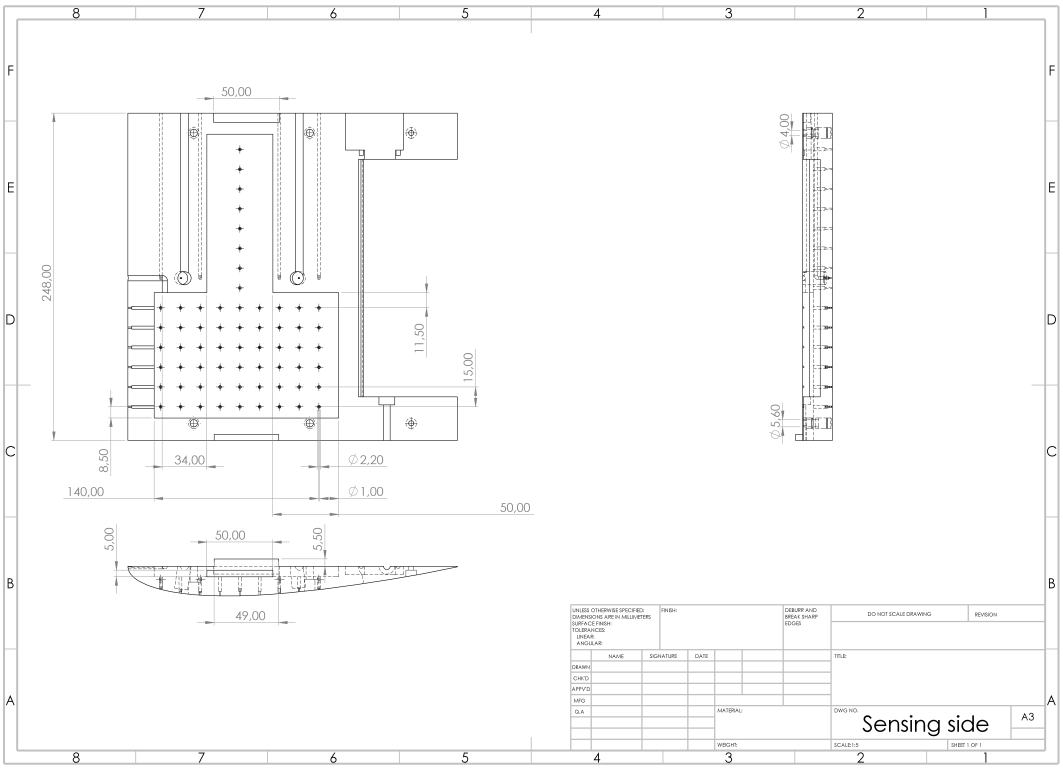


Figure 2.10: *Sensing Side.*

2.4 Non-Sensing Side

The lower part of the *Profile* shows an insert of the same dimensions as the one made inside *Sensing Side*, so that the edges would match perfectly, ensuring an airfoil *closure*. Another insert of this type is created for the placement of the *BreadBoard*, ensuring that the sensors and control module do not interfere with the metal ducts embedded in the pressure taps ducts in the upper part. This additional insert is created inside the larger one, and therefore will have smaller dimensions. The insert's depth is limited by the thickness of *Non-Sensing Side*.

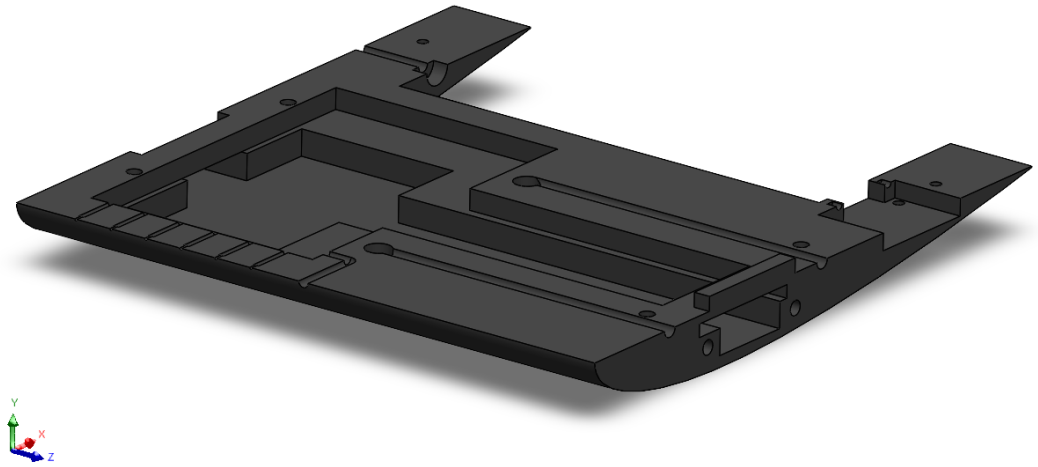


Figure 2.11: Interior of *Non-Sensing Side*.

The lateral holes for cable passage are also much larger to avoid any obstruction during their passage. As in the case of the upper part, the lateral centering system shows a 0.5 *mm* tolerance.

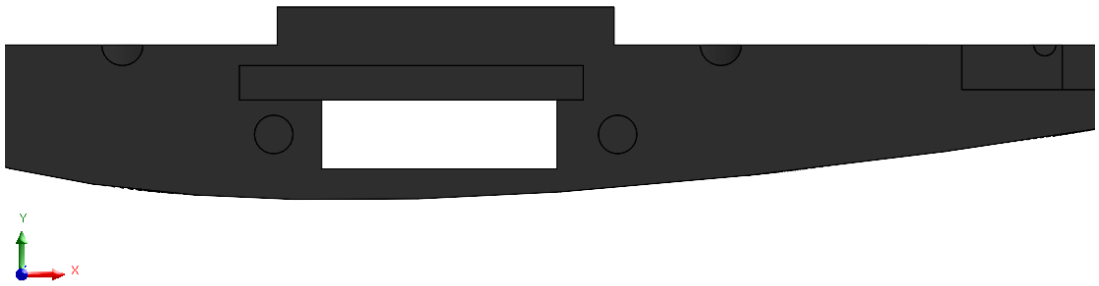


Figure 2.12: *Detail*: Holes for cable passage and centering system.

Threaded *Ruthex* rods will be inserted to allow the screws to fit in every screw's insert made in the *Non-Sensing Side* part. These components, as mentioned, will be inserted by locally heating the material that will form the structure to allow their insertion and locking. By using this method, the *Sensing Side* and *Non-Sensing Side* components will be able to be locked by inserting the screws from the *Sensing Side* side. As mentioned earlier, the surface of *Non-Sensing Side* does not include the insertion of static pressure taps.

Figure 2.13: Surface of *Non-Sensing Side*.

Finally, there are two $M4$ holes placed laterally, which will allow the insertion of threaded rods for fastening the entire structure to the *Support* and, consequently, to the *Beam*. These same holes are also created at the other side of the component, to fix it to the *Side Part*.

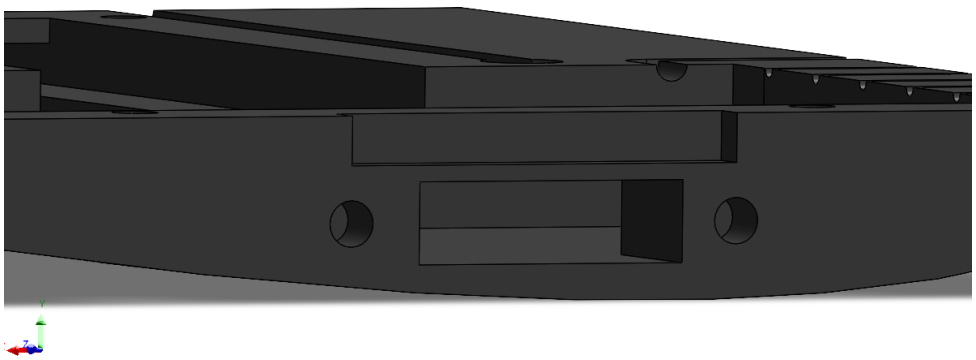


Figure 2.14: *Detail*: Holes for fastening *Support*.

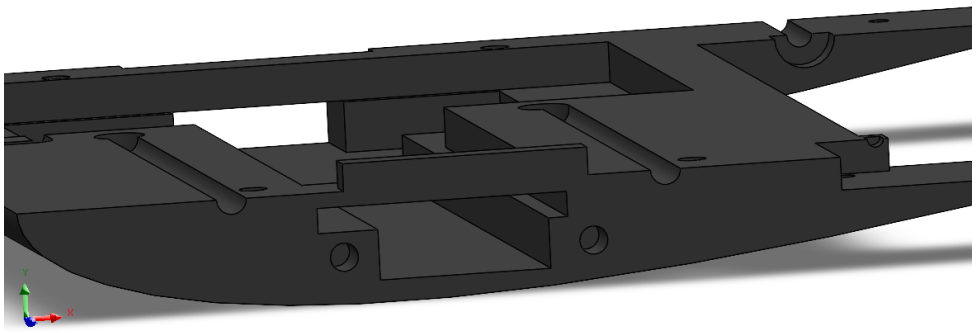


Figure 2.15: *Detail*: Holes for fastening *Side Part*.

Below, the *Non-Sensing Side* part and its dimensions:

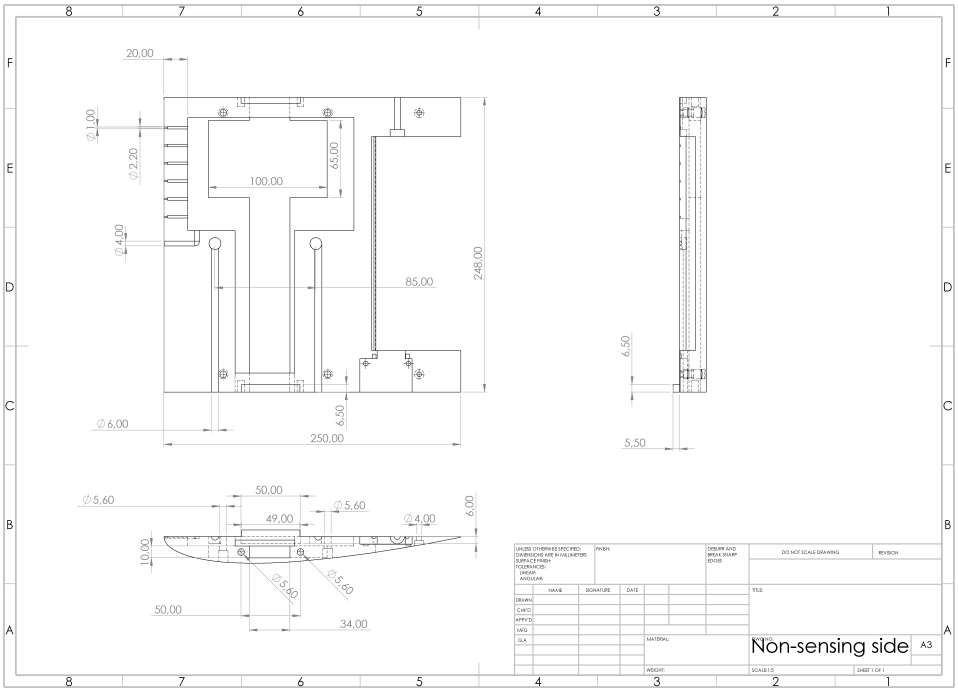


Figure 2.16: *Non-Sensing Side*.

2.5 Flap

The *Flap* is made by extruding a *NACA0018* profile, just like the structure of the *Profile*. This choice was made to ensure continuity between the fixed wing surface and the moving flap, minimizing any changes in aerodynamic characteristic.

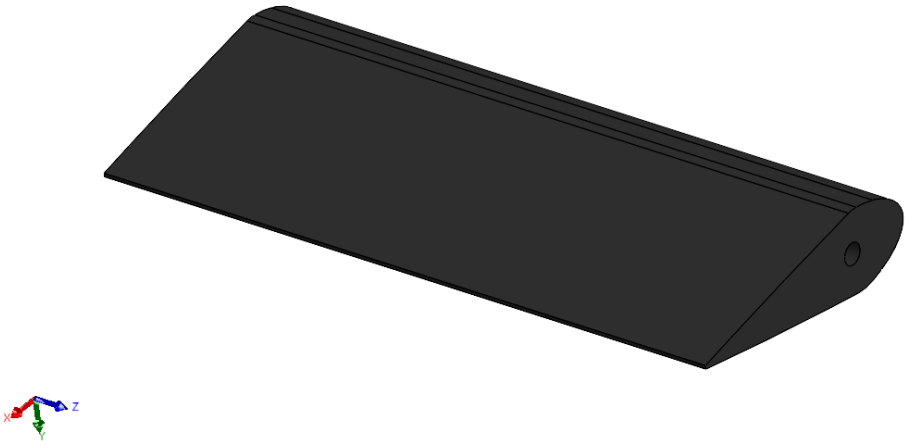


Figure 2.17: Flap.

The *Leading Edge* is specially modified to allow the *Flap* to move freely while avoiding contact with the fixed part of the wing, which has also been adapted accordingly. These modifications also aim to maintain the continuity of the wing surface as much as possible, avoiding aerodynamic issues and keeping the flow as *clean* as possible for accurate measurements.

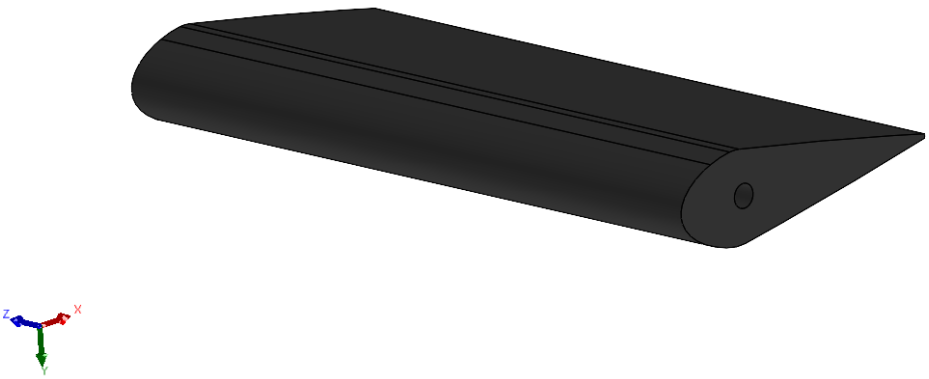


Figure 2.18: Flap.

At the first quarter of the chord, there is a through-hole, 5 *mm* in diameter, into which a shaft will be inserted to allow the interface between the component and the fixed structure (*Profile*). This coupling ensures minimal tolerance, so that the shaft can keep the *Flap* axis as stable as possible. The hole position is not casual. This insert will be placed exactly at the *front quarter* of the chord of the profile, where the aerodynamic center is placed. All the aerodynamic resultant forces on the profile act in this point.

Below, the *Flap* and its dimensions:

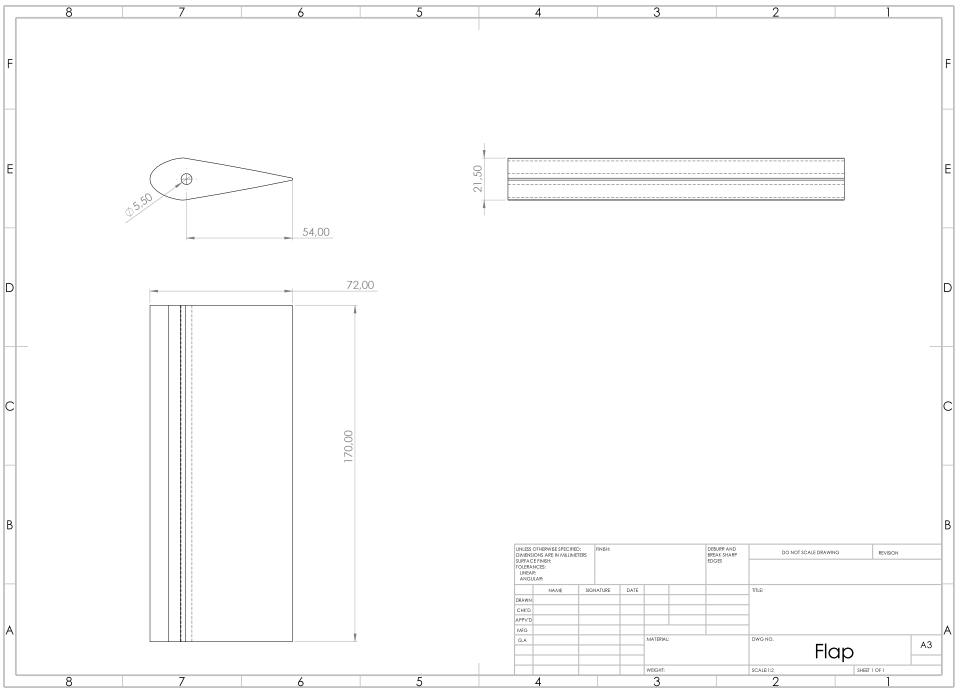


Figure 2.19: *Flap.*

2.6 Side Part

A *Side Part* is to be created, which will be fixed on the side of the *Flapping Wing* opposite to the *Beam*. This component is designed with the only purpose of minimizing any aerodynamic issues for the entire wing structure. It is built as a lateral extension of the structure, rounded and without edges, in order to prevent the formation of vortices or similar phenomena at the tip, which could, even of a small entity, affect the measurements taken at the center of the surface.

Figure 2.20: Side Part.

During the assembly, this component is fixed to the *Non-Sensing Side* part through the insertion of two $M4$ screws. A hole of the same diameter as the shaft, enlarged by a small *offset* of a few millimeters, is made at the trailing edge to allow a smooth passage for the *Shaft*.

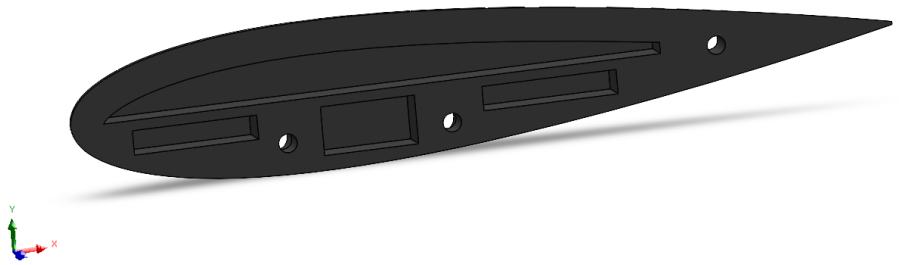


Figure 2.21: *Detail*: inside the Side Part.

Inside, the *Side Part* shows hollow inserts, made simply to facilitate $3D$ printing and create a lighter component.

Below, the *Side Part* and its dimensions:

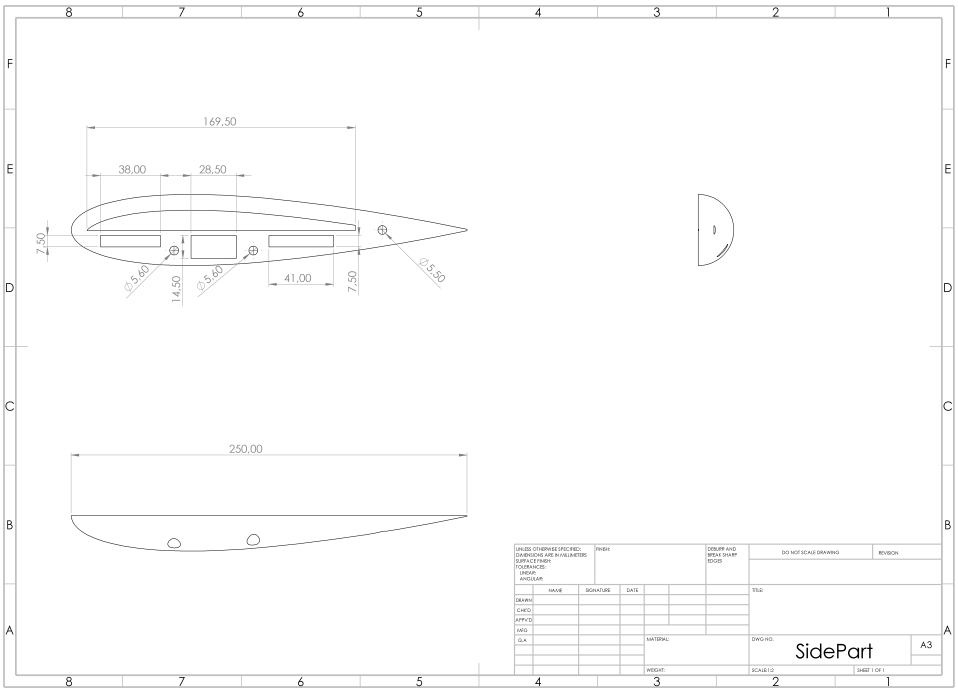


Figure 2.22: *Side Part.*

2.7 Support

The idea behind the lateral *Support* is to ensure a correct and stable *setup* so that the experiment can be carried out without any structural issues that could compromise the measurements. The *Flapping Wing* is attached to a wooden beam (*Beam*), which is fixed to a column by an adjustable pin. The attachment of the *Flapping Wing* to the *Beam* is accomplished through the design and the use of a support (*Support*) that behaves both as a clamp for the wing and as a *Side Plate* to prevent tip vortex problems by simulating the presence of a wall.

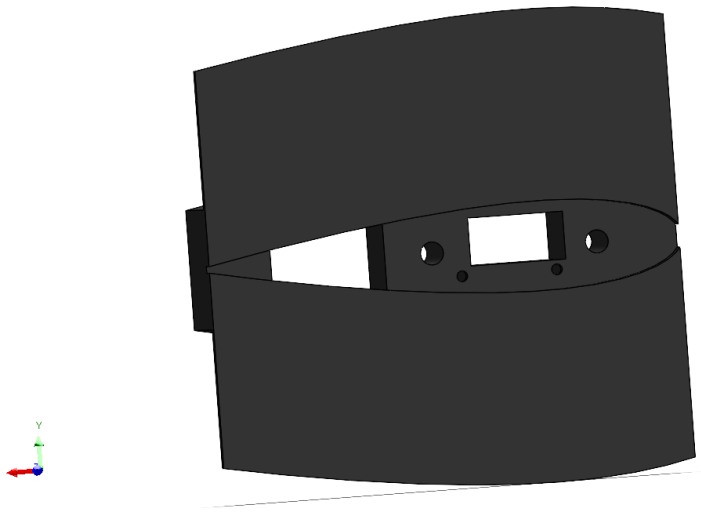


Figure 2.23: *Detail*: Side Plate.

It should be noted that, to avoid overly complicating the structure, a second *Side Plate* is not added on the side of the *Profile* opposite from the *Support*, accepting an approximation for measurements very close to the wingtip. At this side, efforts are still made to avoid significant issues by incorporating the previously mentioned *Side Part*.

The support is fixed to the *Beam* with six $M5$ screws to minimize movement, which could cause changes in the angle of attack of the profile, thus affecting the measurements. The *Flapping Wing* is attached through threaded rods, previously mentioned, which pass through ducts in the support and are directly secured to the *Beam*. Due to printing limitations, it is only possible to print a relatively small surface for the *Side Plate*, but it still covers the entire lateral surface of the wing.

Additionally, ducts are created to let the cables that, starting from inside the *Flapping Wing*, allow to process signals from outside, pass. These conduits are essential for the eventual data acquisition from microphones and pressure sensors, and especially to control the movement of the *Servo Motor*, governed by an external module, ensuring continuity with the lateral holes of the *Profile*.

For a proper assembly and minimal interference, this support acts as a *clamper* for the *Flapping Wing*, effectively reducing its transverse dimension by 10 mm. Between the *Support* and the *Flapping Wing* is provided a 1 mm gap.

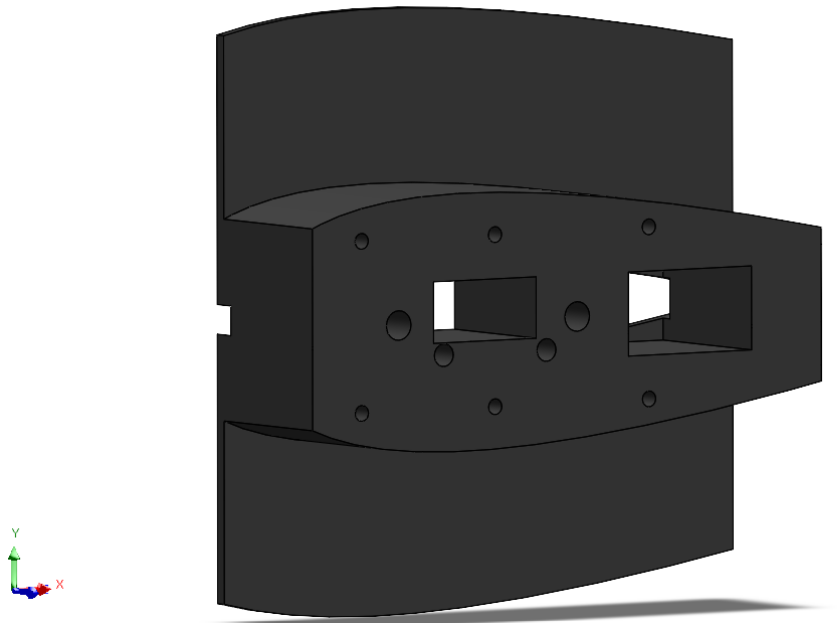


Figure 2.24: Support.

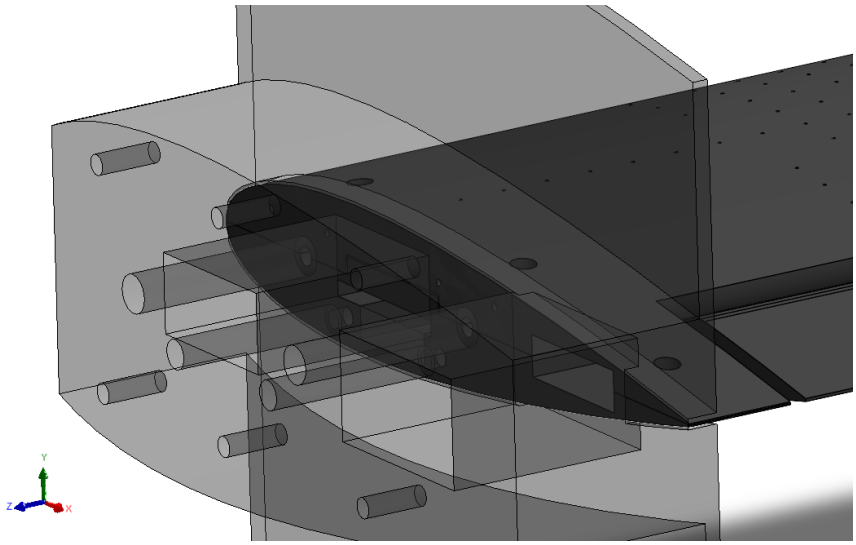


Figure 2.25: *Detail*: *Support-Profile* interface.

Below, the *Support* and its dimensions:

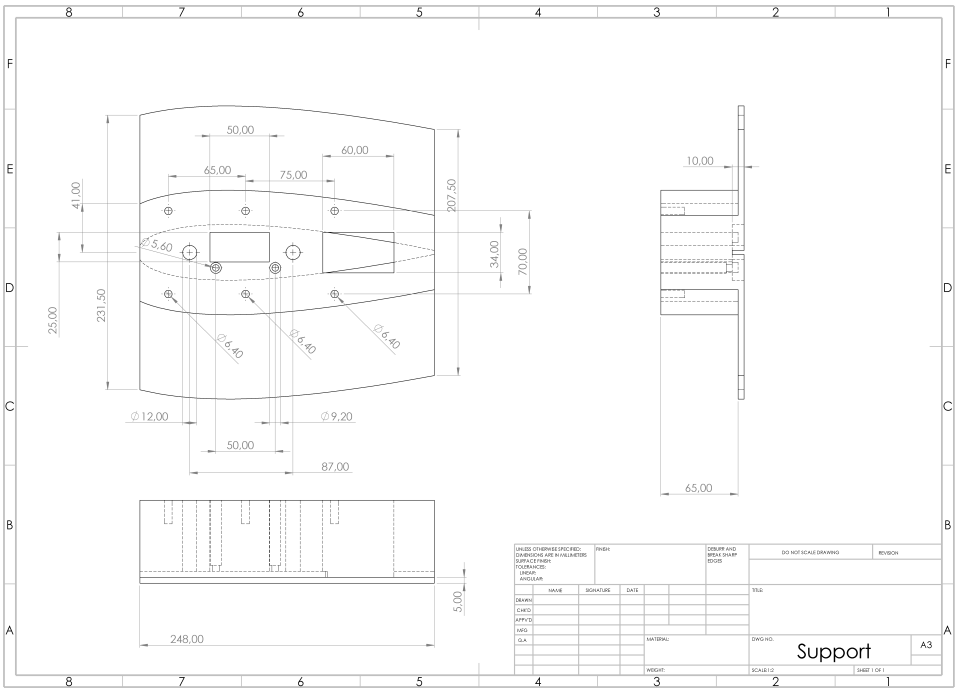


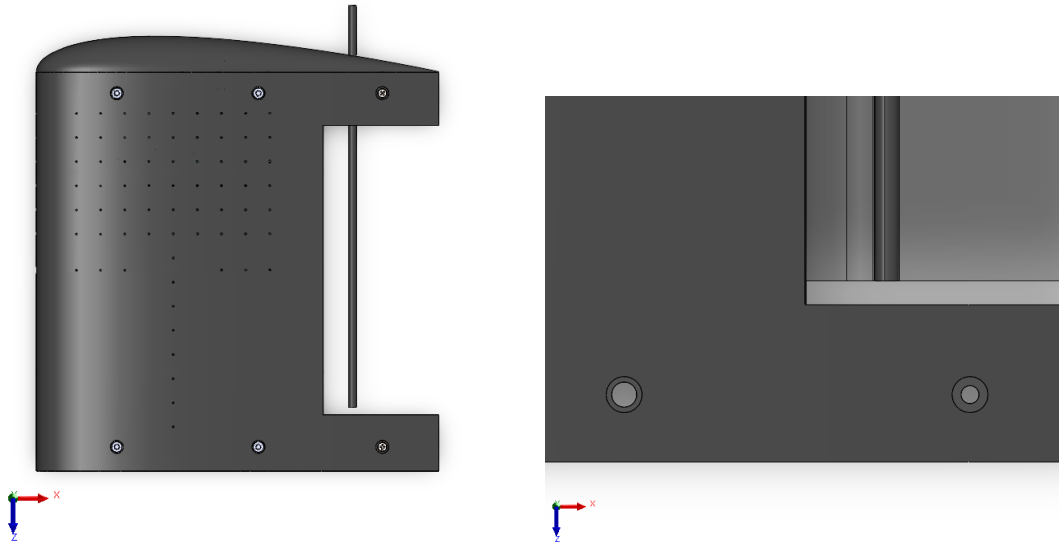
Figure 2.26: *Support.*

2.8 Components

2.8.1 Shaft

The *Shaft* is made by a cylindrical body with a diameter of 5 *mm*, so that it fits into the housing hole of the *Profile*, minimizing tolerance and preventing sliding. This component is not manufactured using *3D Printing* but is supplied as a pre-made component. Despite its length being much greater than the *Span Wing* length, it is decided not to cut off the excess part to facilitate its insertion (removal) during assembly (disassembly).

It is important to note that the function of the *Shaft* is purely structural. This component ensures that the rotation axis of the fixed *Flap* is maintained and ensures the *Flap-Profile* interface as it is inserted into the wing structure. In fact, it should be noted that the *Servo Motor* controlling the movement of the *Flap* is connected to it through a plastic joint glued to the left side of the *Flap*. Therefore, the shaft will not rotate: by remaining fixed, it prevents the translation of the moving *Flap*.



(a) Shaft.

(b) *Detail*: *Shaft-Profile* interface.

It is reminded that this component is pre-manufactured, however, to understand its dimensions, these are shown below.

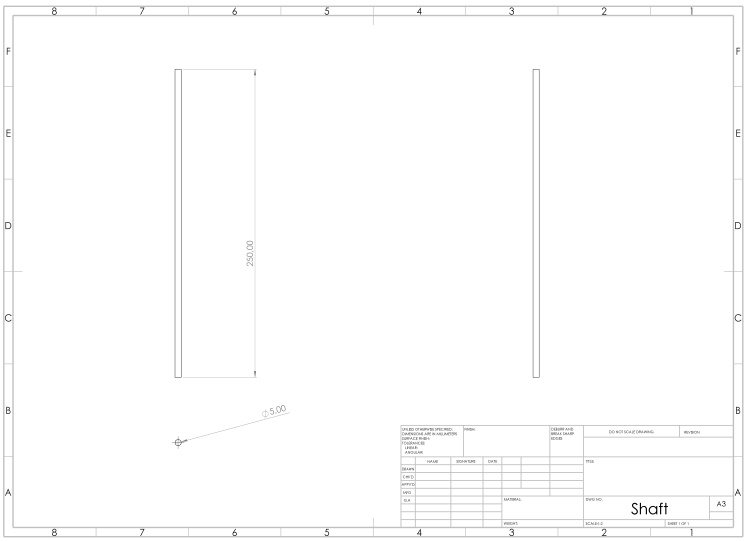


Figure 2.27: *Shaft.*

2.8.2 Beam

The *Beam* to which the *Support* is fixed is simply a plywood beam with appropriate holes made for the passage of screws and threaded rods, which ensure the fastening of the structure. Holes are also made to ensure continuity with the ducts of the *Support*, through which cables coming from inside the *Profile* (in particular from the sensors, the *BreadBoard*, and the *Servo Motor*) pass. Further holes are also made on the *Beam* for the fastening of an adjustable pin, which is also fixed to a vertical column.

This component, like the *Shaft*, is supplied as pre-made. However, as well as for the previous component, the dimensions are shown.

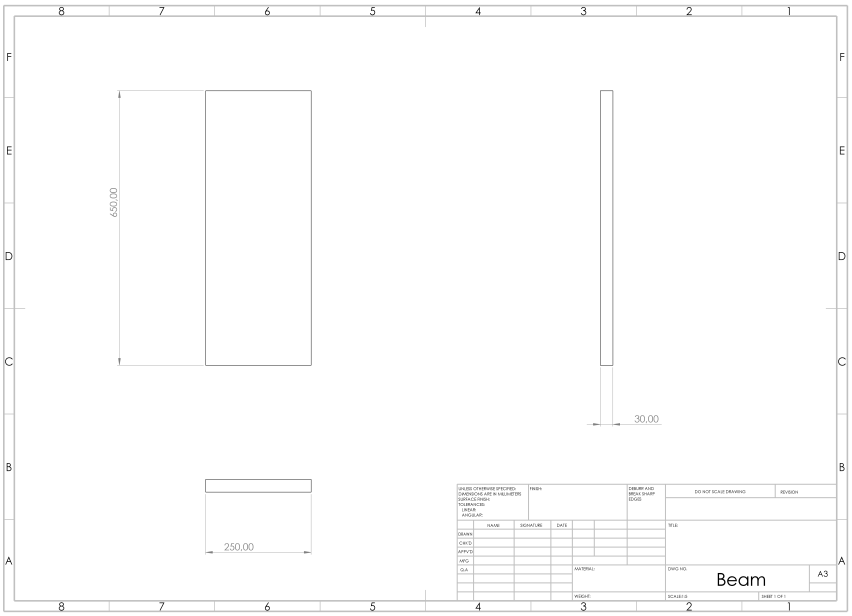


Figure 2.28: *Beam.*

2.9 Complete

Below it is shown a detailed view of the complete structure from both outside and inside.

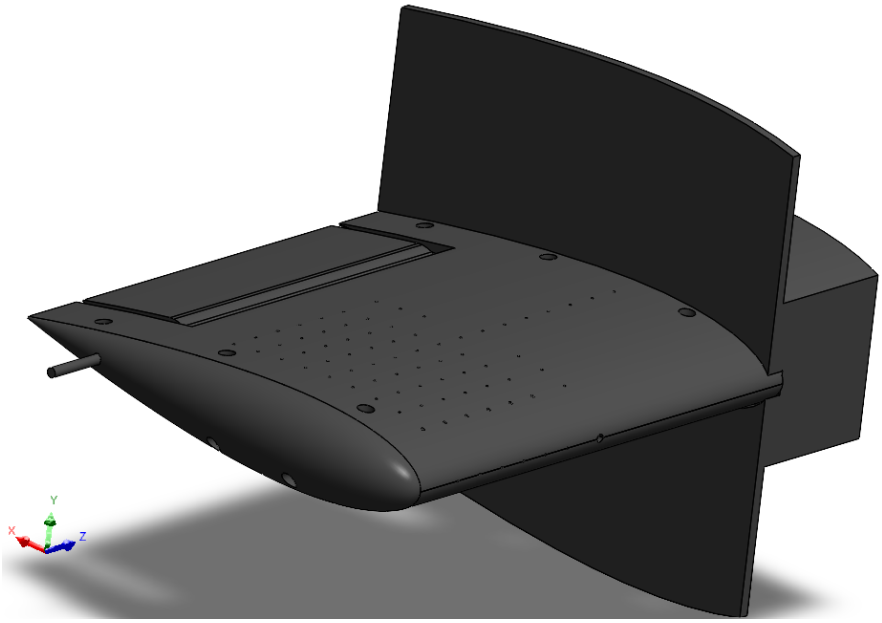


Figure 2.29: Setup from outside: Right Side.

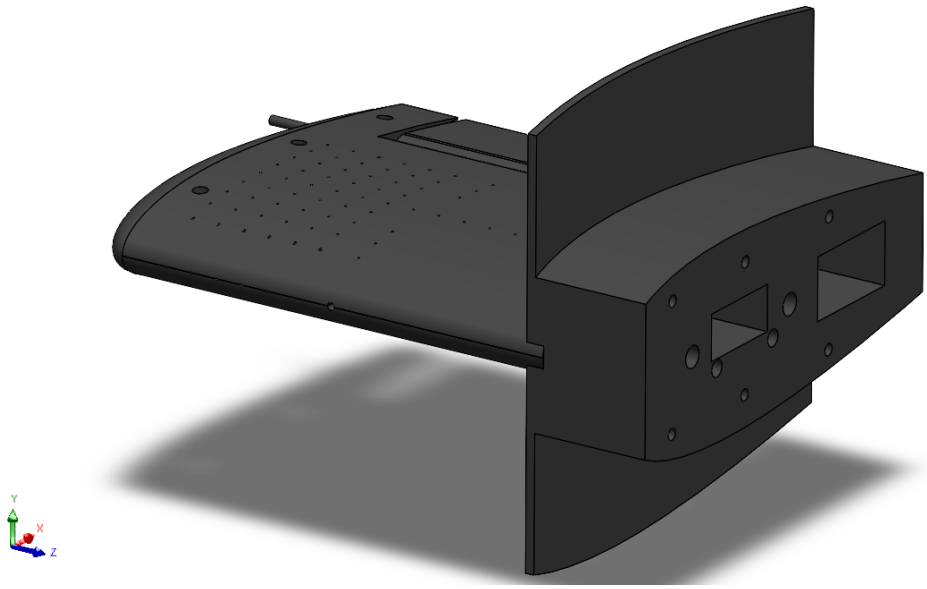


Figure 2.30: Setup from inside: Left Side.

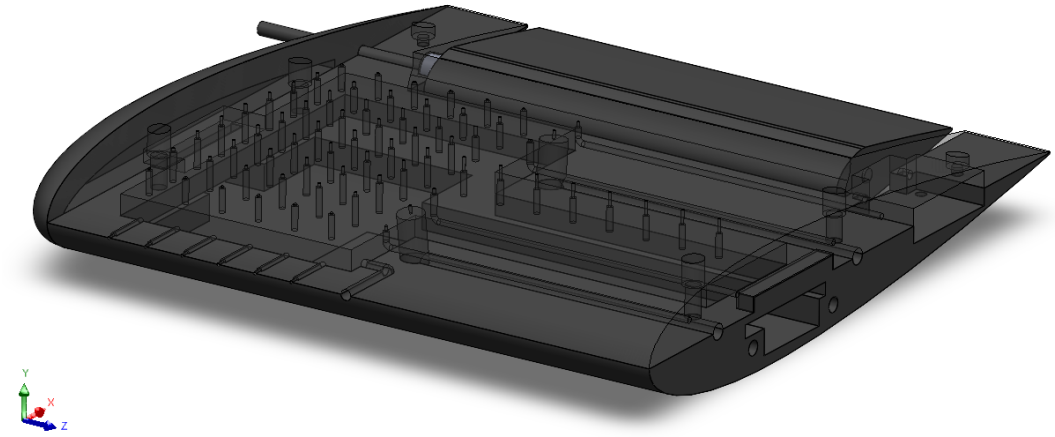


Figure 2.31: *Flapping Wing*: *Sensing Side* in transparency.

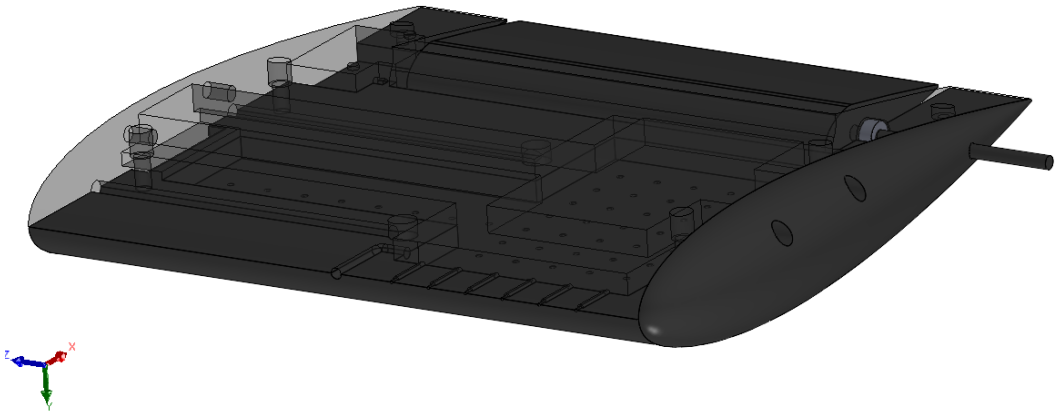


Figure 2.32: *Flapping Wing*: *Non-Sensing Side* in transparency.

Maximum precision is required in the manufacturing of the internal insert in order to achieve a hermetic closure for the various ducts and the housing for the *BreadBoard*. The same precision should also be applied in checking the holes for the fastening screws: it is necessary to ensure that the continuity of the holes from the top surface to the bottom surface is respected.

Chapter 3

Setup & Experimental Design

3.1 3D Printing & Assembly

3D Print

After the design phase, the next step is the 3D printing of the various components previously mentioned. The process takes place using a $3D$ printer. It will be used a *Bambu Lab* $3D$ printer, with the specifications mentioned in the previous chapter. Regarding the choice of the material, *PLA* is selected as a durable material that is also suitable for any post-process after printing. As mentioned earlier, the *Sensing side* and *Non-sensing side* components will pass through a reduction of 3 mm along the *Span Wing* direction to allow the printing of a *raft* layer, ensuring the part does not detach from the print plate, which could cause defects and affect performance. This layer will have a thickness of 1 mm. Therefore, due to the *Bambu Lab* printing dimension limits, it is preferred to proceed with the 3 mm cut mentioned.

Ruthex **inserting**

Once the individual components are obtained, the actual assembly of the structure begins, starting with the insertion of *Ruthex* rods into the corresponding inserts of *Non-sensing side* to allow the connection through screws to *Sensing side*.

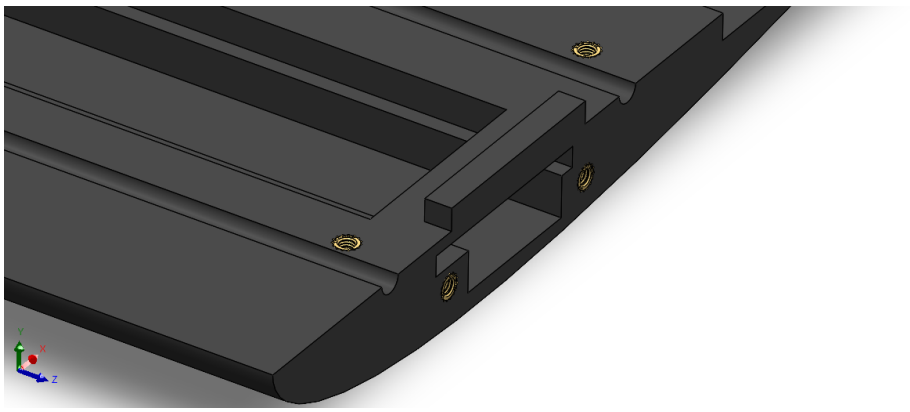
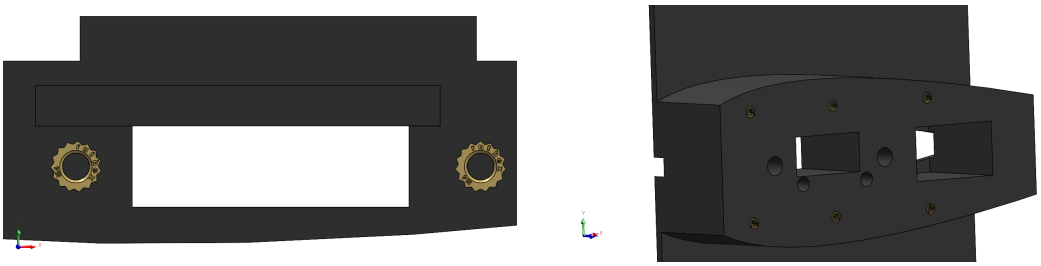


Figure 3.1: *Detail*: insertion of *Ruthex* in *Non-sensing side*.

The insertion of these components is done, as mentioned earlier, by locally heating the *PLA* to allow them to fit. The holes previously made are designed to match the *Ruthex* without any tolerance, making the insertion easier. Once cooled, the *PLA* near the hole will return to its original properties as its hardness. It is important to note that the size of the *Ruthex* follows the size of the designed holes and consequently the size of the screws to be used. Particularly, the two central holes and the two closer to the *Leading Edge* will be $M4$ type, while the two closer to the *Trailing Edge* will be $M3$ type. The *Ruthex* must be completely embedded in their respective holes, so to not create any offset between the surfaces that will be connected.

Similarly, the insertion of additional *Ruthex M4* components takes place at the side of the *Flapping Wing* to allow the fastening of the *Support* and the *Side Part*. The same connection method is planned for the *Support-Beam* interface, so a $M5$ type screw is inserted in the designated slots of *Support*.



(a) *Detail*: insertion of *Ruthex* in *Non-sensing side*.

(b) *Detail*: insertion of *Ruthex* in *Support*.

Figure 3.2: Inserting *Ruthex*.

Mounting the Support-Beam

Once all the *Ruthex* components are inserted and the possibility of fastening the parts is ensured , the *Support* is attached to the *Beam* by inserting six $M5$ type threaded screws, spaced accordingly to prevent any movement of the component which would result in changes in the angle of incidence of the *Flapping Wing*.

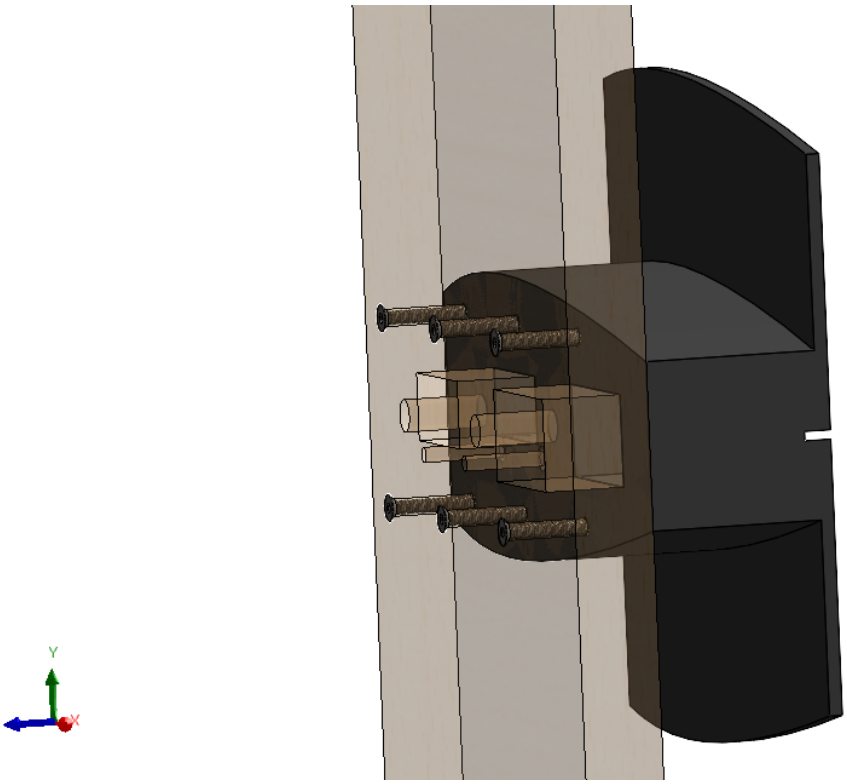


Figure 3.3: *Beam-Support* interface.

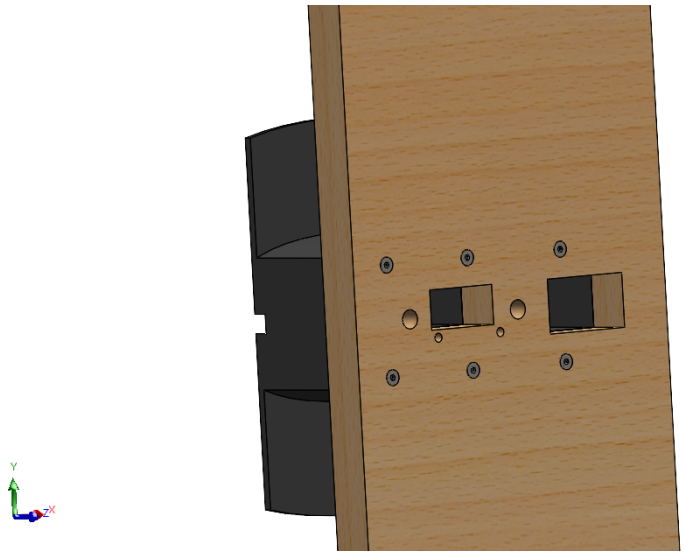


Figure 3.4: *Detail*: screws for fastening the *Support*.

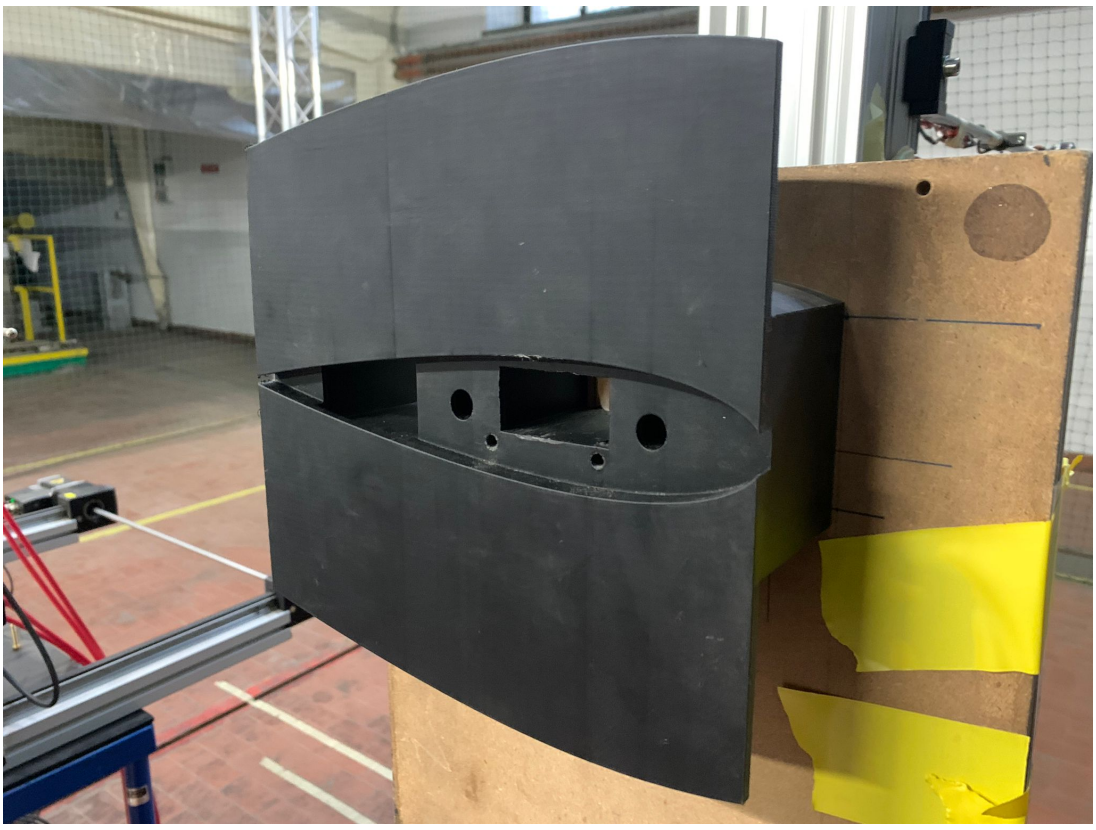


Figure 3.5: *Support*.

INOX Tubes and *Servo Motor* inserting

Before inserting the *BreadBoard* and the sensors inside the *Flapping Wing*, it is important to place stainless steel tubes in the holes inside *Sensing side* where the pressure signals will be acquired. This step is crucial to ensure the continuity of the ducts between the pressure taps and the sensors. These rigid tubes will then be connected to plastic pipes that will carry the signal to the pressure transducers. It is anticipated that pressure signals will be acquired from two pressure taps only, both located along the chord in the middle of the wing surface.

Then, the *Servo Motor* is installed in one of the two halves that will make up his own insert. It is preferred to fix the *Servo Motor* to the half-insert inside *Sensing side*. It is necessary to fix the component to the structure using *hot glue* to avoid detachment during the experiment, which would compromise the entire measurement. This bonding technique is preferred for its effectiveness and the ability to remove the component without damaging *Sensing side* too much.

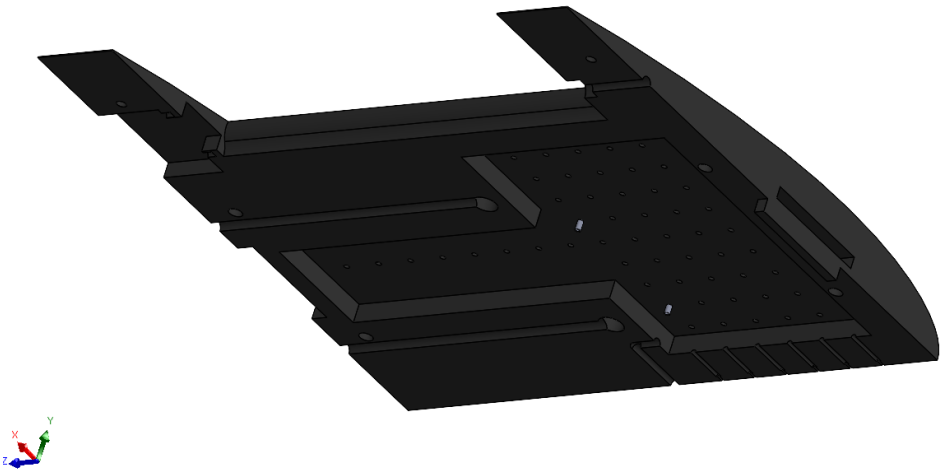


Figure 3.6: INOX Tubes in *Sensing side*.

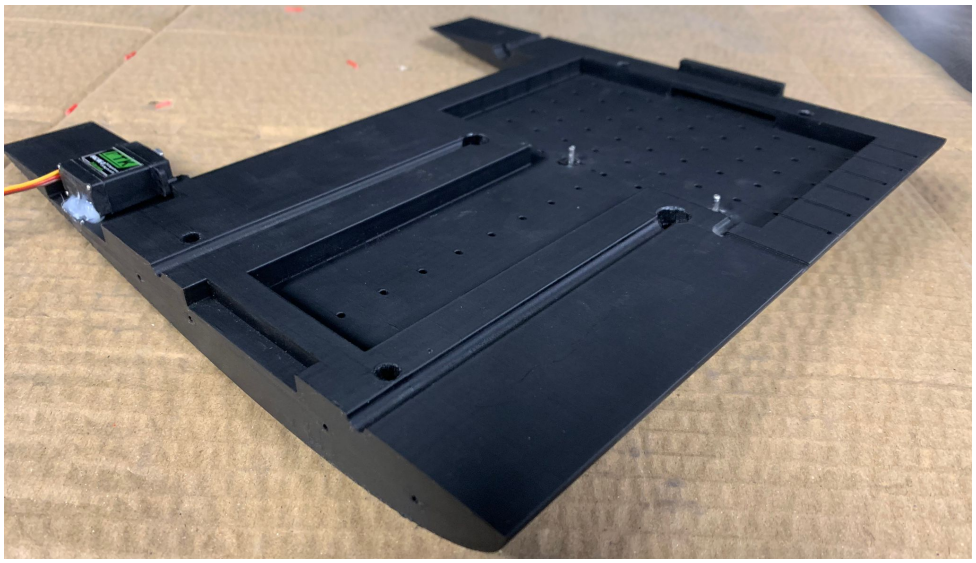


Figure 3.7: *Sensing side*.

Mounting the *Profile*

Once the final configuration of *Sensing side* is obtained, the *Breadboard* is inserted and bonded to the structure in the designated slot, by using glue and the sensors with the control module are properly arranged on it. This step will be discussed further. Then, the pressure sensors are connected to the stainless steel tubes using plastic pipes. Additionally, the *USB Type-C* cable connected to the board and the two tubes for the pressure sensors are routed outside the structure through the lateral holes of *Non-sensing side*.

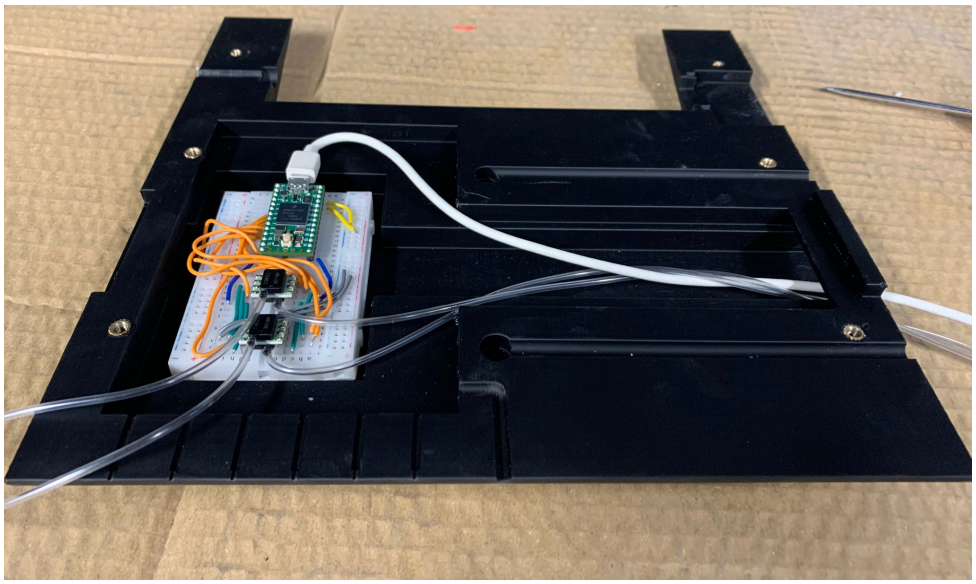


Figure 3.8: *Non-sensing side.*

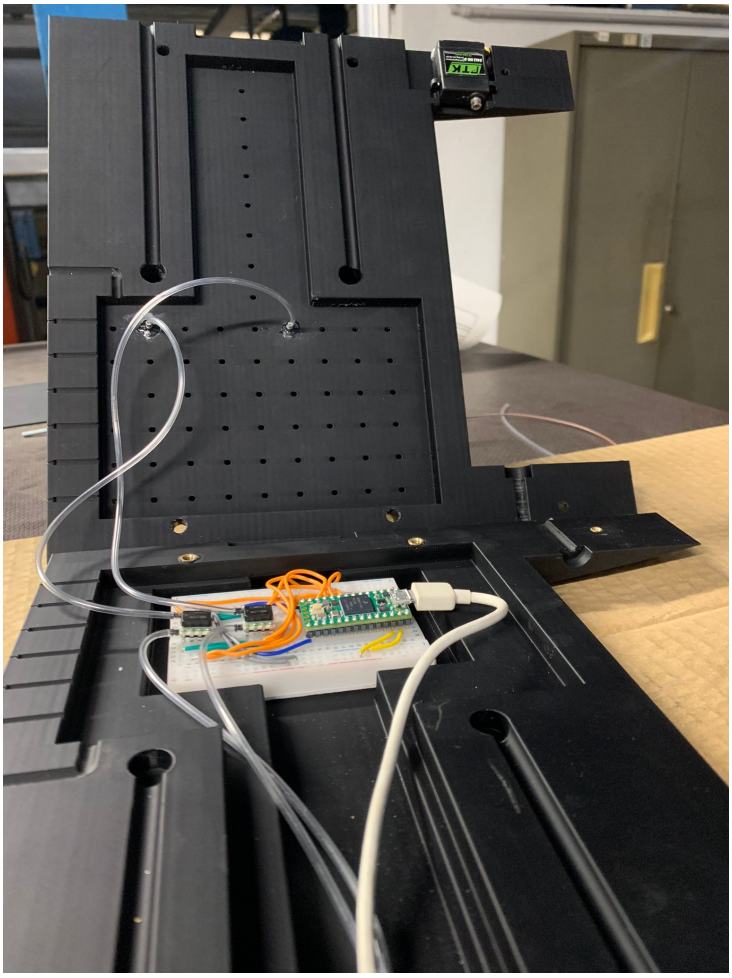


Figure 3.9: *Sensing side & Non-sensing side* inside.

Once these two components are inserted, the fastening of *Sensing side* and *Non-sensing side* can take place. Given the minimal dimension of the previously mentioned plastic tubes and of the cable connected to the acquisition module, their *CAD* visualization is omitted. It should only be noted that this cable will

carry the signal to the outside, passing through the *Support* and *Beam* passing in the dedicated channel. The union of *Sensing side* and *Non-sensing side* results as follows:

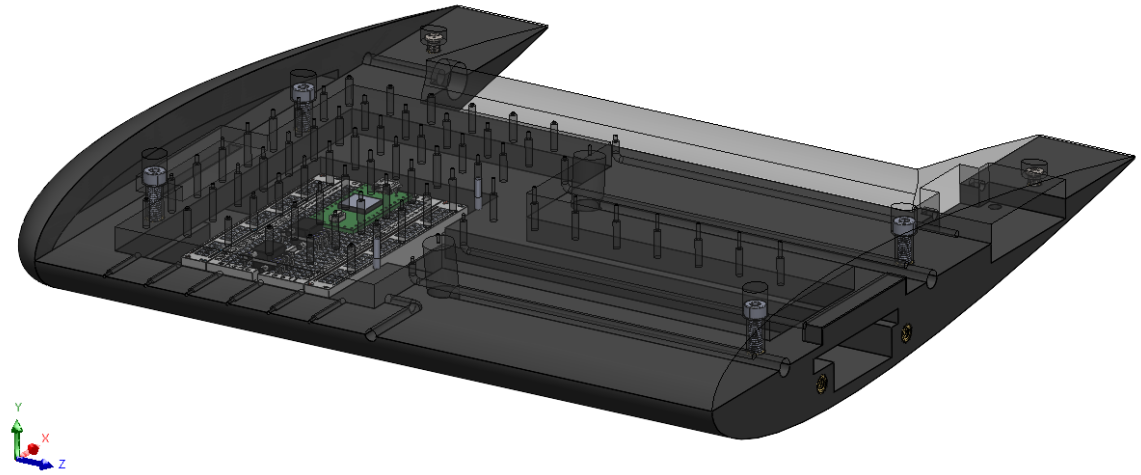


Figure 3.10: Closing *Non-sensing side-Sensing side*.

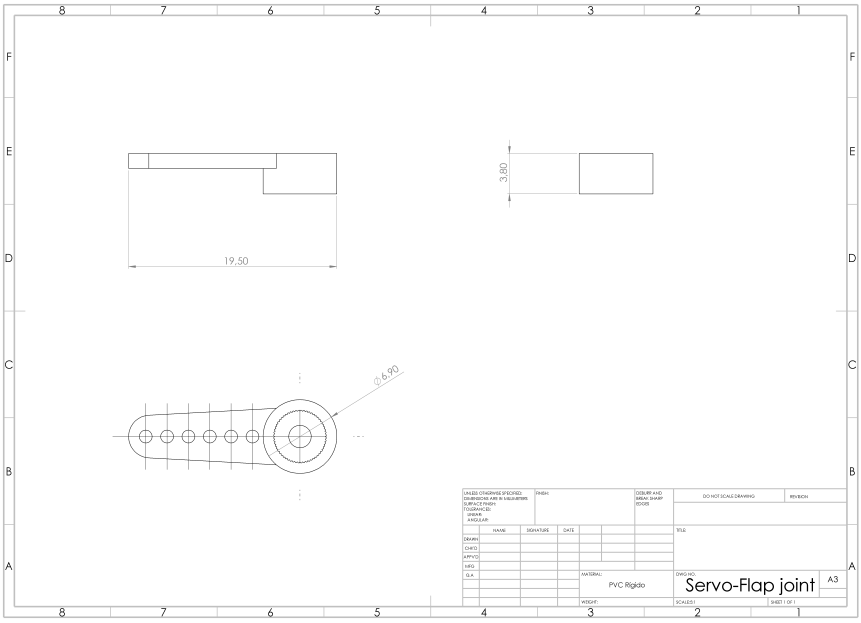
Once the configuration shown is obtained, ensuring perfect alignment of the parts, the *Side Part* is then fixed as previously described. It is specified that, due to the limited space for the slot, *M4* type screws with reduced shank length are required.

Figure 3.11: Fixing *Side Part-Profile*.

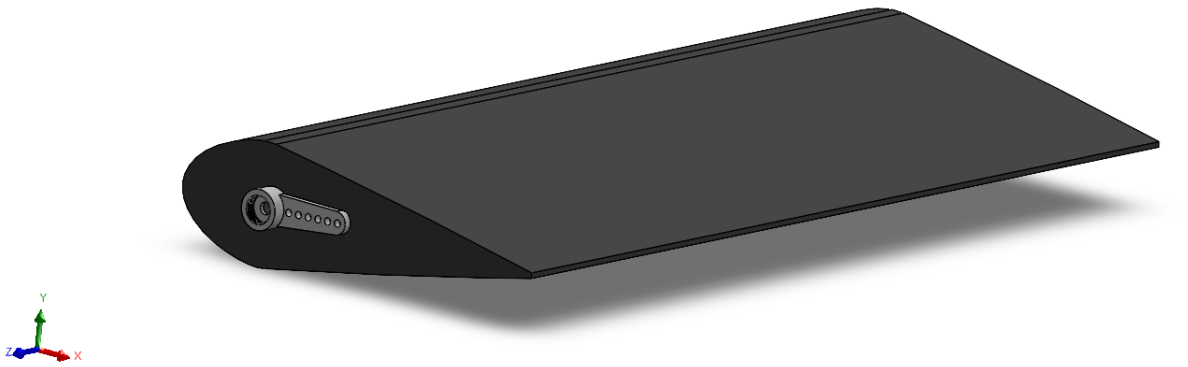
Servo-Flap Fitting

Once the complete structure of the *Profile* is obtained, it is possible to proceed with the insertion of the *Shaft* and *Flap*. However, the *Flap* requires further processing before its insertion. A fitting method of this component with the *Servo Motor* must be considered. As previously mentioned, the idea is to fix a plastic joint at one *Flap* side by using glue.

This joint features an insert of the *Servo Motor head* size, ensuring a proper engagement. This process allows for the perfect fitting of the *Flap* and *Servo*, eliminating the need for the *Shaft*, which, as explained, remains only as a component for fixing the rotational axis of the *Flap*.



(a) Servo Motor pointer: dimensions.



(b) *Detail*: Flap-Servo coupling system.

Figure 3.12: Flap-Servo coupling system.

Figure 3.13: *Flap*, *Shaft*, and *pointer*.

Figure 3.14: Flap-Servo coupling system.

The gluing operation is done with hot glue. Additionally, the *Servo Motor*, housed and engaged in its respective insert within the *Flapping Wing*, is further fixed to the *Sensing side* part using hot glue to prevent it from shifting or detaching from the *Flap*.

Flapping Wing Assembly

Once the mentioned operations are completed, the *Flap* is fixed to the *Profile* through the insertion of the *Shaft*. There is minimal tolerance between the *Shaft* and the *Profile*, so the lateral movements of the *Flap* are controlled and minimized.

The following configuration is thus achieved:

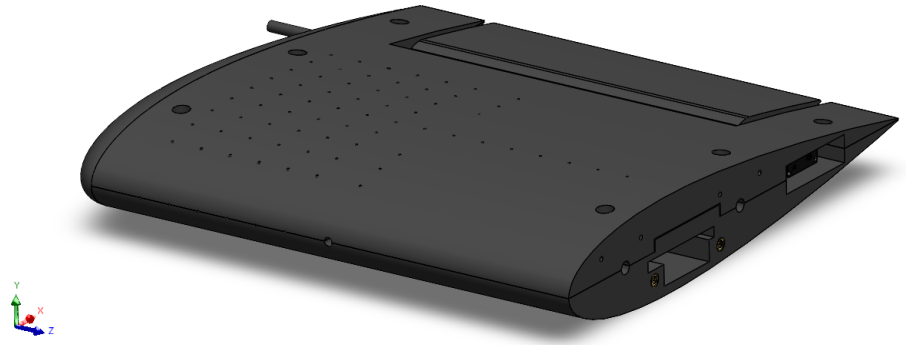


Figure 3.15: *Flapping Wing.*

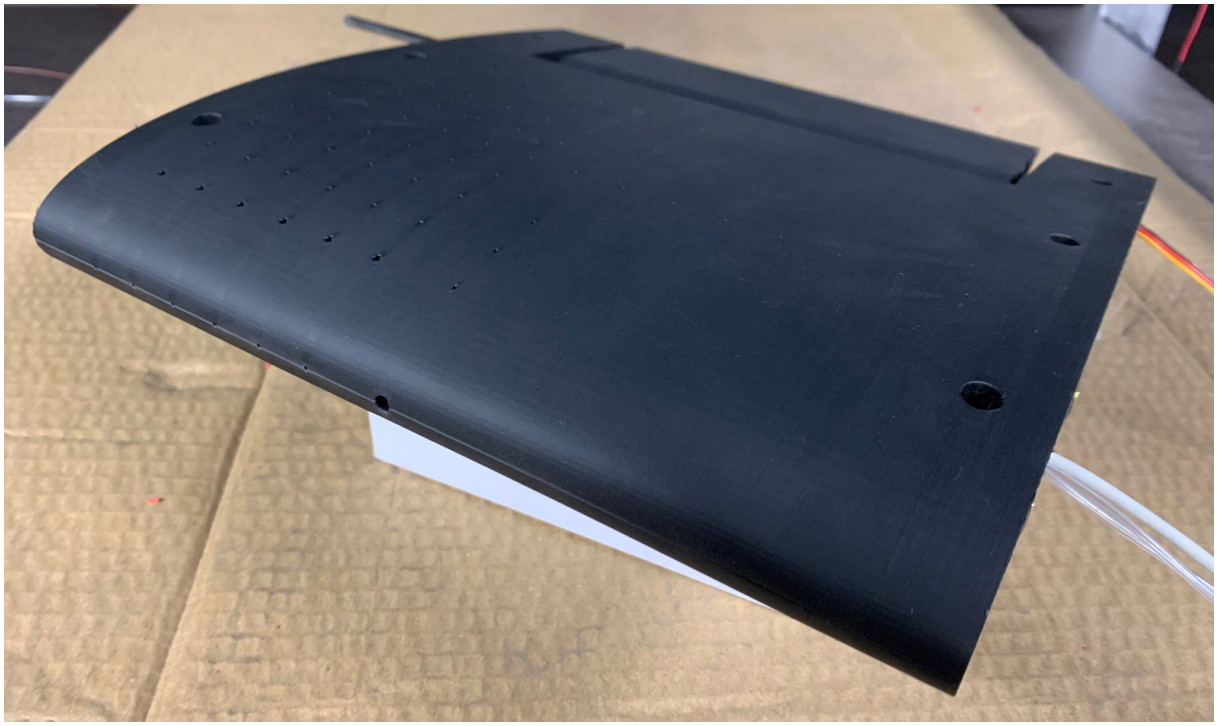


Figure 3.16: *Flapping Wing.*

Final Assembly

Once the complete wing structure is obtained, with the electronic components also inserted inside, the assembly is then fixed to the *Support*. It is important to note that the *Flapping Wing* is not directly fixed to the *Support*. In fact, using threaded $M4$ rods passing through ducts in the *Support*, the *Flapping Wing* is directly fixed to the *Beam*. Once this operation is completed, the full *setup* of the experiment is obtained.

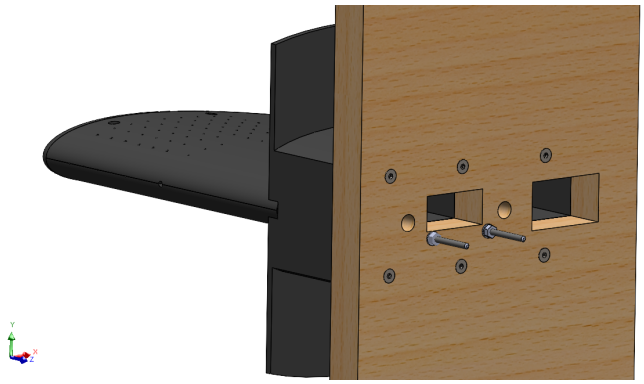


Figure 3.17: *Beam-Flapping Wing* fixing.

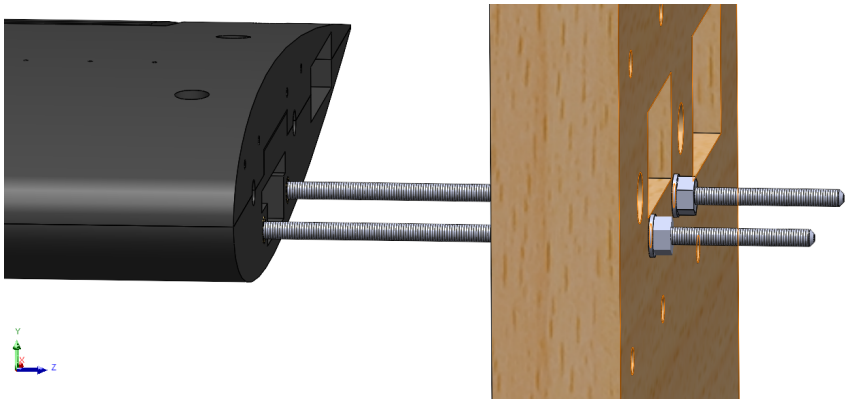


Figure 3.18: *Detail*: *Beam-Flapping Wing* fixing.

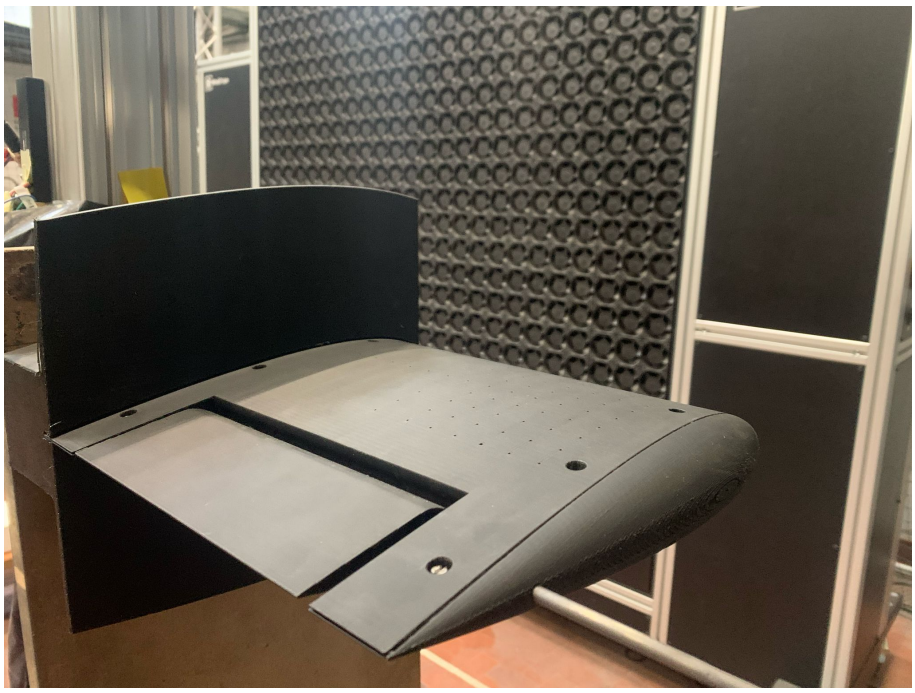


Figure 3.19: Flapping Wing.

It should be noted that, for measurement purposes, it was decided, after the design process, not to use the microphone capsules, so the respective housings mentioned in the first chapter are empty. For data

acquisition, the static pressure ports, whose ducts terminate outside the structure, also mentioned in the previous chapter, will not be used. The signal from the *Board* reaches the outside of the structure by using a *USB-Type C* cable, passing through the dedicated ducts in the *Support* and *Beam*, then connecting to a cable extension and reaching the PC.

It is anticipated that the *Servo* movement is controlled by a second module (*Raspberry Pi Pico* type), connected to it through jumpers. This control module is located outside the structure, fixed to the *Beam* side opposite to the *Support*.

Given the presence of cables and *boards*, the side of the *Beam* opposite to the *Support* is covered with a *Teflon* sheet to prevent issues once the structure is subjected to gust.

Finally, the *Beam* is fixed to a vertical column using a pin with adjustable inclination, aiming to achieve a configuration that keeps the *Flapping Wing* flat, with no incidence or dihedral angle.

An additional sensor (Hot-Wire probe) is placed outside the structure on a dedicated guide, downstream the *Flap*. Since this is not a part of the structure but is placed outside, it will be discussed further.

Figure 3.20: Final Assembly.

3.2 Sensors

This section provides a detailed overview of the sensors used for data acquisition and the *Servo Motor*, which controls the *Flap* movement. The components used for data acquisition are:

- two *Honeywell RSC* pressure sensors;
- a *Hot Wire* probe.

3.2.1 Pressure Measurement

For pressure measurements, it is preferred to use two *Honeywell* sensors from the *RSC* series. Before listing the detailed specifications of the sensor used, it is preferred to provide a complete overview of pressure measurements in aerodynamics.

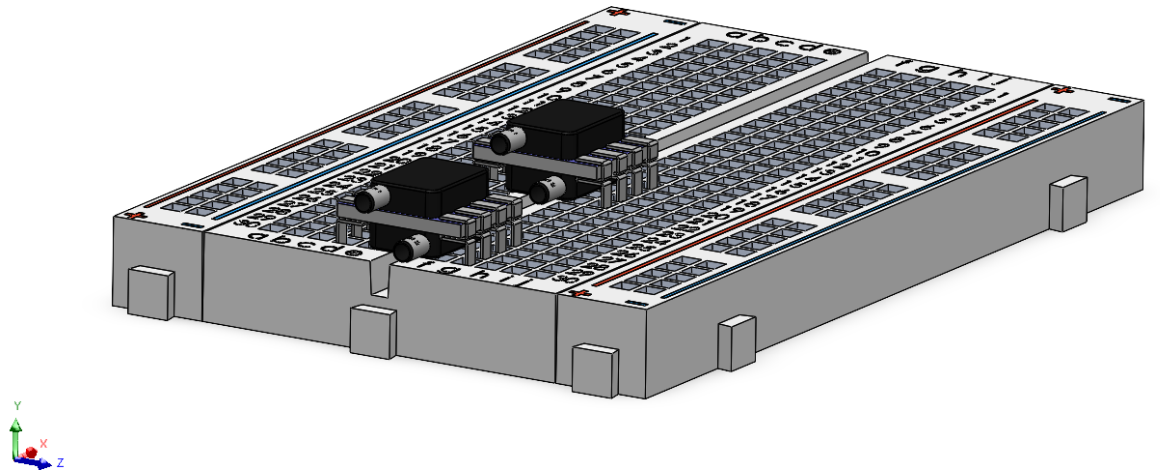


Figure 3.21: Honeywell RSC.

In fluid dynamics, especially adopting an experimental approach, pressure measurement allows for a quantitative study in a simple, reliable, and cheap manner in many situations. Through this measurement, it is possible to: determine the flow velocity magnitude, the flow direction, local forces (pressure or friction), pressure drops, or determine the aerodynamic forces acting on a body.

Specifically, when referring to *pressure* measurement, the following quantities are considered:

- *Total Pressure*: the pressure that occurs at a point on the body where the flow is subjected to an isentropic stop. This process must occur without dissipative phenomena that would increase entropy (shock formation, boundary layer), as such phenomena would reduce the total pressure. The point on the body where this happens is called the *Stall Point*. The total pressure is defined as the air pressure *impacting* the body and is essentially the maximum pressure value expected.
- *Static Pressure*: a scalar quantity that acts with equal intensity in all directions. For instance, in the case of a surface being swept by an air flow, this refers to the action of pressure perpendicular to the surface itself. This value is considered constant in the normal direction to the flow direction.

The two pressure values mentioned above are closely related by the following equation, known as the *Bernoulli relation*, valid for incompressible flows, at $M < 3$:

$$p + \frac{1}{2}\rho V^2 = p_0$$

Where:

- p, static pressure;

- p_0, total pressure is constant along a streamline across the entire field;
- $\frac{1}{2}\rho V^2$, dynamic pressure.

This relation holds for small variations in height along a streamline and arises from integrating the energy conservation equation for incompressible, irrotational, and non-viscous flows [16].

Pressure Probes

It can be concluded that:

- The flow static pressure can be measured on the surface of the profile through a static pressure probe (*pressure tap*). This port features a hole with an appropriate diameter perpendicular to the surface, and thus also to the flow direction. These ports are clearly visible across the upper surface of the *Flapping Wing*.
- The flow total pressure must be measured at the isentropic stall point by placing probes in the frontal plane of the body. At the stall point of the *Flapping Wing*, there are some total pressure ports that, for the purpose of the experiment, will not be used.

In summary, static pressure is measured at every point on a surface swept by the flow using a pressure port whose axis is perpendicular to the surface, so that it *only senses the tangential flow to the surface itself* [16]. Total pressure is measured at the isentropic stop point of the flow using a pressure port aligned with the flow direction.

The fundamental requirements for pressure ports are:

- Low *intrusiveness*, small dimensions;
- High *spatial resolution*, point-wise measurements;
- *Robustness* and *ease while using*.

Pressure Transducers

The pressure taps are linked to *pressure transducers*, which are the components of the measurement chain responsible for quantifying the pressure to be measured using a physical principle. In the current experiment, this role is assigned to the *Honeywell RSC* sensors. Typically, after the pressure signal goes in, this component produces an electrical signal, usually a voltage, which once acquired and processed allows the determination of the measured pressure value.

There are mainly two types of *pressure transducers*, differentiated by the reference they are based on for their measurements: *absolute pressure* transducers and *differential pressure* transducers.
Absolute pressure transducers measure pressure values relative to the *vacuum* condition (corresponding to $p = 0$). The vacuum condition is present in one of the two chambers of the transducer. The advantage of not having a reference pressure to measure is that it simplifies the measurement process. However, a disadvantage is the potential presence of non-linearities associated with significant deflections in the current. For the same pressure p, the pressure jump Δp is larger. The non-linearity also means that sensitivity varies.

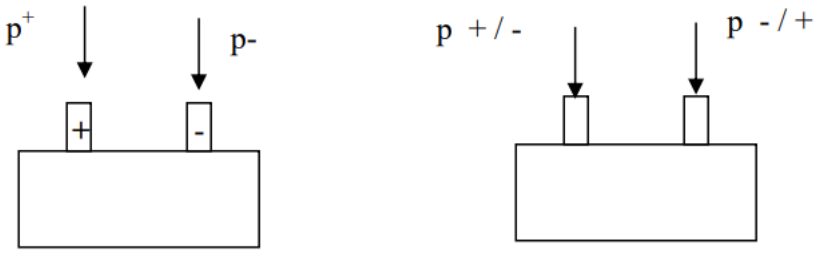
Differential pressure transducers measure the pressure difference $p - p_{ref}$, so they have two inputs: one for the pressure to be measured and one for the reference pressure. The reference pressure used is defined by the operator depending on the measurement to be done. This reference value can be either *ambient pressure* or a pressure chosen by the operators based on their needs. The reference pressure choice must be made considering the magnitude of the pressure to be measured. Indeed, the reference pressure should not be much larger or smaller than the pressure to be measured, as the useful information (the pressure to be measured) would be *masked* by the reference pressure[16].

(a) Absolute Pressure Transducers (b) Differential Pressure Transducers

Figure 3.22: Types of pressure transducers.

Differential pressure transducers are further divided into two categories:

- *Unidirectional*, where the higher pressure must always be applied to the same input marked by the manufacturer with the + sign. The output signal is positive;
- *Bidirectional*, which can measure both positive and negative pressure differences. The electrical output signal changes based on the input pressures. By using these transducers, the inputs can be switched.



(a) Unidirectional Differential Pressure Transducers (b) Bidirectional Differential Pressure Transducers

Figure 3.23: Types of differential pressure transducers.

Pressure transducers, whether absolute or differential, have the following characteristics:

- *Sensitivity* of the transducer: the ratio between the output signal resulting from the pressure difference applied and the pressure difference itself.

$$S = \frac{\Delta s}{\Delta p}$$

  It is required for sensitivity to be high, meaning that, for a unit pressure difference applied, the transducer should respond with the highest possible output in terms of electrical signal;

- *Full Scale (FS)* of the transducer: the measurable pressure range. The extreme values of measurable pressure correspond to the maximum and minimum values of the output electrical signal;
- *Frequency Response*: the ability of the transducer to *follow* the time variations of the pressure. This characteristic depends on the specifics of the transducer. If the goal is to make measurements under unsteady flow conditions or in the presence of rapid pressure fluctuations over time, this characteristic becomes crucial and should be as high as possible;
- *Resonance Frequency*: a property related to the specific design of the transducer and its geometric characteristics. Resonance occurs when the input pressure oscillates at the transducer's natural frequency: under these conditions, signal amplification occurs, and the measured pressure exceeds the applied pressure. The resonance frequency must be significantly higher than the highest expected frequency in the measurements.

- *Spatial Resolution*: a property related to the size of the transducer's sensitive element. The smaller the sensitive part, the more accurate the measurement will be. Precise measurements occur when spatial resolution is high.
- *Measurement Accuracy*: the smallest value measurable by the transducer, given as a percentage of the *Full Scale* ($\%FS$). Typically, it is of the order of $0.5\%FS$. The lower the value, the greater the measurement accuracy.

It is important to note that pressure transducers can be more or less sophisticated components, either electrical or manual. The simplest pressure transducer is the *manometer*, which converts a pressure difference into a change in the height of a manometric fluid by using *Stevin's law*:

$$\Delta p = \Delta h \gamma$$

where:

- Δp, the measured pressure difference;
- Δh, the change in the height of the manometric fluid;
- γ, the specific weight of the manometric fluid.

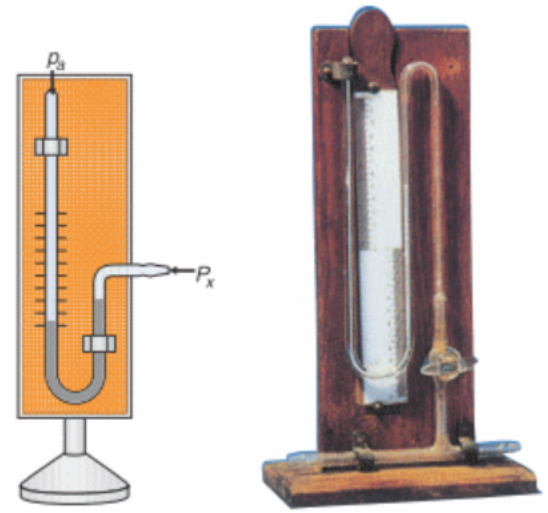


Figure 3.24: Manometric tube.

Electrical pressure transducers, like the *Honeywell RSC* sensors used in the current experiment, are characterized by *sensitive elements* that convert the pressure difference Δp into an electrical output signal (voltage) proportional to the potential ΔV. These transducers can be:

- *Transducers with flexible diaphragms*: either *Resistive* or *Capacitive*;
- *Piezoelectric* or *Piezoresistive* transducers.

In the case of *capacitive* transducers, the variation in the capacitance of a capacitor due to the pressure force is used. One of the capacitor plates is fixed, while the other is flexible. Under the effect of the applied pressure, the flexible plate deforms and changes the distance between the plates, altering the capacitance. The variation in capacitance $C(t)$ is linked to the variation in the electrical output signal over time, $V(t)$. The capacitance of a capacitor is defined as:

$$C = \frac{K_d A}{X_0}$$

where:

- C, capacitance [F];

- K_d, dielectric constant of the capacitor;
- A, surface area of the plate $[m^2]$;
- X_0, distance between the capacitor plates $[m]$;

The main parameter in measurement is the distance between the plates. Due to deformation, the distance $X_0(t)$ between the flexible and fixed membranes changes. A change in distance corresponds to a variation in capacitance C, which in turn leads to a variation in potential ΔV. The relationship between the pressure difference Δp and the voltage difference ΔV is called the "calibration curve" of the transducer.

The frequency response of such transducers depends on the dynamic deformation of the membrane. Larger membranes have higher inertia and longer response times, resulting in a lower frequency response. At the same time, the larger the membrane surface, the greater the transducer's sensitivity, as for the same Δp, the force needed to deform the membrane will be higher, increasing the output electrical signal. Capacitive transducers are generally characterized by moderate frequency response, which depends on the dimensions of the sensible element of the transducer.

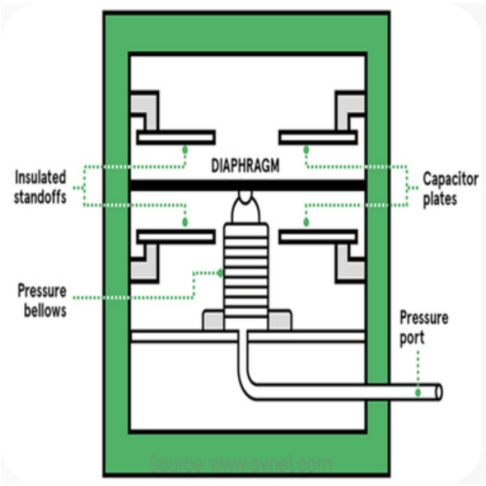


Figure 3.25: *Schematic*: capacitive differential transducer.

Resistive transducers (*strain gauges*) use the variation in resistance associated with the deformation of a wire. This deformation occurs due to the pressure applied to the sensitive element. The *strain gauge* is essentially a resistive coil glued onto a plastic support. In a resistive transducer, the main variable is the change in length of the sensible element Δl. It is recalled the resistance R definition:

$$R = \frac{\rho l}{A}$$

where:

- R, resistance $[\Omega]$;
- ρ, resistivity of the material $[\frac{m}{\Omega}]$;
- A, cross-sectional area of the wire;
- l, length of the wire.

Due to the pressure difference Δp, a change in resistance ΔR occurs. The relative variation in resistance $\frac{\Delta R}{R}$ of the strain gauge is directly associated with the relative deformation ε, generally given by: $\frac{\Delta l}{l}$. The relative variation in resistance corresponds to a relative variation in the voltage applied across the resistance. The *sensitivity* K of the strain gauge is defined as:

$$K = \frac{\frac{\Delta R}{R}}{\frac{\Delta l}{l}}$$

The relationship between the pressure difference Δp and the voltage difference ΔV is governed by the *calibration curve* of the transducer.

For $\frac{\Delta l}{l} > 10^{-3}$, the phenomenon of *hysteresis* occurs, where under conditions of membrane discharge (reduction of Δp), the electrical signal does not follow the same path as the calibration curve under loading conditions. This behavior is also present in capacitive transducers. Additionally, these sensors are very sensitive to temperature and environmental conditions, as material expansion can occur even due to *Joule's effect*, resulting from temperature variations in the flow or ambient temperature. Particularly, the resistivity ρ is strongly influenced by temperature changes ΔT. Flow temperature changes can therefore introduce spurious effects into the measurements.

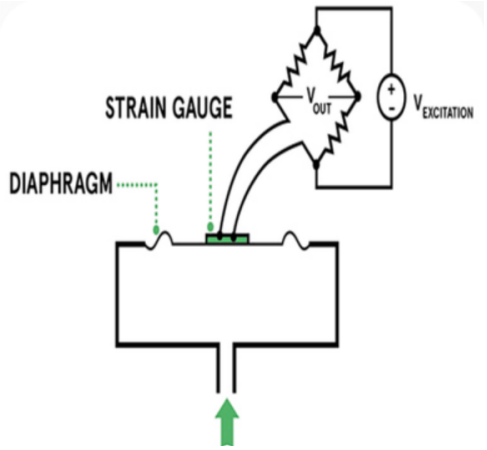


Figure 3.26: *Schematic*: resistive differential transducer (*strain gauge*).

In *piezoelectric* transducers, the piezoelectric property of certain materials is used. This property gives these materials their ability to generate an electric field if subjected to mechanical deformation. The reverse process also occurs, where mechanical deformation happens when an electric field is applied. The piezoelectric constant d is defined as:

$$d = \frac{Q}{F}$$

where:

- d, piezoelectric constant $[\frac{C}{N}]$;
- Q, electric charge on the piezo material;
- F, applied force.

This constant can be interpreted as the ratio between polarization and the applied pressure force.

$$d \approx 10^{-12} \ \frac{C}{N}$$

The electrical signal and the applied pressure are related through a calibration constant that must be determined experimentally.

Piezoelectric sensors are very sensitive to accelerations, which makes them widely used for measurement purposes. However, this property makes their use more complicated in pressure measurements. Therefore, a solution *in tandem with two crystals*[16] is used to cancel the acceleration sensitivity effects. These transducers are characterized by: good *stability* against temperature changes, a very small *sensitive part* requiring low power, *resonance frequencies* higher than other transducers, and very high *frequency response* and *spatial resolution*. They also feature a *preamplifier* and high *sensitivity*, making them particularly suitable for measuring fluctuating pressures at high frequencies.

Honeywell RSC

The sensors used for the experiment are the *Honeywell RSCDRRM2.5MDSE3* model. These sensors, part of the *RSC* series, are piezoresistive silicon pressure transducers. Specifically, the used model is a bidirectional differential type with an accuracy of $0.5\%FS$. These sensors are calibrated and thermally compensated for sensor offset, sensitivity, temperature effects, and non-linearity using a 24 *bit* analog-to-digital converter with integrated $EEPROM$[1][17]. Pressure data can be acquired at rates between 20 and 2000 samples per second via an SPI[2] interface. These sensors are designed for use with non-corrosive and non-ionic gases, such as air.

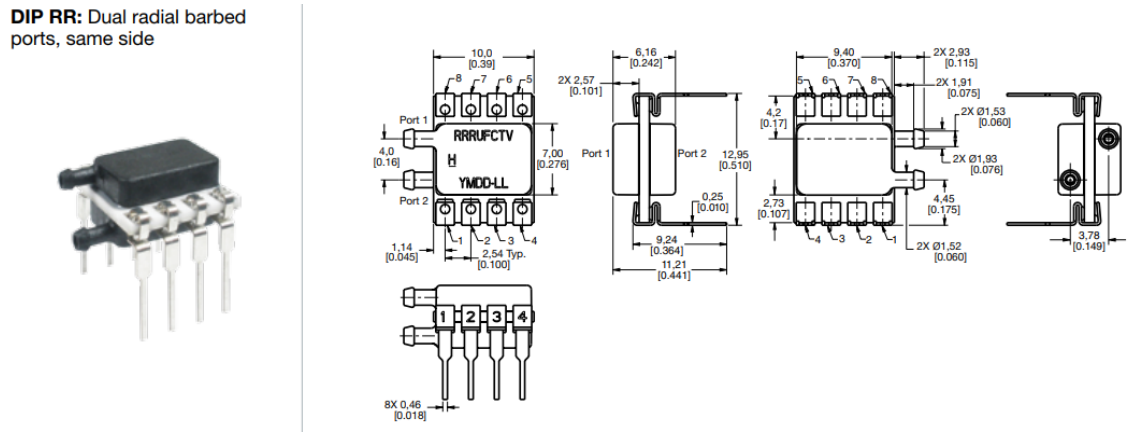


Figure 3.27: *Detail*: Honeywell RSCDRRM2.5MDSE specifications

SPI (*Serial Peripheral Interface*) is a serial communication protocol that allows a microcontroller (or another main device) to communicate with peripheral devices such as sensors, memories, or analog-to-digital converters. The protocol uses a 4-wire connection:

- *MOSI* (Master Out Slave In): the pin through which the master sends data to the slave device;
- *MISO* (Master In Slave Out): the pin through which the master receives data from the slave;
- SCK (Serial Clock): the pin that transmits the clock signal, which synchronizes communication;
- SS (Slave Select): the pin that indicates when the slave device should be activated to send or receive data.

SPI is a high-speed protocol and is used for applications that require rapid and reliable communication between electronic devices.

Regarding the sensor model:

$$RSCDRRM2.5MDSE3$$

it is possible to notice:

- RSC, product series;
- D, package type;
- RR, indicating the model *Dual radial barbed ports, same side*;
- M, temperature range from $0°C$ to $50°C$;
- $2.5MD$, *pressure range* of ± 2.5 *mbar*;

[1]EEPROM stands for Electrically Erasable Programmable Read-Only Memory. It is a type of non-volatile memory that can be electrically read, written, and erased. Unlike RAM, which loses data when the device is powered off, EEPROM retains data even without power. It is often used to store data that must remain unchanged even after power off, such as configurations, calibrations, or system parameters.

[2]Serial Peripheral Interface

- *S*, output *SPI*;
- *E*, *External Math* as a transferable function;
- 3, operating voltage.

The sensors have 8 different pins, summarized as follows:

***Honeywell RSC* Pinout**		
Pin	Name	*Description*
1	SCLK	*External Clock Source*
2	DRDY	*Data Ready: Active Low*
3	DIN	*Serial Data Input*
4	CS-ADC	*ADC Chip Select: Active Low*
5	GND	*Ground*
6	VCC	*Positive Supply Voltage*
7	CS-EE	*EEPROM Chip Select: Active Low*
8	DOUT	*Serial Data Output*

Table 3.1: Pinout[17].

The two sensors are connected to a *Breadboard* and to a *Teensy 4.0* module that controls the acquisition of signals, using *jumpers*. *Teensy 4.0* provides power at 3.3 V or 5 V. Given the sensor specifications, we choose a 3.3 V power supply.

Both sensor inlets are connected to plastic tubes, one of which will be connected to a stainless-steel *INOX* tube connected to the pressure ports mentioned earlier. The other plastic tube will follow the same path as the USB Type-C cable that powers the *Teensy 4.0* module, leading to the outside of the *Beam*. This pipes, particularly, will measure the reference pressure in still air. The two tubes from the two respective sensors that are not connected to the pressure probes and will act as reference are connected, outside the structure, to a *T-joint* that will ensure a single reference value is measured.

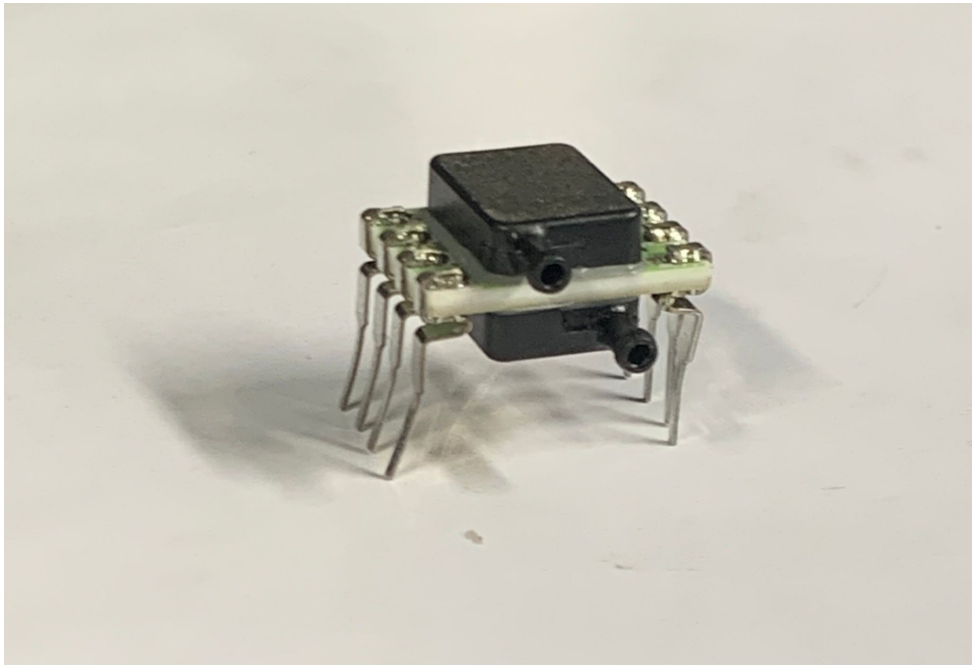


Figure 3.28: *Honeywell RSC* used.

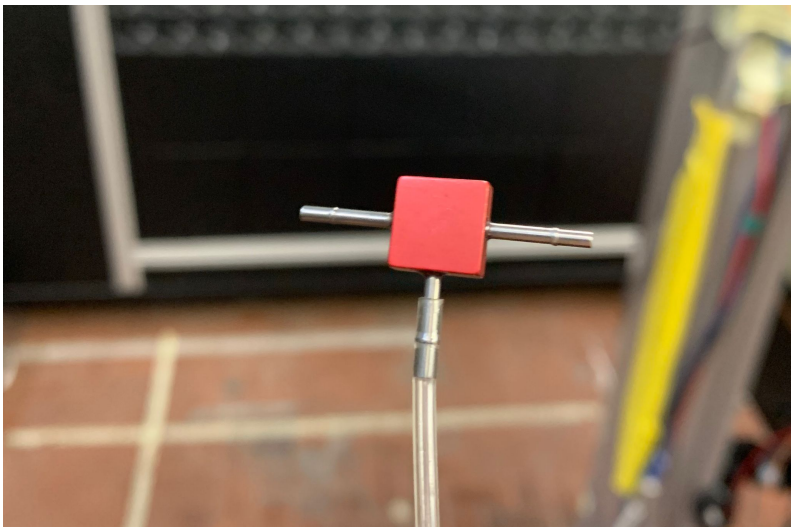


Figure 3.29: T-joint used.

By choosing to use two identical sensors for pressure measurement it is clear that creating an electrical circuit on the *Breadboard* will allow to save space and will make the entire electronic setup less complicated. For this purpose it will be build a simple electrical circuit that will connect the sensors pin which do not request a specific connection with the acquisition module. In fact, the sensor pin for the clock (1), DRDY (2), DIN (3), EEPROM (7) and DOUT (8) do not necessitate a different connection with the board for the two different sensors. For this reason it is possible to link this pins between the two sensors and associate this signal to the specific module pin to save space. Also the VCC (6) and GND (5) may be linked between the two sensors. The ADC-CS (4) pin of the two sensors, where the pressure signal is measured, has to be linked specifically with the board by two different jumpers to avoid data overwriting.

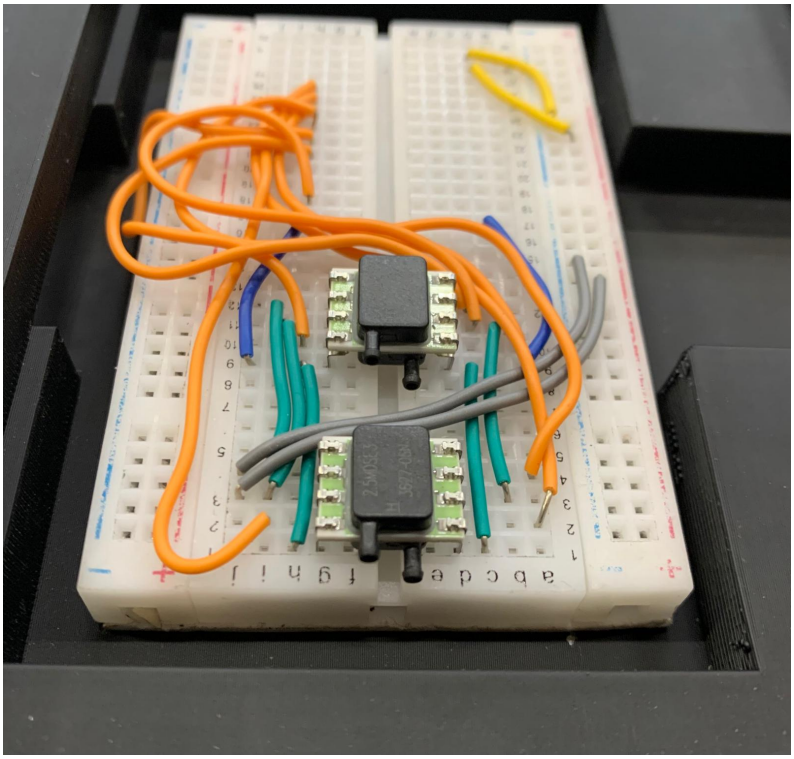


Figure 3.30: Elecrical Circuit.

3.2.2 Hot-Wire Anemometry

The experimental study of turbulent flows, such as in this experiment, requires using probes that meet strict requirements. In particular, given the highly turbulent nature of the airflow downstream the *Flapping Wing*, it is necessary to use *hot-wire anemometry*.

Figure 3.31: Hot Wire Probe.

This measurement technique allows for:

- Low *intrusiveness*, given the small size of the probe. This feature ensures minimal disturbance to the wake flow;
- High *spatial resolution*, as the sensitive element of the probe is smaller than the smallest scale to be measured;
- High *frequency response*;
- High *sensitivity*, defined as $S = \frac{\Delta E}{\Delta V}$;
- *Stiffness* of the structure, ensuring the stability of the *probe-probe holder* system.

This technique, through a *hot-wire anemometer*, allows for highly accurate measurements of *velocity* (magnitude and/or direction) or *temperature* of the flow. The signal acquired through this technique is *continuous*. However, the *Hot Wire* technique has some disadvantages compared to other techniques for measuring turbulent flows. In particular, the *Hot Wire* probe requires proper calibration before measurements and, due to the significant influence of environmental conditions, it may require multiple calibrations in a day. Additionally, due to the extreme delicacy of the sensitive element, great care must be taken to avoid contamination, damage or failure.

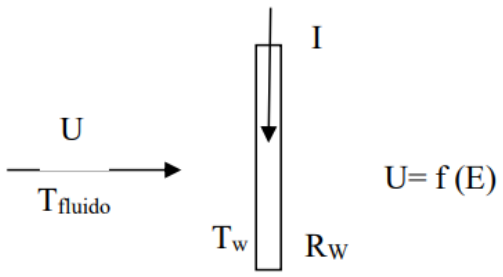


Figure 3.32: Working principle of a *Hot Wire*.

The physical principle used by the *Hot Wire* probe is *Joule's effect*. The sensitive element of the probe, the *hot wire* (a conductor made of metal), is heated and maintained at temperature through electric current. This sensitive element is brought to a certain temperature T_w then is cooled by the flow, which has a lower temperature $T_f < T_w$, causing a decrease in its temperature by ΔT. The temperature difference ΔT results in a variation in the resistance of the wire ΔR, which in turn causes a change in the electrical signal ΔV. The potential difference ΔV is related to the velocity of the current.

Given the length l of the sensitive element and the diameter d of the section of the wire, and assuming large values of $\frac{l}{d}$, the wire is treated as *infinitely long*. We proceed by analyzing the heat exchange between the flow and the hot wire. The amount of heat exchanged depends on:

- *velocity* of the flow U;
- *temperature difference* between the wire and the flow, $T_w - T_f$;
- *fluid properties*, such as density ρ, viscosity μ, and conductivity k;
- *wire properties*, such as resistivity coefficients;
- dimensions of the sensitive element;
- flow incidence angle α.

Heat exchange occurs mainly through:

- *Natural convection* q_n;
- *Forced convection* q_f;
- *Conductivity of the wire* towards the metal supports q_c;
- *Radiation* q_i.

It is assumed that *radiation* and *natural convection* are negligible compared to *forced convection*. Radiation becomes significant when the temperature of the hot body exceeds approximately $300°C$, while natural convection is important when fluid velocities are low ($U < 0.5\ \frac{m}{s}$)[18]. The deviation from the *continuous medium* assumption is also neglected.

The forced convection heat transfer q_f is governed by the following equation:

$$q_f = h \cdot (\pi dl) \cdot (T_w - T_f)$$

where (πdl) refers to the *geometry* of the sensitive element, $T_w - T_f$ is the thermal exchange, and h is the *heat transfer coefficient* for forced convection.

Under thermal equilibrium, the heat exchanged between the sensor and the flow is balanced by the heat generated per unit of time due to *Joule's effect* from the electric current passing through the wire:

$$q_j = R_w I^2$$

where R_w is the resistance of the wire and I is the current through it. The temperature difference between the wire and the flow, which governs the heat exchange rate, is related to the heat flux by the following expression:

$$\Delta T = \frac{R_w I^2}{h \cdot (\pi dl)}$$

As a result, the temperature difference, and thus the heat exchange, is inversely proportional to the fluid velocity, which can be measured through the heat transfer relationship.

Calibration of the Probes

The hot-wire anemometer measure is not absolute since the probe responds with a voltage signal when the sensor is cooled by the airflow. To convert the voltage signal read into velocity, a previous calibration of the system is required. This operation must be conducted in a *known flow* in terms of velocity and direction of the current. In general, for a given probe in a given medium with a fixed overheating ratio, the response depends not only on the velocity magnitude but also on the flow direction. The relationship to be found is:

$$E = f(U, \alpha, \beta)$$

The calibration equation expresses the cooling of the sensor in terms of $E = E(U)$ under conditions where the sensor is perpendicular to the current ($\alpha = 0, \beta = 0$). There are various proposed relations linking voltage and velocity of the flow that produces the cooling of the sensor. Some of the most common and widely used calibration equations found in the literature are reported below. The previously mentioned *King's Law* is provided:

$$E^2 = A + BU^n$$

where A, B, and n need to be determined during calibration, and there are several way to proceed. Another approach is based on *polynomial laws*. Furthermore, interpolating polynomials or other types of relations can be used, particularly fourth or fifth-degree polynomials are often employed. In the current experiment, a fourth-degree polynomial law is preferred to calibrate the *Hot Wire*:

$$U = a + bE + cE^2 + dE^3 + eE^4$$

Due to the extreme sensitivity of the probe to environmental conditions, to achieve the most precise possible values for the cooling law coefficients, calibration of the *Hot Wire* is performed before each measurement.

Dantec 55P11

The probe used is a *Hot Wire* model *Dantec 55P11*. This model has straight prongs, with the sensor perpendicular to the axis of the probe. It measures both mean and fluctuating velocities in one-dimensional flows. It is mounted with the probe axis parallel to the flow direction[19].

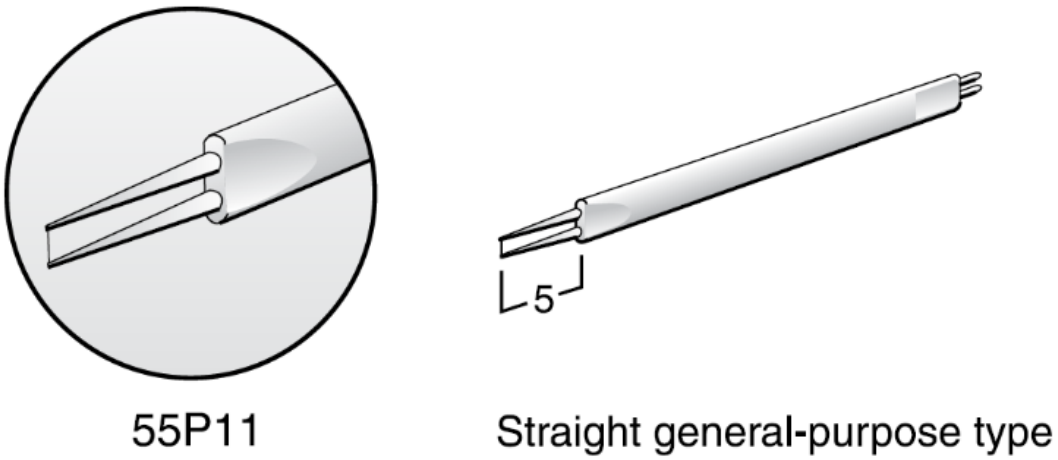


Figure 3.33: *Dantec 55P11.*

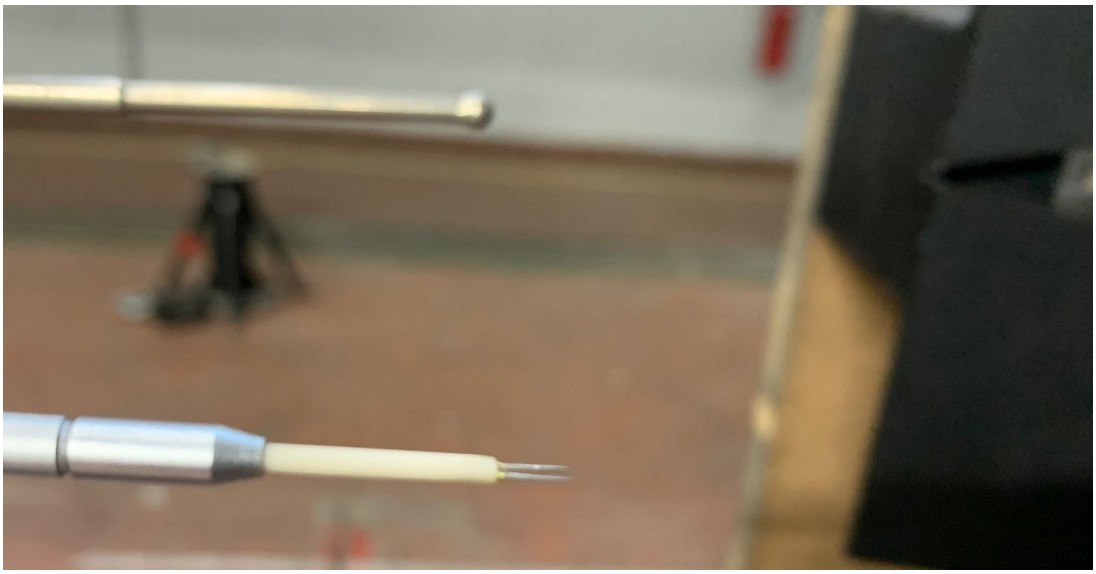
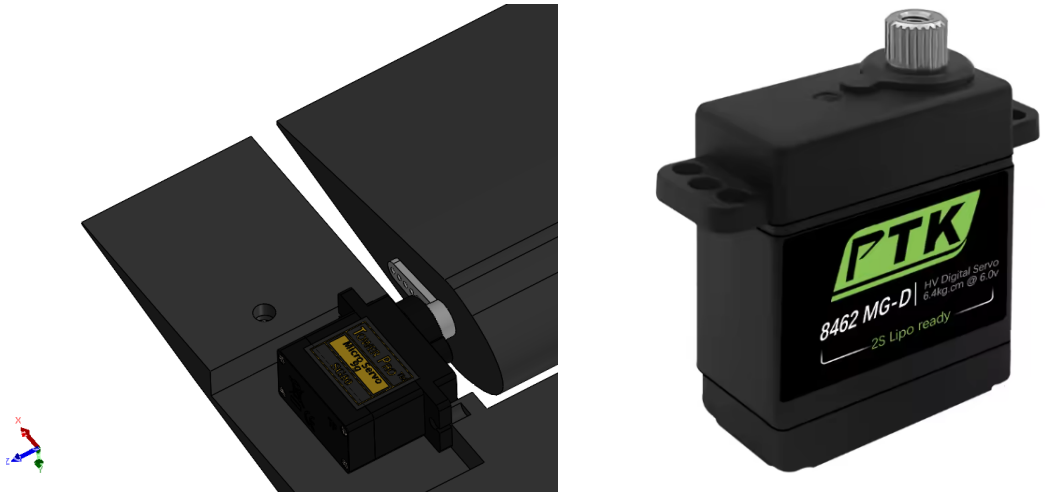


Figure 3.34: *Dantec 55P11.*

This probe is mounted on an adjustable vertical guide and positioned in the wake, a few centimeters from the *Trailing Edge* of the *Flap*.

3.2.3 Servo Motor

The *Servo Motor* responsible for the *Flap* movement is a *PTK & VOTIK 8462 MG-D*. This component is controlled by a PWM^3 signal. The component responsible for controlling the *Servo Motor* is a *Raspberry Pi Pico* module connected through *jumpers* to the *Servo* itself.



(a) Servo Motor (*SolidWorks*) (b) Servo Motor

Figure 3.35: Servo Motor.

A PWM signal is a technique used to modulate the average power delivered by varying the pulse duration in a signal cycle. Essentially, it involves generating a digital signal that alternates rapidly between two states, high and low (ON/OFF), but changing the percentage of time the signal stays high (ON) during each cycle. The pulse width (or *duty cycle*) is the percentage of time the signal stays high compared to the total cycle time.

This component operates at a frequency of 333 Hz[20], effectively setting a limit on the acquisition and movement speed equal to this frequency. The operating voltage range varies from 4.8 to 7.4 $V\ DC$[20]. A voltage close to the maximum operating voltage will result in better performance, so it is anticipated that an external power supplier will be used to provide 7.2 $V\ DC$ to the *Servo Motor*.

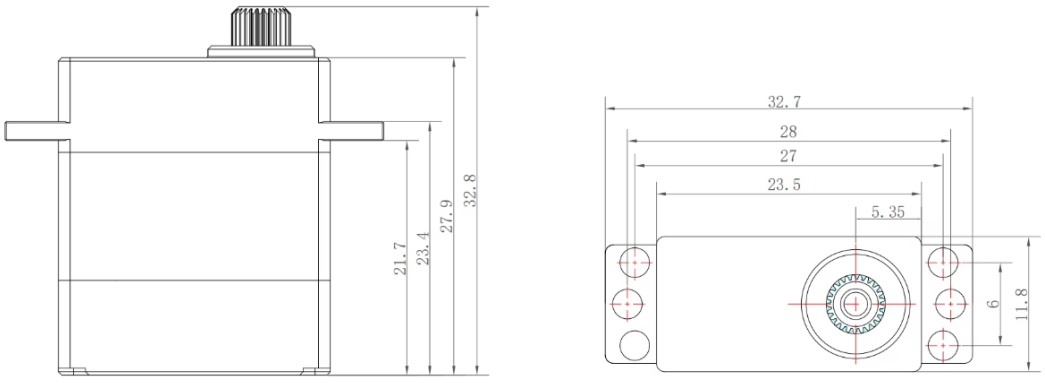


Figure 3.36: Servo Motor: dimensions.

Regarding the performance of the *Servo Motor*, a *no-load speed* of $0.076\frac{s}{60°}$[20] is observed, achieved at

[3]Pulse Width Modulation

the maximum stall torque of $7.6kg \cdot cm$[20], obtained at the maximum supply voltage.

This component, made of metal, was preferred due to its significantly small size and low weight. In fact, the *Servo Motor* has the following dimensions and weight:

- Dimensions: $23.8 \times 27.6 \times 11.3\ mm$[20];
- Weight: $16.7\ g$[20].

It should be noted that the housing made in the *Flapping Wing* still has larger dimensions to accommodate, if necessary, larger components with potentially better performance. Given the requirements of the current experiment, this *Servo Motor* is considered perfectly suitable.

For the current experiment, the *Servo Motor* needs to move within a range of 80°, particularly between $-40°$ and $+40°$ relative to the horizontal plane of the *Flapping Wing*. Since the *Servo* is fixed in position, attached to the *Flapping Wing* using hot glue, this fixed position is assigned as the 0° angle, and the *Servo*'s movement is regulated with this position as the reference.

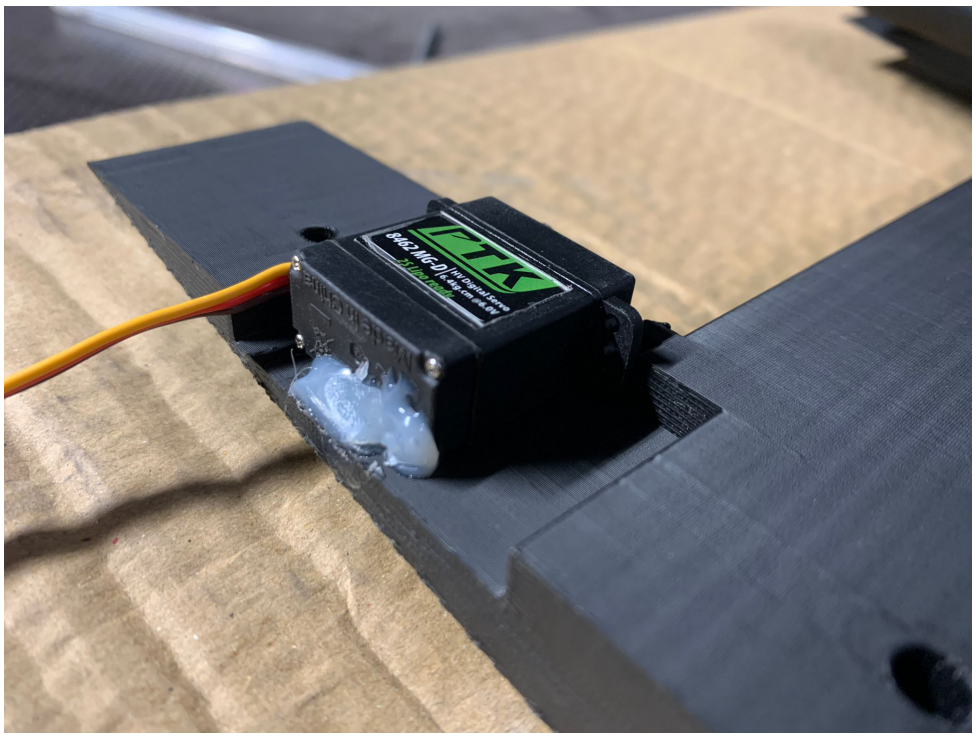


Figure 3.37: *Servo PTK-8462* used.

3.3 Boards

3.3.1 Teensy 4.0

Teensy 4.0 is a compact powerful micro-controller designed for electronics and programming projects. It is based on a *NXP i.MX RT1062* chip, a 32-bit micro-controller with a clock speed of up to 600 MHz, making it one of the fastest boards in its category. It is compatible with the *Arduino IDE*.

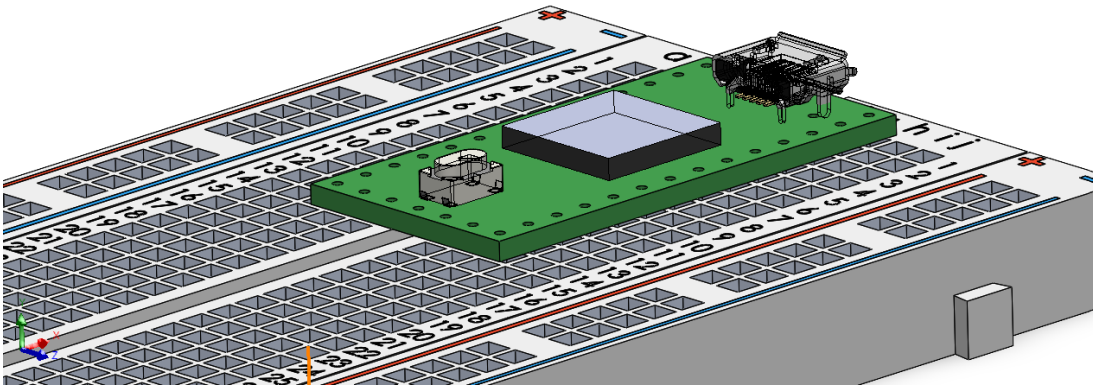


Figure 3.38: Teensy 4.0.

Teensy 4.0 has a wide variety of input/output (I/O) pins that allow to connect sensors and other electronic devices. Additionally, it supports advanced features such as USB communication, audio processing, and graphics, and is used in applications that require high performance, such as audio device control and robotics.

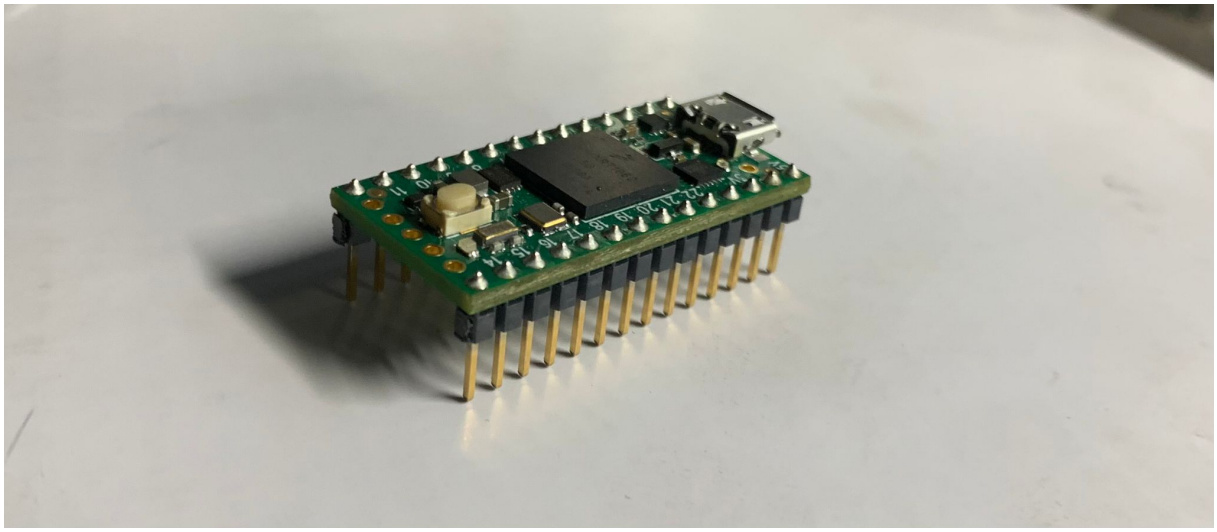
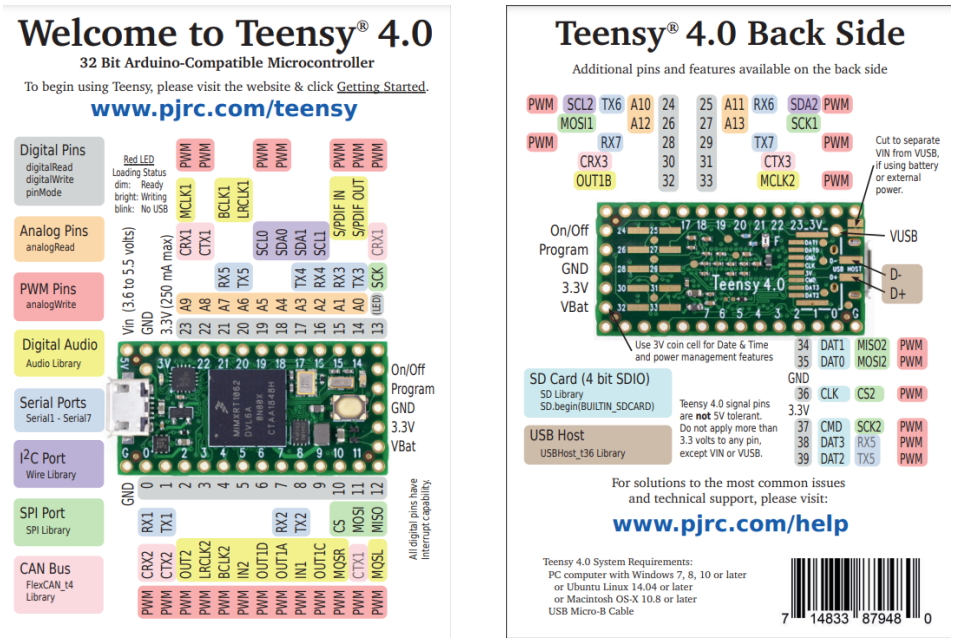


Figure 3.39: *Teensy 4.0* used.



(a) Front Page. (b) Back Page.

Figure 3.40: Teensy 4.0 Pinout[21].

Key features include:

- 600 MHz clock speed;
- 1024 KB of SRAM[4];
- 2048 KB of flash memory (64K reserved for recovery & EEPROM emulation)[21];
- Support for USB host and USB device;
- Potential audio and video processing capabilities.

This control module will also be used for its compact size, as it will be placed directly inside the *Flapping Wing*, enabling a simpler and more effective configuration for acquiring data from the two *Honeywell RSC* sensors. It is worth noting that the choice of this *board* followed several attempts, such as the idea of using a single module for both sensor control and *Servo Motor*, which was discarded due to the inability to achieve high-frequency responses. Another option, before using the *Teensy 4.0*, was the *Arduino R4* board. However, due to size constraints, it would not have been possible to house it directly inside the structure and it would have required the design of a jumper cable system to carry the signal from the *BreadBoard*, where the sensors are connected, to the outside of the structure, where the *Arduino* would have been located.

3.3.2 Raspberry Pi Pico

The Raspberry Pi Pico is a low-cost, high-performance micro-controller board with flexible digital interfaces. This component is as follows:

[4]Static Random Access Memory

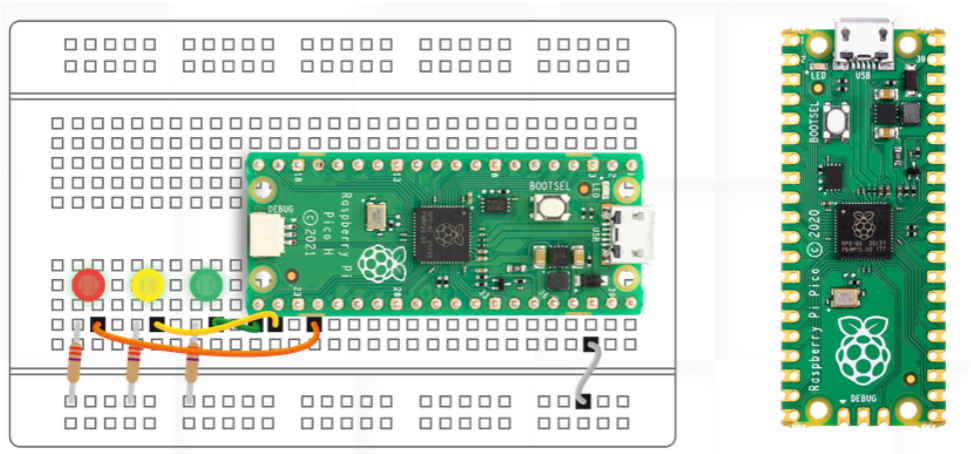


Figure 3.41: Raspberry Pi Pico.

The main features include[22]:

- RP2040 micro-controller chip designed by Raspberry Pi in the UK;
- Dual-core *Arm Cortex M0+* processor with a flexible clock speed up to 133 MHz;
- 264 KB of $SRAM$ and 2 MB of on-board flash memory;
- USB1.1 support for both device and host modes;
- Low-power modes such as *sleep* and *dormant*;
- *Drag-and-drop* programming[5];
- 26 multifunctional GPIO[6] pins;
- 2 SPI, 2 I2C *Inter-Integrated Circuit*, 2 UART[7], 3 ADC[8] at 12 *bit*, 16 controllable PWM channels;
- Integrated clock and timer features;
- Temperature sensor;
- Integrated floating-point libraries for fast calculations.

The Raspberry Pi Pico comes as a castellated module, allowing for direct soldering onto support boards[22].

[5]Drag-and-drop: This refers to the process of moving a file from one window to another on a computer. In this case, it involves dragging the program file (typically a .uf2 file for the Raspberry Pi Pico) and dropping it into the microcontroller's memory.

[6]General Purpose Input/Output

[7]Universal Asynchronous Receiver/Transmitter

[8]Analog-to-Digital Converter

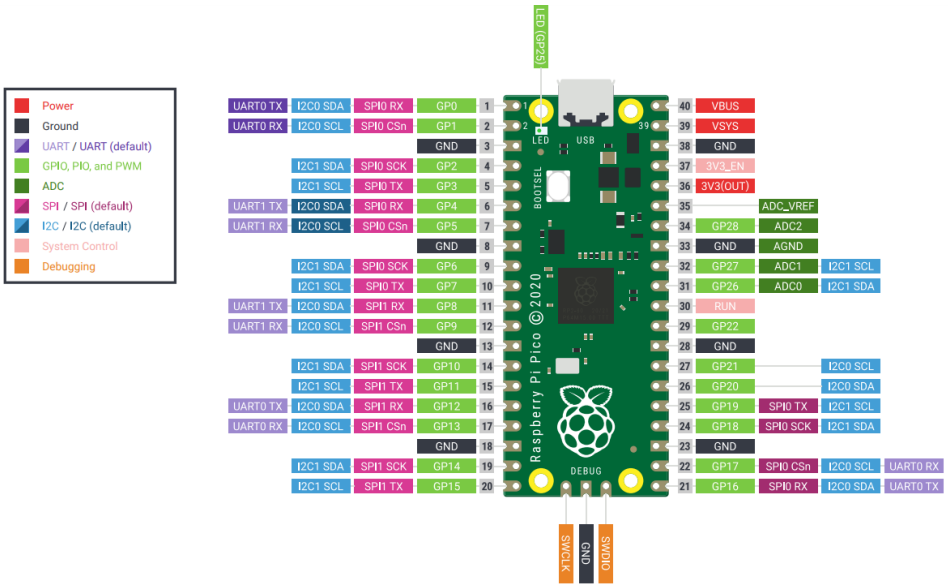


Figure 3.42: Raspberry Pi Pico Pinout[22].

This component is placed outside the *Flapping Wing* and is responsible for controlling the *Servo Motor*, which governs the movement of the *Flap*. The connection between the *Raspberry Pi Pico* and the *Servo Motor* is made via jumpers. Since the *Raspberry Pi Pico* cannot supply more than 3.3 V of power, an external power supply will be used, as mentioned earlier, to provide 7.2 V for optimal performance of the *Servo Motor*.

To control the *Servo Motor*, the *RP2DAQ*[9] project was used. This project includes both pre-compiled firmware and an easy-to-use Python module for control. The firmware handles all the technical micro-controller tasks, including managing parallel tasks and ensuring reliable communication, and is optimized to make the most of the Raspberry Pi's performance. All actions of *RP2DAQ* are triggered by a Python script on the computer, which eliminates the need for the user to program in C and deal with hardware debugging, which can be error-prone[23].

[9]Raspberry Pi Pico for Data Acquisition

3.4 Wind Shaper

The instrument used to simulate gust conditions for the current experiment is the *Wind Shaper*. The *Wind Shaper* is a cutting-edge multi-fan wind generator that allows laboratory environments to precisely replicate and control wind conditions, revolutionizing the way wind testing and research are conducted. With its unrivaled capabilities, *Wind Shaper* takes wind simulation to new levels, enabling a world of possibilities[24].

Figure 3.43: *Wind Shaper.*

This device allows, through a simple and intuitive interface, to replicate flow conditions at a given speed, direction, and potentially with a defined mathematical law. Specifically, for the current experiment, this tool will be used to reproduce a sinusoidal gust condition with a defined period and amplitude.

Chapter 4

Testing

After the design and the *Flapping Wing* assembly phase both the data acquisition and the *Flap* movement system have to be checked. For this purpose, as a first step, it is necessary to try all these systems separately. First, the pressure data acquisition will be tested, focusing on the time per acquisition, the response in frequency of the system and the reliability of the data. Then, the *Servo Motor* will be tested, checking its reliability and the timing, to make sure it works as fast as possible to reach an efficient setup. Once both systems (data acquisition and movement) are tested it will be possible to check a merged-version of them, focusing on timing and data reliability. It is reminded that in the final setup configuration the *Hot Wire* voltage signal is acquired by a *NIDAQ* console that sends data at one serial port of the *PC*, so the timing and the frequency of this data acquisition are known. However, also the *Hot Wire* anemometer will be tested to make sure the data are reliable and no issue occurred.

It is mandatory to repeat the experiment's purpose. After all the setup assembly and preliminary test, a *Reinforcement Learning Algorithm* will be tried, with which it will be possible to *predict* and *actuate* a precise *Flap Movement* to control the *periodical gust effects*. This control will consist in a particular *Flap Movement* that will control the aerodynamic oscillation due to the artificially generated gust. How this algorithm works, and which variables are used, will be explained in detail in the next sections.

At the beginning of every measurement it is necessary to perform the *Hot Wire* anemometer calibration due to its strong dependence on the environmental conditions as mentioned in the previous chapter.

4.1 Hot Wire Calibration

Before the *Hot Wire* calibration it is necessary to check that the probe itself is working and can acquire data. The data acquisition from the *Hot Wire* is demanded to a *NIDAQ*[1] console, a precise console for data acquisition.

At the beginning of all measurements it is mandatory to *calibrate* the *Hot Wire Probe*. For this purpose, the probe is collocated slightly above the *Flap* symmetry plane, moving on the vertical guide to measure precisely the airflow velocity and record all the data for the calibration. Obviously, this position is necessary in order for the probe not to interfere with the flap and to record the same data that would have been recorded without the *Flapping Wing* presence.

For the calibration, it is necessary to record as many airflow speed configurations as possible. For this purpose, eight airflow configurations will be tried, imposing the airspeed value using the *Wind Shaper*. Particularly, only configuration with constant airspeed will be used, for a constant amount of time. Eight different power percentages will be tried, specifically: 0%,10%,15%,20%,25%,30%,35%,40%,45%.

As said in the previous chapters, the *Hot Wire* probe measures the airflow speed by a sensible element shown as a *Hot Wire*, which cooled by the airflow sends an electrical signal in *Volt*, associated to a particular speed in $\frac{m}{s}$. The airflow velocity is measured by a *5 holes probe* located near the *Hot Wire* to collect

[1] National Instruments Data Acquisition

realiable data. The *5 holes probe* signal is processed by a DSA^2 which will return velocity values in the correct measure unit. These values will be compared with the *Hot Wire* signal acquired with a *NIDAQ* console.

Once all the data is collected and processed, after computing the *mean* value of all the *Hot Wire* signals for all the airflow configurations used, it is possible to obtain the *calibration law coefficients*. For the calibration, despite all of the possible formulas mentioned in the *Hot Wire Anemometry* section, it is preferred to use a *4th Order Polynomial Law*:

$$U = aE^4 + bE^3 + cE^2 + dE + e$$

Comparing *Hot Wire* and *5 holes probe* processed data it is possible to find the a,b,c,d,e coefficients that will allow the conversion of the *Volt* values in $\frac{m}{s}$.

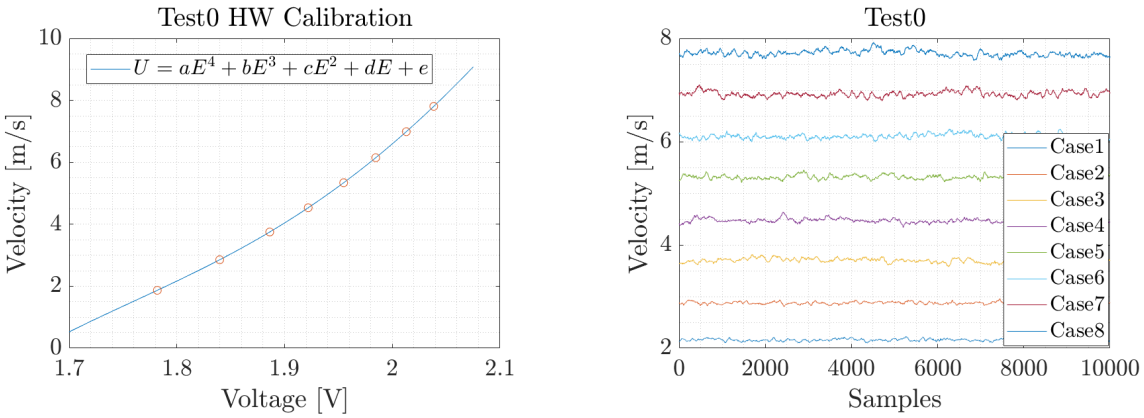


Figure 4.1: Hot Wire Calibration.

By processing the calibration data it is possible to obtain the velocity data corresponding to the different power percentages of the Wind Shaper:

Wind Shaper %Power-Flow velocity		
Case	Power Percentage %	Velocity $[\frac{m}{s}]$
1	10	1.90
2	15	2.85
3	20	3.75
4	25	4.50
5	30	5.35
6	35	6.15
7	40	7.00
8	45	7.80

Table 4.1: Power Percentage-Velocity

4.2 Data Acquisition

4.2.1 Honeywell RSC

Once the *Hot Wire* functioning is confirmed and once the probe calibration is completed the *Honeywell RSC* functioning has to be checked. Note that the pressure data acquisition from the *Honeywell RSC* sensor is demanded to the *Teensy 4.0* board inside the structure. Some preliminary test will be operated

[2]Dynamic Signal Analyzer

before the experiment, only to check the pressure data reliability, acquired using a simple *C* script, launched by *Arduino IDE*. In fact, this interface is compatible with the used board .

Honeywell provides a particular library called *honeywell.h* for this purpose. By using this library it will be possible to acquire data directly in *Pascal*. Note that *Teensy 4.0* allows the *parallel acquisition* of the two sensors, thus allowing to save time and showing high performances. Another library is used to manage the *SPI* interface, mentioned in the *Honeywell RSC* section: *spi.h*. By using this library it will be possible to assign all the *SPI* functions to particular pins on *Teensy 4.0*.

For the sake of completeness, it must be said that the final architecture for the *Data Acquisition* system has passed through several modifications to reach the best configuration and optimize spaces, performances and results at best. In fact, the first attempt was based on the use of an *Arduino* board, even though it was not possible to obtain parallel processes. In this first attempt, a single *Arduino* board acquired data from the two pressure sensors linked to it. Another *Arduino* board was demanded to acquire data from the *Hot Wire* anemometer. This solution, by giving problems in terms of time required to do the operations, did not permit to insert the board inside the structure requiring a cable system to let the signal go from the sensor to the board. For this reasons in the final configuration we chose to acquire data from the pressure sensor by using a *Teensy 4.0* board, smarter and smaller as we can insert it in the structure. For data acquisition from the *Hot Wire* we chose, in the end, to use the *NIDAQ* console.

Once checked the *pressure* data reliability in *Pascal [Pa]* it will be possible to check the *Servo Motor* functioning too.

Below plots, made from pressure data acquired in gust conditions, are shown. This plots demonstrate how these sensors can acquire reliable data, showing the sinusoidal gust trend.

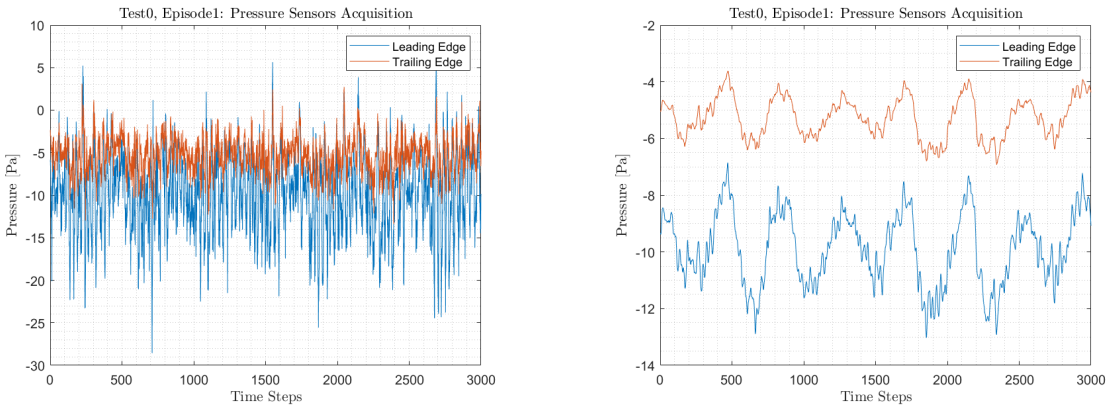


Figure 4.2: Pressure Data Acquisition.

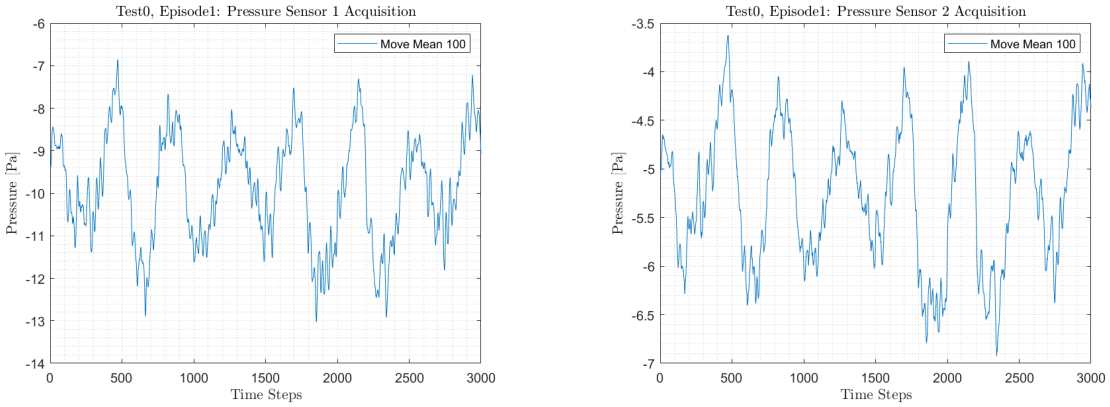


Figure 4.3: Pressure Data Acquisition.

4.2.2 Servo Motor

The *Servo Motor* is controlled by a *Raspberry Pi Pico* board, mentioned in the previous chapter. A particular library for *Raspberry* board will be used, allowing to control the *Servo Motor* and to manage the data[23]. At this point, we simply proceed by checking the *Servo Motor* movement, ignoring all the probes data.

By using the *Rp2daq* library[23] it is possible to configure the *Servo Motor* in a particular *Raspberry Pi Pico* pin. It is noted that the *Servo* shows three exit pins, one for the *DC* alimentation, one for the *GPIO pin* and one for the *ground* signal. All the *Servo* parameters, as the frequency, the maximum and minimum pulse width, have to be declared. In this case these values are set at:

Servo Motor Parameters	
Frequency [Hz]	190
Max Pulse Width [V]	0.58
Min Pulse Width [V]	0.21

Table 4.2: Servo Motor Parameters.

The library used allows to set a particular *duty value* to control the *Servo*, based on the *Duty Cycle*[3]. The *Servo* works by *PWM*, modulating the signal pulse width keeping the frequency value constant. So, by using this technique, it is possible to control the *Duty Cycle*. To gain a *PWM 16-bit resolution* it is imposed a *wrap value* of 65535.

To check the*Servo Motor* functioning, a list of 100 random angle values, varying in the flapping wing future working range, will be generated. This values will be converted into a *PWM duty value*:

$$pulse\_width = min\_pulse\_width + \frac{angle}{range} * (max\_pulse\_width - min\_pulse\_width)$$

$$duty\_value = int(pulse\_width * frequency) * wrap\_value$$

All these random angles, once converted in *duty value* will be applied to the *Servo*. At this point a crucial control has to be applied to the *Servo* operations and the time required. This control is done by printing the time between two different *Servo* actions and the *mean* time over the total amount of angles given.

Once the *Servo Motor* and the *Raspberry Pi Pico library* functioning is verified, it will be possible to merge the *Servo* control and the *Data Acquisition* function to check how much time this operations will require.

4.2.3 Merging Servo-Sensors

At this point, it is possible to merge the sensors and *Servo* functioning, setting a list of random angles and acquiring data from these sensors for all *Servo* moves. During the design phase an Arduino board had been chosen for data acquisition. This board could not do parallel processes so there was the necessity to find a way to speed the processes a little bit. For this purpose it was preferred to use a *Socket* technique to create a *Server-Client* model between *Raspberry Pi Pico* and the *Pc-Arduino*. By working in this way it is possible to obtain higher frequency and very low time spent doing operations.

Socket is a specific mechanism which allows the communication between devices. In this case we prefer to allow communication between a *Server*, who performs as the *Servo Motor* controller, and a *Client*, who manages the data from the sensors and processes them, sending commands for the *Servo Motor* to the *Server*. The entire operation proceeds in this way:

- the *Server* creates a *Socket*, which acts like a physical cable to which the *Client* will connect;
- the *Client* connects to the *Server* using the *Socket*;

[3]Percentage of time while the signal is *High* on the total cycle time.

- once the connection is obtained, the *Client* creates a 100 random angles list and, for each of them, it sends a command to the serial port in which *Teensy 4.0* is linked. This command starts the data acquisition. Then, for each angle, the *Client* sends the angle value to the *Server* using the *Socket*, allowing the *Servo Motor* movement. Data acquisition and saving are managed separately using two different threads to save time and increment the speed of the operations. So, in two different threads the *Client* reads *n_data* values for each sensor and save them in a *.csv* file.

At the end of the cycle, a *.csv* file which recorded the entire time history of the process, will be built. For each angle, also the time occurred to complete one single move and the acquisition of *n_data*, will be printed. It is expected time interval of 1 ms. At the beginning, we tried to acquire data from the *Hot Wire* probe by using an *Arduino* board, which would send data to the serial port to be processed in the way explained above, exactly as done with the pressure sensors. However, *NIDAQ* console was found to be more efficient for this purpose, so for the final experiment it was preferred to use this dedicate data acquisition system for *Hot Wire*.

From this preliminary test, we demonstrate that in a small amount of time it is possible to acquire and store a small amount of data and move the *Flap* of an angle in the range of work, which is $[-40°, +40°]$.

Then, by choosing a *Teensy 4.0* it has been possible to acquire and save data in two separate threads by doing these processes in parallel. For this reason the *Socket* technique has been discarded. Both *Teensy 4.0* and *Arduino* allowed to acquire data at high frequency. Particularly, the combination of *Teensy 4.0* for pressure data acquisition, *Raspberry Pi Pico* for *Flap* motion and *NIDAQ* console for *Hot Wire* data acquisition showed the best results in term of efficiency, time per move and response in frequency.

This preliminary test has been done by setting a *max time steps* value to 3000. By acquiring one data from each sensor and moving the flap by a random angle it has been possible to collect the results below.

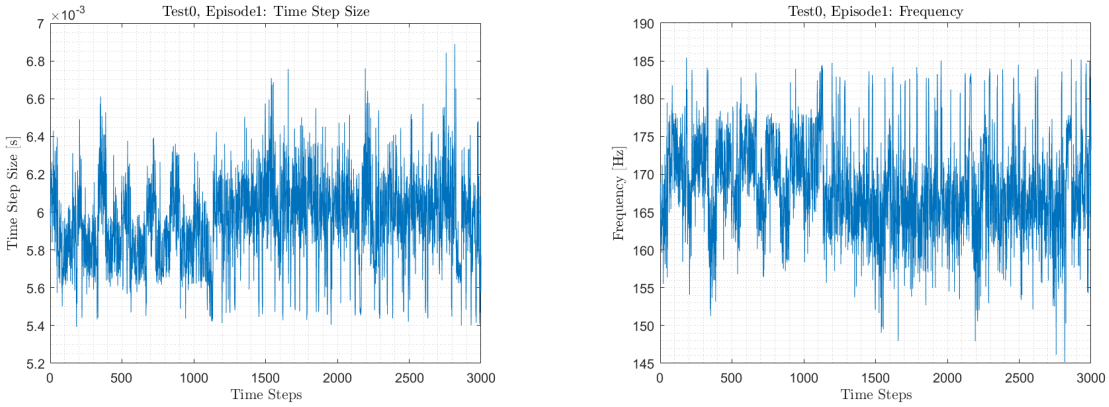


Figure 4.4: Time per move and response in frequency.

This performance show:

$$mean\ frequency = 168\ Hz$$

$$mean\ time\ per\ move = 0.005968\ s$$

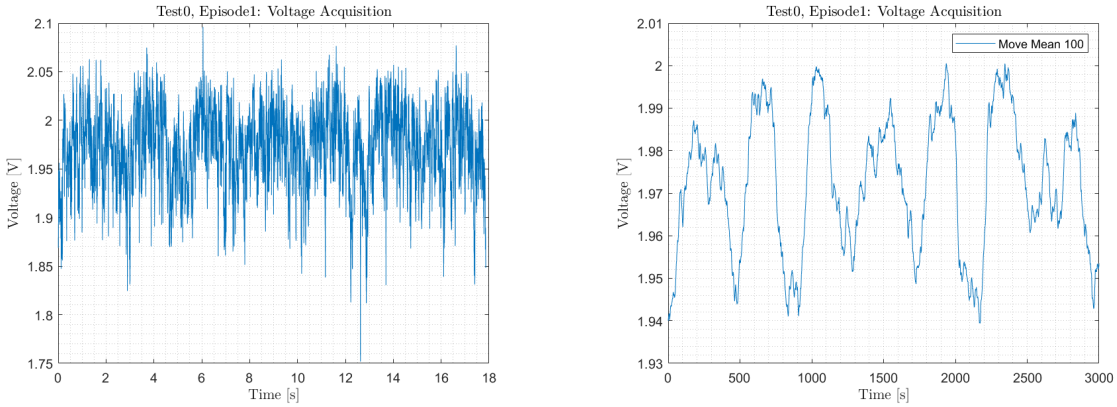


Figure 4.5: HW Voltage Acquisition.

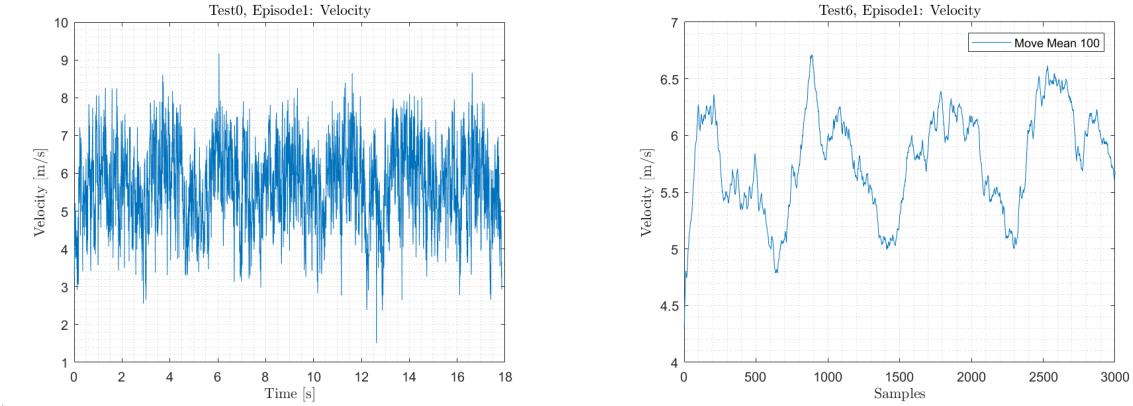


Figure 4.6: Flow Velocity.

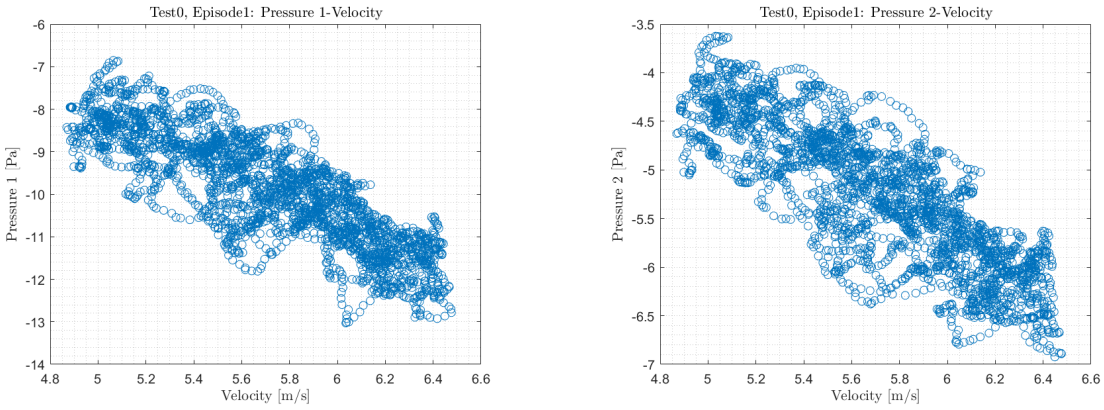


Figure 4.7: Pressure vs Velocity.

All these plots are derived from a preliminary test. By looking at the plots above it is possible to verify the reliability of the acquired data. These plots show perfectly the sinusoidal gust in terms of voltage (and, so, velocity) and pressure. In addition these plots show the pressure changes with the airflow velocity. In particular, it can be noticed that the pressure on the *Flapping Wing* surface decrease with the airflow increasing in the wake region. This result is crucial to verify the physical reliability of the data. The time step size and the frequency plots confirm the reliability of the setup. This fantastic results lead to the possibility to try a *Reinforcement Learning Algorithm* to process the acquired data and predict the *Servo Motor* movement in a small amount of time to minimize the effect of the gust.

4.3 Reinforcement Learning Algorithm

4.3.1 Reinforcement Learning Introduction

Once the entire setup has been tested and it has been noted that the flap response and the data acquisition are fast enough, it is possible to use the *Flapping Wing* as a tool for developing and testing machine learning algorithms.

Before explaining in detail this experiment, for the sake of completeness, it is good to show how a *Reinforcement Learning Algorithm* works. *Reinforcement Learning* (RL) is a particular area of machine learning and optimal control. His logic is based on actions done by looking at the actual state to maximize a reward variable. As a machine learning algorithm, based in fact on a vast amount of data, this method aims to make the object learn from experience and act without any command sent to it. It is necessary to explain more in detail the mechanism.

The RL approach is much more focused on goal-directed learning from interaction with respect to other approaches to machine learning. In fact RL problems involve learning what to do and how to map situations to actions so as to maximize a numerical reward signal.[25]. Due to the influence of the state on the final output, these types of problem are defined as *closed loop*. This way of proceeding with actions that maximize a reward given by the environment is called *Markov Decision Process* (MDP). Reinforcement Learning Problems are defined in MDP as in the figure below:

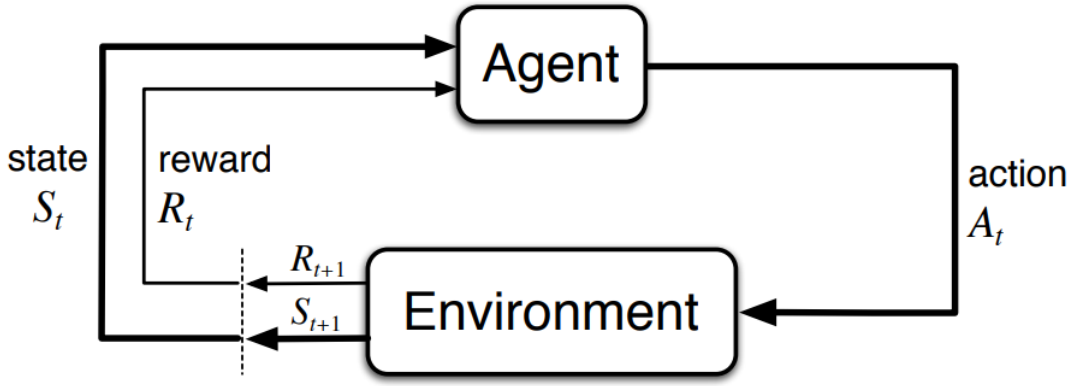


Figure 4.8: Markov Decision Process (MDP).

Reinforcement learning is different from supervised learning, the kind of learning studied in most current research in field of machine learning. In fact, supervised learning is learning from a training set of labeled examples provided by a knowledgable external supervisor. Reinforcement learning is also different from what machine learning researchers call unsupervised learning, which is typically about finding structure hidden in collections of unlabeled data[25]. In the Reinforcement Learning problems there are not data available and labeled provided, all the data are provided by the interactions with the environment and the learning is obtained by trying different actions to maximize a parameter (reward). For this reason, the agent needs to *explore* all the actions first, in order to find the best which maximizes the reward, so it can use and *exploit* this actions later.

Another key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an *uncertain environment*. This is in contrast to many approaches that consider subproblems without addressing how they might fit into a larger picture[25].

4.3.2 Elements of Reinforcement Learning

The main components of an RL problem have been previously mentioned: *agent* and *environment*. However, it is possible to recognize four elements of an RL problem: *a policy*, a *reward signal*, a *value function* and an eventual *environment model*.

The term *policy* refers to the way the learning agent behaves at a given time. In other words a policy defines a mapping of the environment in the previous state to chose the best action for the current state.

The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine the behaviour[25].

The *reward* is defined as a variable given to the actor from the environment at every new state, based on the action done and its goodness. The only objective of the actor is to maximize that variable over the long run. The reward shows the goodness of the previous action without giving any information about the long run.

For this purpose a *value function* is defined . This function specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state[25].

The *environment model*, optional in an RL problem, gives some inference on how the environment behaves. The existence of the model allows to predict the next action and reward. So *Environment models* are used for *planning* and eventually chose actions not yet experienced by predicting the behavior of the environment itself.

4.4 Gust Mitigation Application

Once built and tested, the *Flapping Wing* setup is used as a tool for algorithm and codes validation. For this purpose it is provided a RL Learning Algorithm for *gust mitigation*. Basically, this algorithm uses the pressure sensors to visualize and record the flow state to which a particular *Flap* angle is associated. In this state, the *Hot Wire* probe records a particular voltage value and compares it to a reference voltage value acquired in no-speed flow conditions. The goal of the algorithm is to move the *Flap* rapidly to control *in time* the *Shedding Phenomenon* due to the presence of the setup. This phenomenon could be visualized by comparing the *Hot Wire* value with the *reference voltage value* trying to minimize the squared difference.

This algorithm is classified as an *off-policy* algorithm. *On-Policy* learning Algorithms are the algorithms that evaluate and improve the same policy which is being used to select actions. That means we will try to evaluate and improve the same policy that the agent is already using for action selection. *Off-Policy* learning algorithms evaluate and improve a policy that is different from Policy that is used for action selection. Among *Off-Policy* methods are[26]:

- Continuous exploration: As an agent is learning other policy then it can be used for continuing exploration while learning optimal policy. Whereas On-Policy learns suboptimal policy;
- Learning from Demonstration: Agent can learn from the demonstration;
- Parallel Learning: This speeds up the convergence i.e learning can be fast.

4.4.1 Environment

The algorithm uses a *Gymnasium* to simulate the environment where the *agent* can be trained. *Gymnasium* is an open-source toolkit for developing and testing reinforcement learning (RL) algorithms. It is the successor to *OpenAI's Gym* and is maintained by the *Farama Foundation*. Gymnasium provides a collection of environments where RL agents can be trained, including classic control tasks, Atari games, robotic simulations, and more[27]. All reinforcement learning (RL) environments in *Gymnasium* follow the same structure and function names, making it easy to switch between different environments without changing the RL algorithm.

This structure consists of three main functions:

- *reset()*, which resets the environment and returns the initial observation;
- *step()*, which executes the action and returns: *obs*, *reward*, *terminated*, *truncated*, *info*;
- *close()*, which closes the environment when done.

In the case of the *Flapping Wing*, a personalized environment class for *Gymnasium* has been created.

The first function of the class handles the initialization of variables. In this function, variables such as ports, baud rate, path for saving, and servo parameters are initialized. Then, another function is created to open a connection to the serial port, from which all the data from the sensors arrives. The *action space*, which defines the range of action values, is also created. Finally, the *Offset* is computed using a specific function. This function calculates the *Offset* by acquiring data from three sensors (two pressure sensors and one voltage measurement from the Hot Wire anemometer) 1000 times, then parses them and computes the mean values. These values are initially saved into a dictionary called *data_ buffer* and later into a *.csv* file. Once the offset is calculated, the connection to the serial port is closed using another dedicated function.

The *reset()* function initializes the serial connection using the previously mentioned function and acquires 3000 data for each sensor in the same manner as the *Offset* acquisition process. Before acquiring data, the *Flap* is set to the zero-angle position to measure the gust values without interference. A dictionary is created to store *state*, *action*, *reward*, *next state*, and the *done* flag. Using a specific function, pressure and voltage data are acquired from the serial connection and stored in a buffer, which also includes the acquisition time history. The previously computed *Offset* values are subtracted from the pressure and voltage values in the buffer. The mean pressure values are computed and stored in an *observation* array, and the mean voltage value is also calculated. If a flag called *action_in_state* is set to *True*, the action value, which at this point is 0, is also saved in the *observation* array. At the end of this phase, the mean voltage is assigned to the V_{ref} variable, and the *observation* array is stored in the dictionary under the *state* key.

The *step()* function is the core of the reinforcement learning algorithm. During each iteration, the agent selects an *action* based on the *state* value, which corresponds to a *Flap* angle. This action is then normalized to ensure it stays within the *action space* range ([−1,1]). Once the action is determined, the servo is commanded to move to the specified angle, and new data from the sensors (pressure and voltage) are acquired. The mean value of the new pressure and voltage data is computed and stored in specific variables. These data may be filtered to reduce noise in the measurement. For each step, 4 values for each sensor and the corresponding time history are acquired and stored.

The reward for the action is calculated based on the voltage difference between the measured voltage (E_{filter}) and a reference value (V_{ref}), which was established during the reset phase. The error between the two is squared and made negative. The reward is computed as:

$$reward = -(E_{filter} - V_{ref})^2$$

A smaller difference results in a higher reward, indicating that the action was closer to the desired outcome.

The system updates the *observation* array with the latest sensor data, and if the *action_in_state* option is enabled, the selected action is also included in the state. In this case, the *observation* array will consist of pressure sensor data and the action value. Data such as *reward*, *next state*, *action*, and the *done* flag are then stored for training purposes. If the maximum number of steps (*max_timestep*) is reached, the episode ends, and all the data is saved. A function to update the episode number (*update_num_episode()*) and create the corresponding files is provided. Otherwise, the process continues to the next step.

At the end of each step, the temporary *state*, *reward*, *next state*, *action*, and *done* flag are reset, and the updated *observation* array is stored in the dictionary under the *state* key. This variable is expected to be used by the agent to compute the *action* value.

In summary, the process follows a series of steps in which the agent makes decisions based on the current observation. It selects an action that adjusts the flap's angle, and this action is normalized to fit with the action space. The environment then executes this action, moving the flap accordingly. New sensor readings, including pressure and voltage, are gathered, and if selected, a filter is applied to minimize data noise. The reward is computed by comparing the measured voltage to the reference value (V_{ref}), with a smaller difference leading to a better reward. The observation is updated with the latest sensor data, and if the *action_in_state* option is active, the selected action is also included in the state, providing more information. The system then checks whether the episode has ended by verifying if the maximum number of steps has been reached. If so, the episode concludes, the data is saved, and the environment is reset for the next episode. If not, the agent continues the process, refining its actions based on the feedback received.

4.4.2 Train

The *train* phase is responsible for training the Reinforcement Learning (RL) agent to control a *Flap*. The agent interacts with the environment, acquiring data from sensors and adjusting the flap angle to optimize the system. The algorithm used is Soft Actor-Critic (SAC), which balances exploration and exploitation, progressively improving the agent's decisions.

Once the environment is created, the SAC agent is initialized. This agent receives the state and action dimensions, the device on which calculations will be performed (in this case, the CPU), learning rate, gamma, and tau. If resuming from a previous training session, saved model weights and replay memory are loaded.

At this point, the training loop begins. For each episode, the environment is reset, and the agent receives the initial *state*. In the step loop, the agent decides which action to take. If enough data has been accumulated in the replay memory, the agent uses the learned policy to select the most effective action; otherwise, it selects a random action. The chosen action is then passed to the environment, which returns the new state, reward, and an indication that the episode has ended.

The reward is calculated based on the squared difference between the filtered voltage and the reference value. The agent stores this experience in the replay memory, which will be used to update the neural network weights. If the maximum number of steps is reached or the environment signals the end of the episode, the episode score is saved, and the agent updates the model weights using the replay buffer. The score for each episode is defined as:

$$score+ = reward$$

After the training phase, the agent evaluation begins. In this phase, the agent no longer explores but executes actions based on the learned policy. Thirty test episodes are conducted, collecting scores and saving them for future analysis.

Finally, all data is saved in .csv files, including training and evaluation episode scores. The updated neural network models are also saved for reuse in future training sessions or further optimization.

4.4.3 Agent

Soft Actor-Critic (SAC) is an algorithm that optimizes a stochastic policy in an off-policy manner, forming a bridge between stochastic policy optimization and DDPG-style[4] approaches.

A key feature of SAC is entropy regularization. The policy is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy. This is closely related to the exploration-exploitation trade-off: increasing entropy results in more exploration, which can accelerate learning later on. It also helps prevent the policy from prematurely converging to a poor local optimum[28].

In this case the SAC agent is initialized with all its variables and parameters. A *reply_buffer* to store the experiences is created. The *act* function chooses the action based on the agent current policy making a trade-off between exploration and exploitation. During the test phase there is no exploration. After every episode the agent *learns* by updating its policy before starting the next episode.

4.4.4 Results

The RL Algorithm has been tested in four different cases, each one in different gust conditions, Then the results in terms of *Pressure 1*, *Pressure 2*, *Airflow Speed* and *Actions* are plotted. The four gust conditions are summed below. The amplitudes and the offsets are expressed in term of WS power percentage.

[4]Deep Deterministic Policy Gradient is an off-policy reinforcement learning algorithm used for environments with continuous action spaces.

Gust Sinusoidal Laws	
Case	$Law\ [Asin(\frac{2\pi}{T}t) + c]$
1	$30 + 5sin(\frac{2\pi}{5}t)$
2	$30 + 5sin(\frac{2\pi}{2.5}t)$
3	$30 + 5sin(\frac{2\pi}{5}t)$
4	$30 + 5sin(\frac{2\pi}{2.5}t)$

Table 4.3: Gust Sinusoidal Laws

Pressure

Getting more in detail with the plots it is possible to notice, for each case, the variables in the first and the last episode of the corresponding test. Particularly, *Pressure 1* and *Pressure 2* are plotted combining, for the mentioned episodes, the acquired data in the *reset* step (*Unforced*) and the acquired data while the *Flap* is moving (*Forced*). By using this plotting strategy it is possible to notice the effects of the *Flap* movement for gust mitigation by comparing the data from the two different steps.

By looking at these plots, it is clear there is no difference between the *Forced* and the *Unforced* phase. This is obviously predictable due to the position of the pressure sensors in the *Flapping Wing* structure. In fact, the pressure values acquired by these sensors do not change within the two phases due to the position of the pressure taps on the wing profile where the *Flap* movement has no effects. However it is possible to notice the perfectly periodic trend of the pressure values due to the gust presence.

Airflow velocity

Then, the velocity plots show huge changes of this variable between the *Forced* and the *Unforced* phase. This phenomenon is due to the effect of the *Flap* movement which optimizes the gust mitigation process. However, this changes are obviously more visible in the last episodes because the training of the agent leads to better mitigation and smarter moves. In fact, comparing velocity *Forced* in the first and in the last episode it is shown the weak effect of the agent before and the strong effect in the last episode, verifying the reliability of the setup and of the algorithm. It is possible to obtain a huge decreasing in the gust amplitude, verifying the reliability of the agent. Despite the gust values attenuation, the periodical trend remains visible in these plots.

Actions

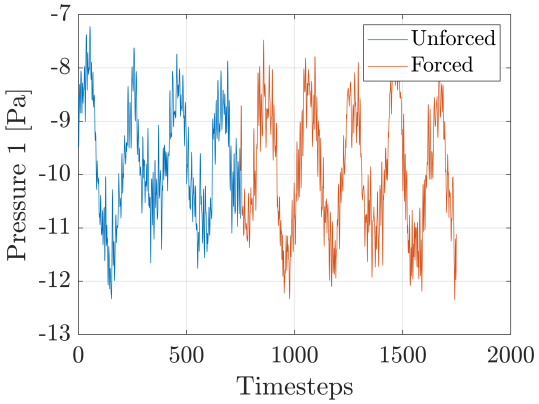
The *action* values are numbers between -1 and 1 due to the *normalization* and *re-normalization* functions previously mentioned . In fact, every *action* taken by the agent is previously converted into a number in the [0,1] range, then processed and, in the end, re-normalized.

The first plots show the *Pressure 1* values for each *action* value. It is possible to notice how randomly these *actions* are done in the first episode compared with the specific trend shown in the last episode plot. In fact, in the last episode, actions are taken in a specific way, obtaining precise *Pressure 1* values.

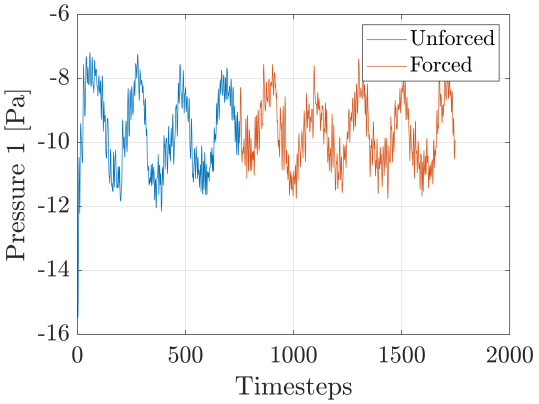
In the second plots, the *Action-Time Steps* trends between the first and the last episode are compared. These plots show the periodical occurrence of actions corresponding to the periodical gust in the last episode. In the first episode, as predictable, the actions are taken randomly among the entire time history due to the weak training done.

The last plots show the *scores* decreasing during all the episodes of a single test, reaching the asymptotic conditions, corresponding to the minimum reward value This trend verifies the algorithm reliability and the test success.

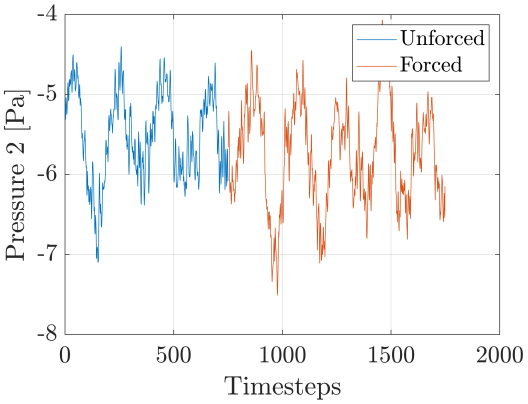
Case 1



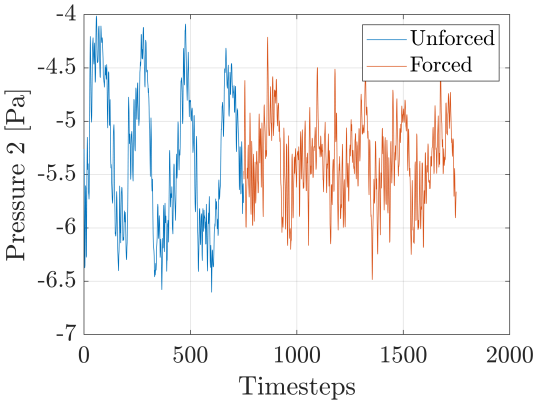
(a) First Episode: Pressure 1.



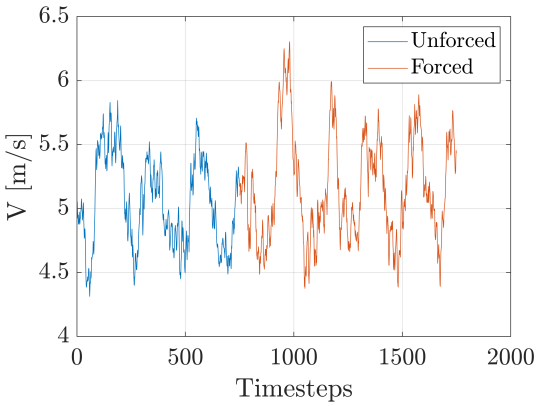
(b) Last Episode: Pressure 1.



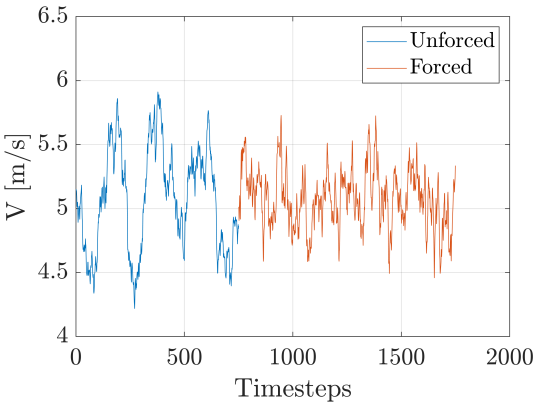
(c) First Episode: Pressure 2.



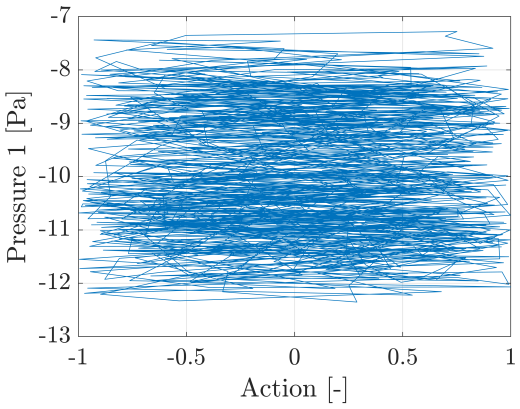
(d) Last Episode: Pressure 2.



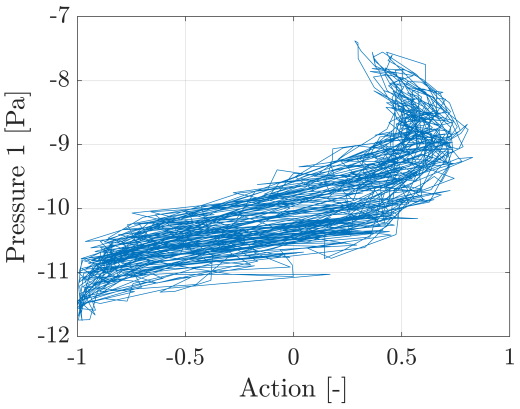
(e) First Episode: Velocity.



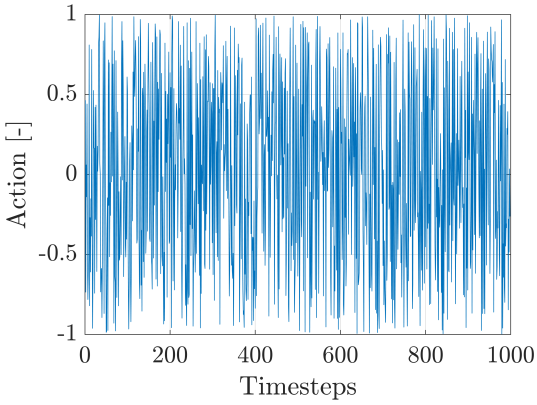
(f) Last Episode: Velocity.



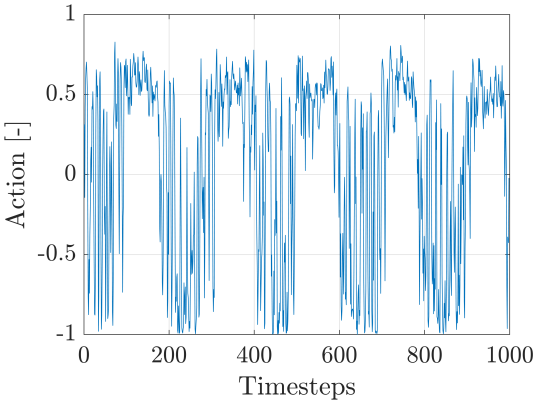
(g) First Episode: Action-State.



(h) Last Episode: Action-State.



(i) First Episode: Actions.



(j) Last Episode: Actions.

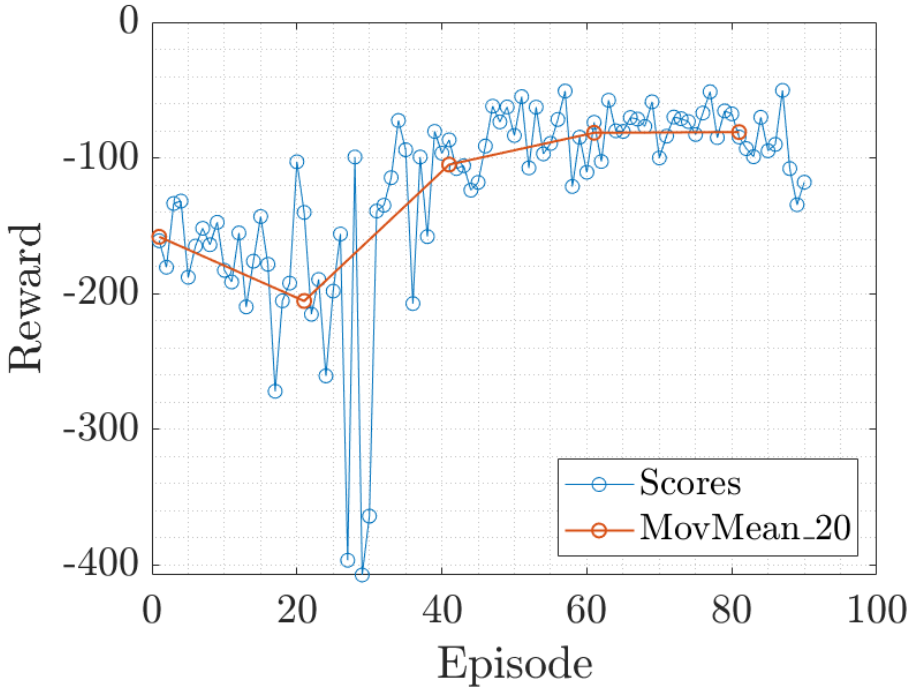
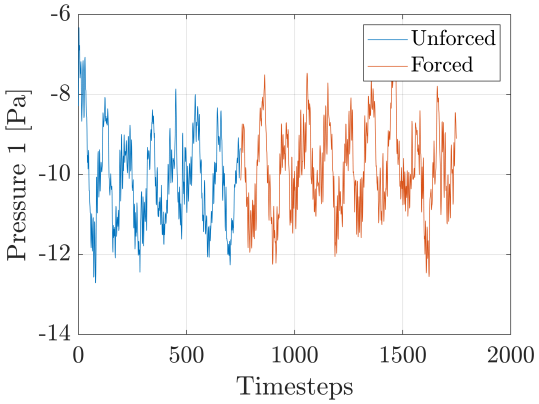
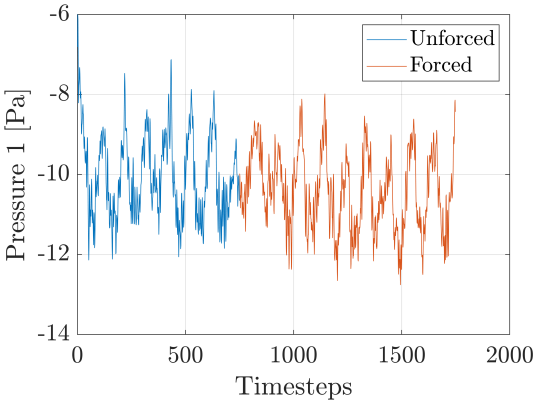


Figure 4.9: Scores

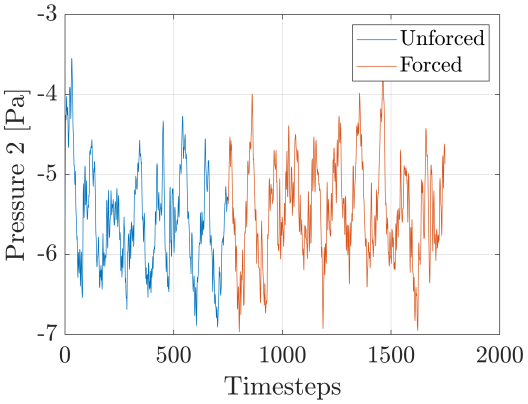
Case 2



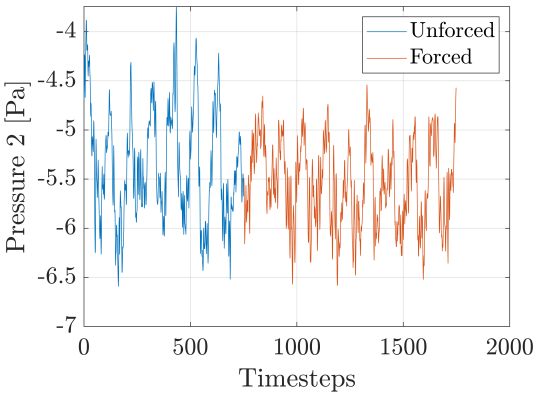
(a) First Episode: Pressure 1.



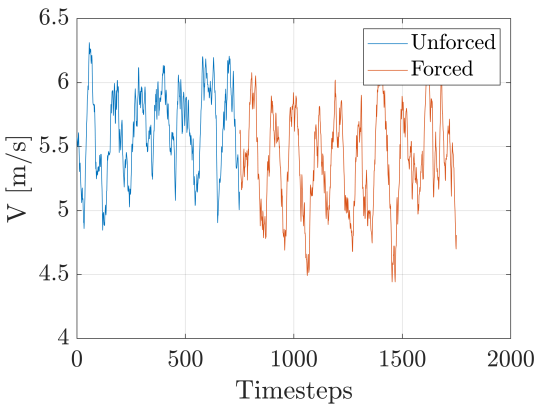
(b) Last Episode: Pressure 1.



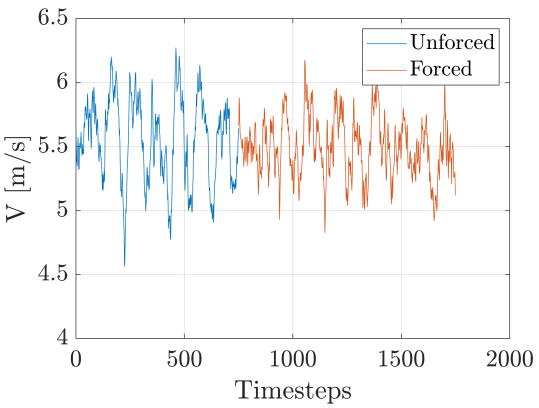
(c) First Episode: Pressure 2.



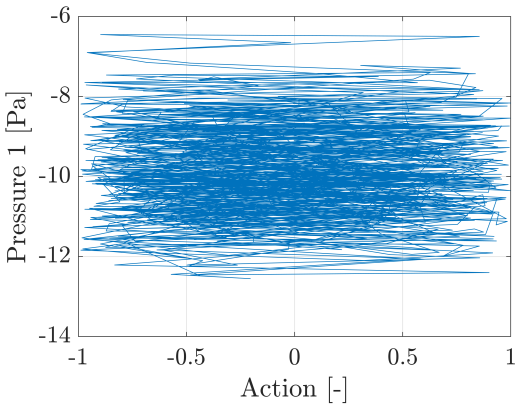
(d) Last Episode: Pressure 2.



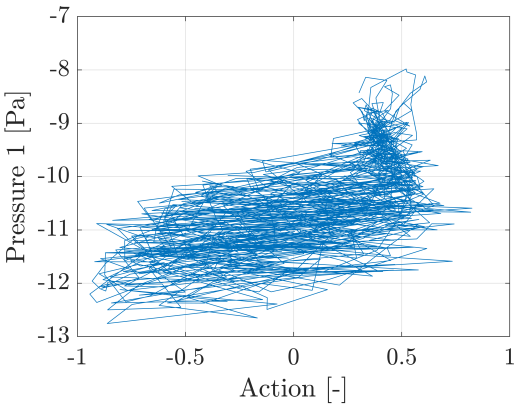
(e) First Episode: Velocity.



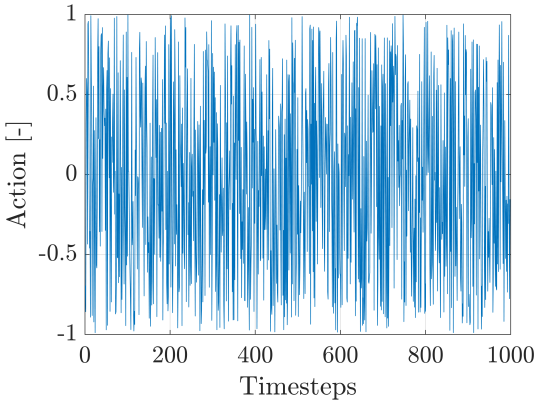
(f) Last Episode: Velocity.



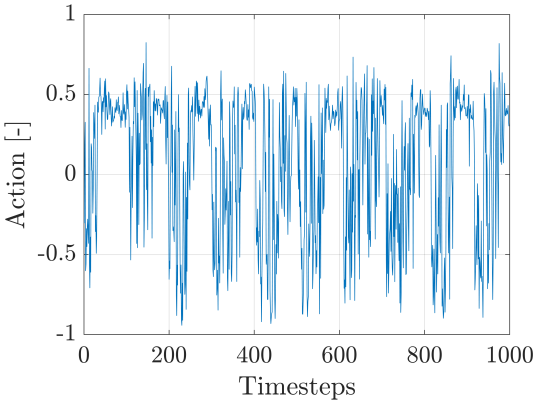
(g) First Episode: Action-State.



(h) Last Episode: Action-State.



(i) First Episode: Actions.



(j) Last Episode: Actions.

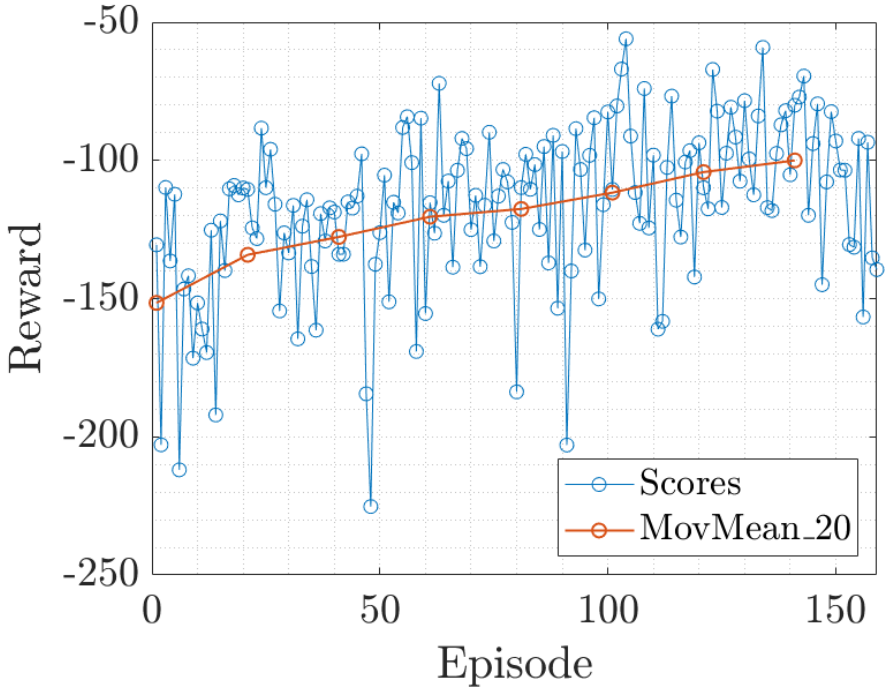
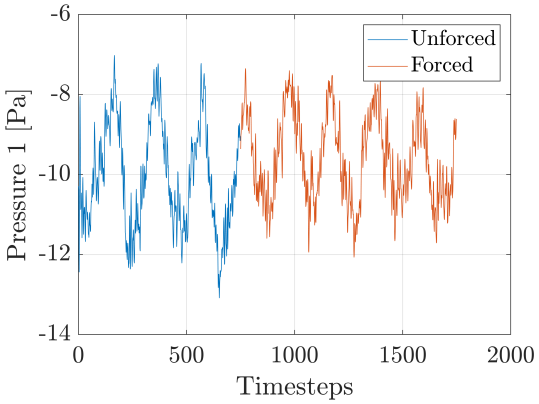
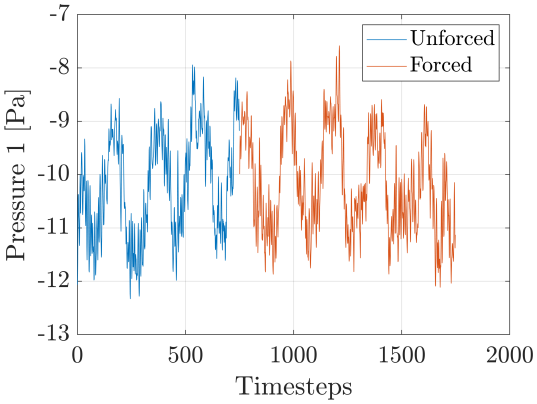


Figure 4.10: Scores

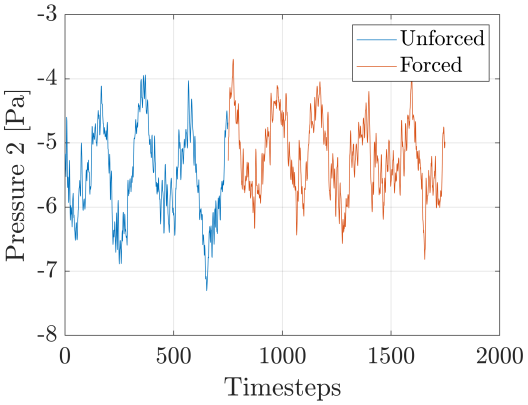
Case 3



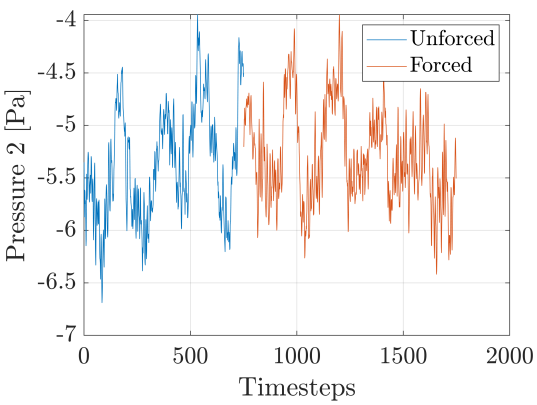
(a) First Episode: Pressure 1.



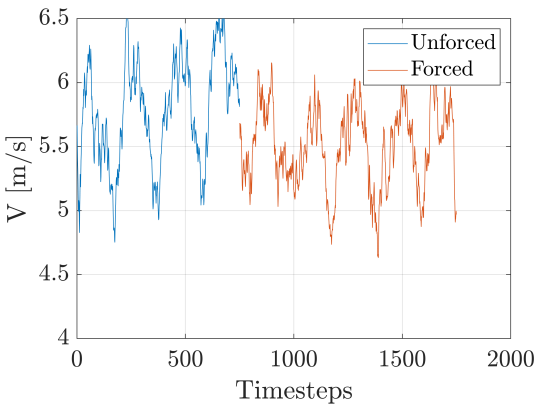
(b) Last Episode: Pressure 1.



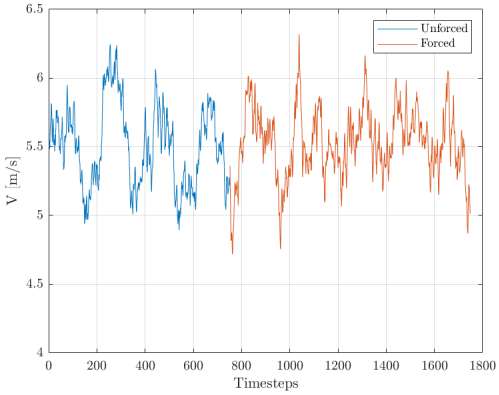
(c) First Episode: Pressure 2.



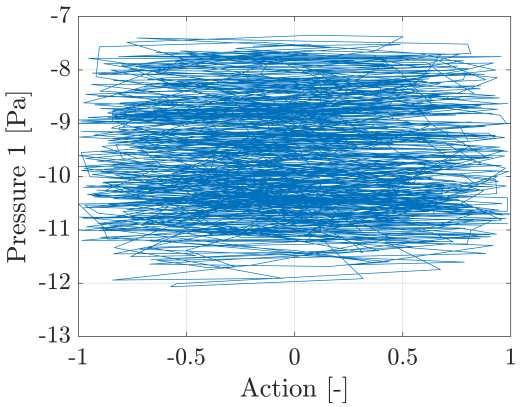
(d) Last Episode: Pressure 2.



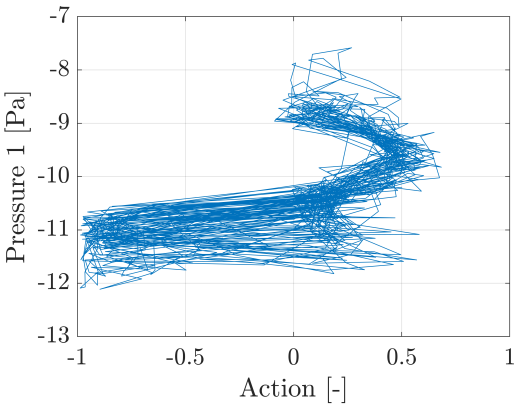
(e) First Episode: Velocity.



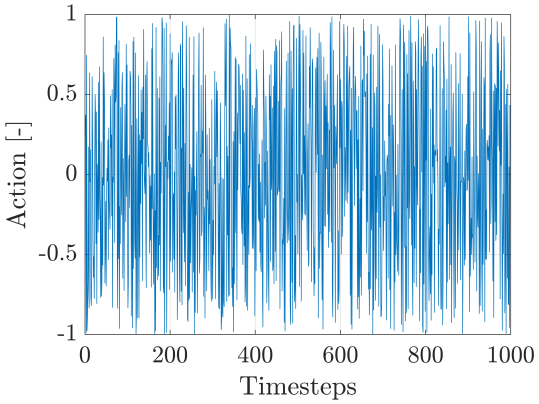
(f) Last Episode: Velocity.



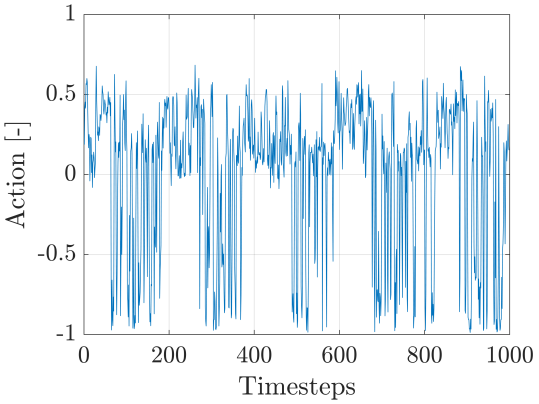
(g) First Episode: Action-State.



(h) Last Episode: Action-State.



(i) First Episode: Actions.



(j) Last Episode: Actions.

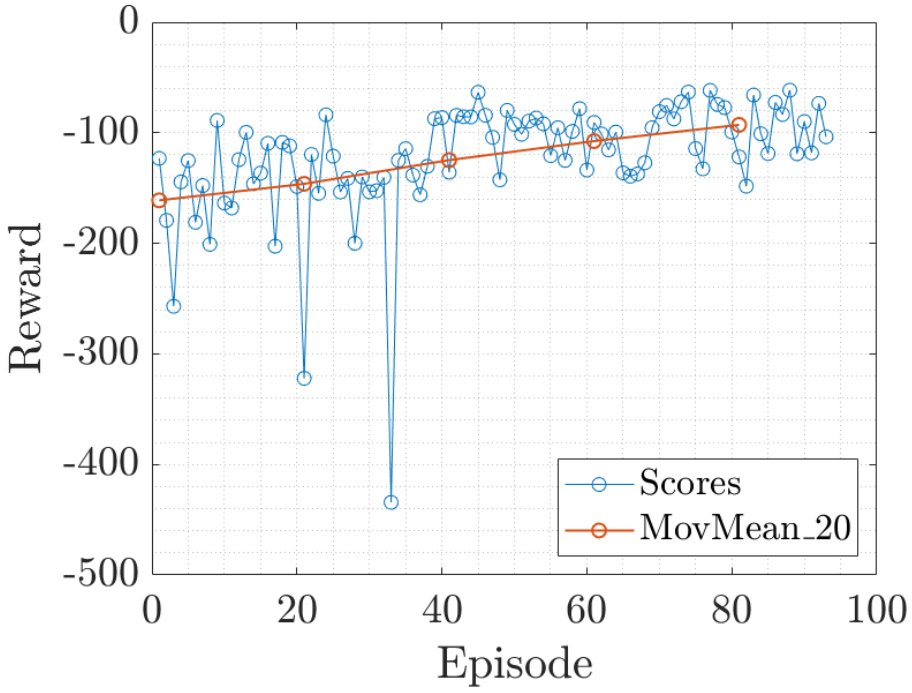
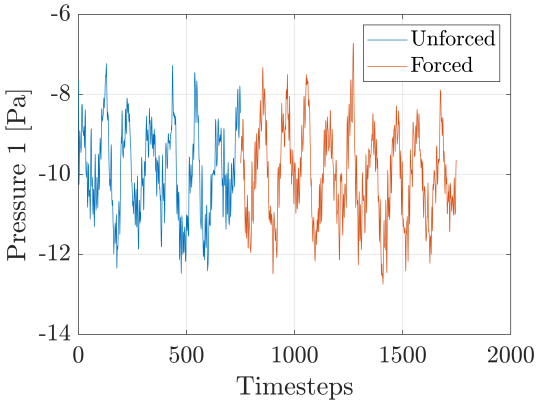
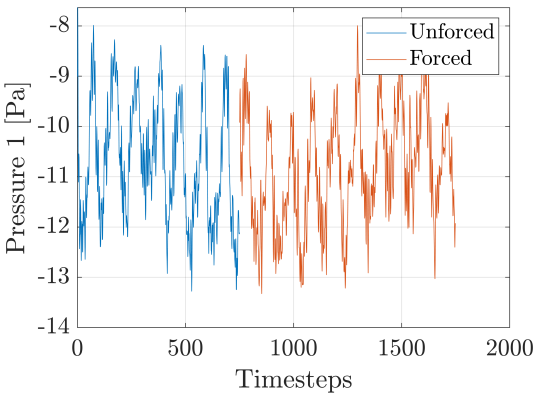


Figure 4.11: Scores

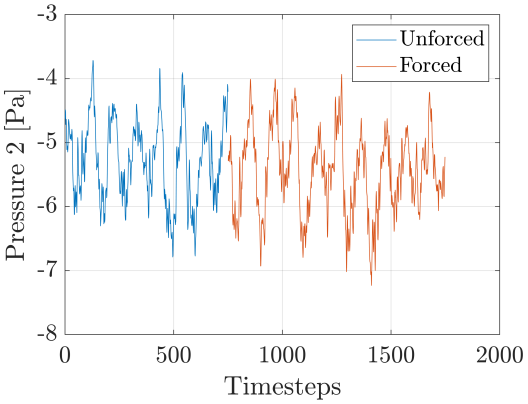
Case 4



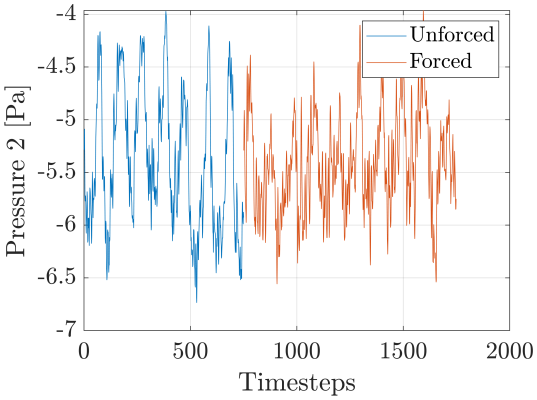
(a) First Episode: Pressure 1.



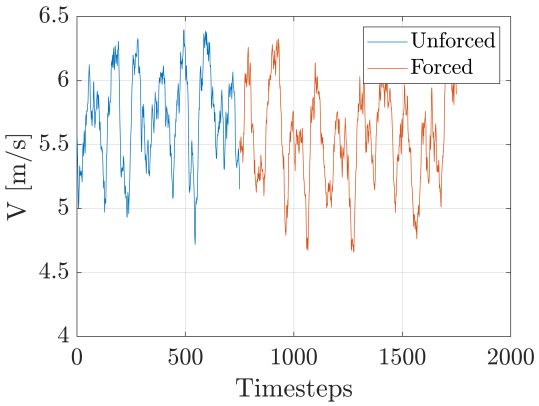
(b) Last Episode: Pressure 1.



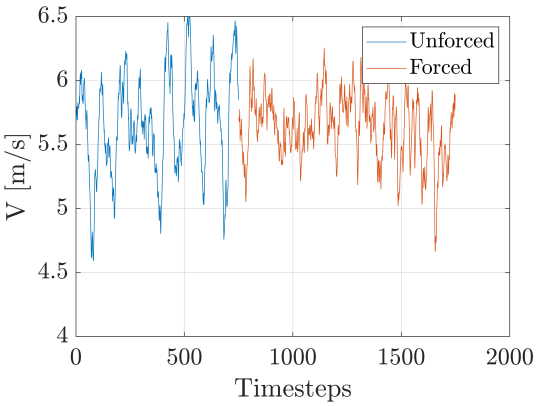
(c) First Episode: Pressure 2.



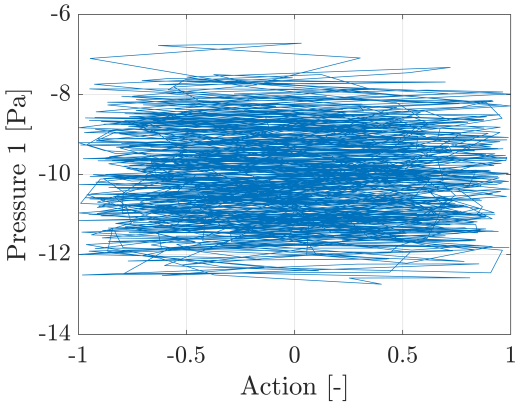
(d) Last Episode: Pressure 2.



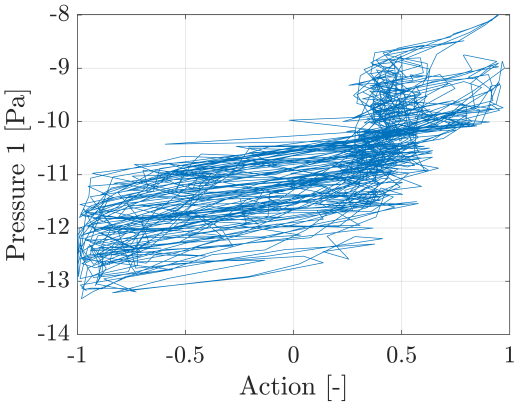
(e) First Episode: Velocity.



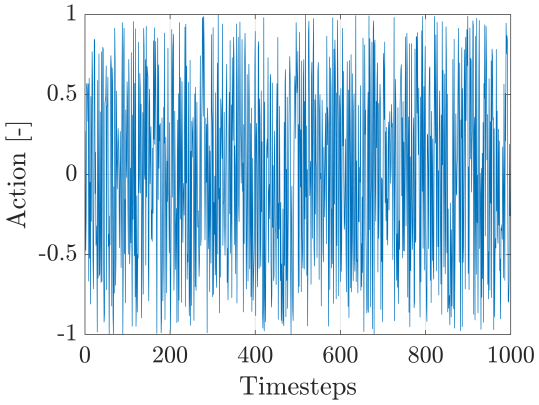
(f) Last Episode: Velocity.



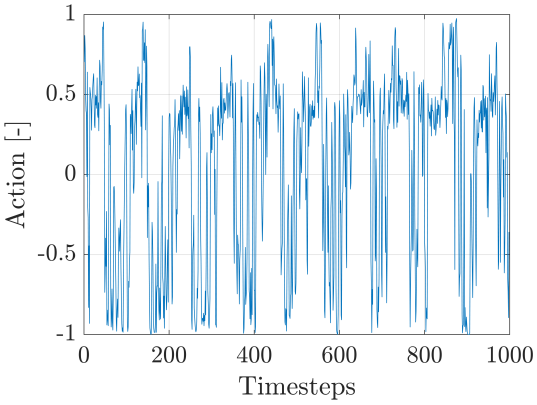
(g) First Episode: Action-State.



(h) Last Episode: Action-State.



(i) First Episode: Actions.



(j) Last Episode: Actions.

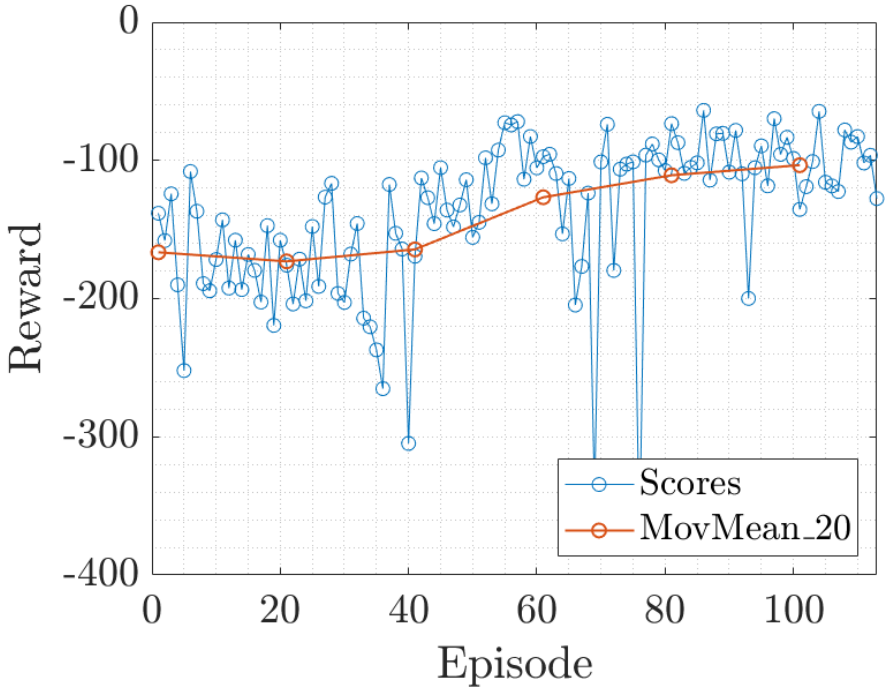


Figure 4.12: Scores

Comparison

Below the comparison between the *scores* changing trends in the four gust cases. It it possible to notice positive trend in each one of these independent tests, verifying the reliability of the algorithm and, for this reason, the reliability of the setup.

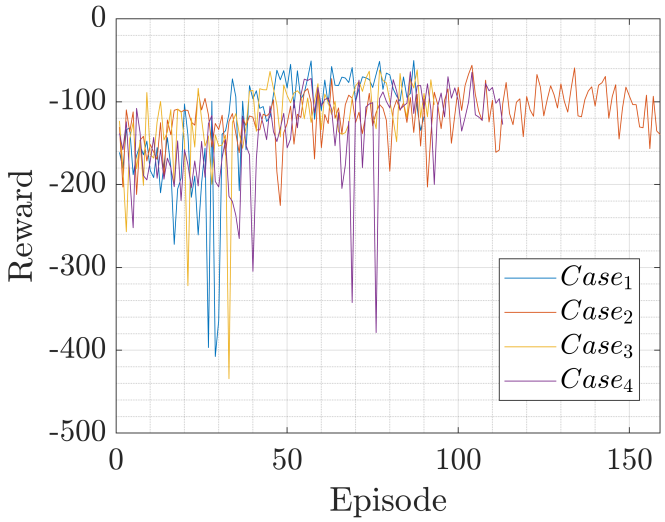


Figure 4.13: Scores comparison

Gust Amplitude Reduction	
Test	*Max Reduction [%]*
1	−11.71
2	−5.15
3	−4.80
4	−4.60

Table 4.4: Gust Amplitude Reduction Percentage

Chapter 5

Conclusions

This project has passed through multiple phases, each playing a crucial role in ensuring the success of the experiment. As previously mentioned, the setup takes inspiration from existing studies on similar configurations setups for codes validation or for *machine learning* algorithms[29] testing. Starting from these articles, the initial *CAD & Design* phase was carried out, starting with a preliminary model. Throughout the development process, the setup design underwent several refinements to optimize its overall performance and adaptability. Several prototypes of the structure were built and tested, leading to a final version that was set as optimal for experimental purposes. The design evolution was driven by the need for precision, robustness, and ease of assembly, ensuring that the setup could be effectively used in future researches and could allow potential modifications.

Moreover, extensive efforts were made in choosing the electronic components to maximize the reliability and efficiency of data acquisition. Initial choices, such as specific microphone capsules and Arduino boards, were abandoned and replaced with more suitable alternatives, leading to a significant improvement in both performance and ease of integration. A key requirement of the setup was to maintain a balance between structural solidity and ease of assembly, allowing for effortless reuse in future experimental scenarios. This flexibility ensures that future modifications can be made with minimal effort while maintaining high levels of precision and reliability.

The *Experimental Setup* detailed in this work has demonstrated remarkable effectiveness in terms of efficiency and robustness, making it a reliable tool for codes and algorithms validation and testing. In particular, the application of Reinforcement Learning (RL) algorithms has led to impressive results in mitigation of periodic gust disturbances through *active flow control*. This capability holds significant potential for practical applications, such as reducing aerodynamic instabilities around structures and improving overall system resilience in turbulent environments. Additionally, the high-frequency response of the data acquisition system presents opportunities for further refinement, potentially leading to the development of even more sophisticated experimental setups.

Beyond its current capabilities, the completed setup has been designed to reach the best flexibility, allowing for future enhancements and extended experimental applications. Specifically, the *Flapping Wing* structure features two $M4$ threaded holes on both sides of the wing, enabling the potential attachment of additional profiles. This adaptability allows for modifications that can extend the wing's transverse dimension, facilitating experiments under varying aerodynamic conditions. Furthermore, the structure incorporates dedicated housings for microphone capsules within the *Profile*. Although not utilized in the present study, these housings offer the potential for future acoustic measurements, providing deeper insights into airflow characteristics and enabling more comprehensive experimental analyses.

In conclusion, the developed and validated *experimental setup* has proved to be a highly effective and reliable tool for laboratory research. Its modular design, combined with robust performance, ensures its applicability for a wide range of future investigations. Whether for testing advanced control algorithms, refining aerodynamic models, or exploring novel applications, this setup stands as a valuable asset for experimental research.

Bibliography

[1] Angelo Lerro. *Meccanica del Volo - Parte 2*. Corso di Meccanica del Volo. Anno Accademico 2024/2025, Politecnico di Torino. 2024 (cit. on p. 4).

[2] Embry-Riddle Aeronautical University. *Maneuvers and Gusts*. 2024. URL: https://eaglepubs.erau.edu/introductiontoaerospaceflightvehicles/chapter/maneuvers-gusts/ (cit. on pp. 4, 5).

[3] Wikipedia. *Controllo PID*. 2025. URL: https://it.wikipedia.org/wiki/Controllo_PID (cit. on pp. 6, 7).

[4] Università degli Studi di Ferrara. *Controllori PID*. 2025. URL: https://www.unife.it/ing/lm.meccanica/insegnamenti/dinamica-controllo-diagnosi-di-sistemi-b/materiale-didattico/Controllori_PID.pdf (cit. on pp. 6, 7).

[5] Hamid R. Berenji. *Fuzzy Logic Controllers*. Tech. rep. Mountain View, CA 94035: Sterling Software, Artificial Intelligence Research Branch, NASA Ames Research Center, 1992 (cit. on pp. 7, 8).

[6] Author Name(s). *Model Predictive Control*. 2014. URL: https://pselab.chem.polimi.it/wp-content/uploads/2014/03/DECDPC/Lez-07-Model-Predictive-Control.pdf (cit. on p. 9).

[7] Oliviero Alessandra. «Data-Driven Control: Teoria e Applicazioni». Anno Accademico 2021-2022, 23 settembre 2022. Tesi di Laurea. Università degli Studi di Padova, Dipartimento di Ingegneria dell'Informazione, Corso di Laurea in Ingegneria dell'Informazione, 2022 (cit. on p. 9).

[8] Zhidong Bai and Arni S.R. Srinivasa Rao. «Handbook of Statistics». In: Elsevier, 2024 (cit. on p. 10).

[9] V. Breschi, A. Chiuso, and S. Formentin. «The role of regularization in data-driven predictive control». In: *arXiv preprint arXiv:2203.10846* (2022) (cit. on p. 10).

[10] IBM. *What is Machine Learning?* 2025. URL: https://www.ibm.com/think/topics/machine-learning (cit. on pp. 10, 11).

[11] MIT Sloan. *Machine Learning Explained*. 2025. URL: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained (cit. on p. 11).

[12] Sandeep Kumar. *Supervised vs Unsupervised vs Reinforcement*. Jan. 2020. URL: https://www.analyticsvidhya.com/blog/2020/01/supervised-vs-unsupervised-vs-reinforcement/ (cit. on p. 11).

[13] Peter I. Renn and Morteza Gharib. «Machine learning for flow-informed aerodynamic control in turbulent wind conditions». In: *Nature Machine Intelligence* 2.6 (2020), pp. 309–318. DOI: 10.1038/s42256-020-0183-4. URL: https://doi.org/10.1038/s42256-020-0183-4 (cit. on p. 12).

[14] ScienceDirect. *Vortex Shedding*. 2025. URL: https://www.sciencedirect.com/topics/engineering/vortex-shedding (cit. on p. 12).

[15] NACA. *NACA 0018 Airfoil Details*. 2025. URL: http://airfoiltools.com/airfoil/details?airfoil=naca0018-il (cit. on pp. 15, 16).

[16] G. Iuso. *Aerodinamica Sperimentale: Definizione del Problema Fisico e Catena di Misura, Tecniche per la Misura della Pressione*. Corso di Laurea in Ingegneria Meccanica e Aerospaziale, Politecnico di Torino. Dipartimento di Ingegneria Meccanica e Aerospaziale, Anno Accademico 2023–2024. Mar. 2024 (cit. on pp. 47, 51).

[17] Honeywell. *TruStability™ Board Mount Pressure Sensors, RSC Series Datasheet*. Datasheet for TruStability™ Board Mount Pressure Sensors RSC Series. 2025. URL: https://www.honeywell.com (cit. on pp. 52, 53).

[18] G. Iuso. *Aerodinamica Sperimentale: ANEMOMETRIA a FILO/FILM CALDO*. Corso di Laurea in Ingegneria Meccanica e Aerospaziale, Politecnico di Torino. Dipartimento di Ingegneria Meccanica e Aerospaziale, Anno Accademico 2023–2024. Mar. 2024 (cit. on p. 56).

[19] Dantec Dynamics. *Miniature Wire Probe (Straight)*. 2025. URL: https://www.dantecdynamics.com/product/miniature-wire-probe-straight/ (cit. on p. 57).

[20] PTK. *PTK 8462 MG-D 7.6Kg 7.4V Metal Gear Digital 9g Size Servo for Fixed Wing Glider Model Aircraft Robot Datasheet* (cit. on pp. 59, 60).

[21] PJRC. *Teensy 4.0*. 2025. URL: https://www.pjrc.com/store/teensy40.html (cit. on p. 62).

[22] Raspberry Pi Foundation. *Raspberry Pi Pico Series Documentation*. 2025. URL: https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html (cit. on pp. 63, 64).

[23] Filip Dominec. *rp2daq: Raspberry Pi Pico for Data Acquisition*. https://github.com/filipdominec/rp2daq. 2025 (cit. on pp. 64, 69).

[24] Windshape. *Windshaper: Generatore di vento multi-ventola per ambienti di laboratorio*. 2025. URL: https://windshape.com/ (cit. on p. 65).

[25] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html (cit. on pp. 72, 73).

[26] «On-Policy vs Off-Policy Learning». In: *Towards Data Science* (2025). URL: https://towardsdatascience.com/on-policy-v-s-off-policy-learning-75089916bc2f/ (cit. on p. 73).

[27] The Farama Foundation. *Gymnasium Documentation*. 2024. URL: https://www.gymlibrary.dev/ (cit. on p. 73).

[28] OpenAI. *Soft Actor-Critic*. 2025. URL: https://spinningup.openai.com/en/latest/algorithms/sac.html#soft-actor-critic (cit. on p. 75).

[29] Cong Feng, Qing Wang, Changhua Hu, and Ligang Gong. «Active Disturbance Rejection Attitude Control for Flapping Wing Micro Aerial Vehicle With Nonaffine-in-Control Characteristics». In: *IEEE Access* 8 (Jan. 2020). Received December 29, 2019, accepted January 14, 2020, date of publication January 21, 2020, date of current version January 31, 2020, pp. 21223–21233. DOI: 10.1109/ACCESS.2020.2968482. URL: https://doi.org/10.1109/ACCESS.2020.2968482 (cit. on p. 86).

List of Figures

Acknowledgements