E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Remote Control of a Fan Array Wind Tunnel

Programa Internacional del Grado en Ingeniería Informática

Trabajo Fin de Grado

Author: Fermin Sola Bienzobas

Director: Daniel Alaez Gomez

Pamplona, January 17, 2025

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Acknowledgments

I would like to express my heartfelt gratitude to Jose Javier Astrain and Jesus Villadangos for their invaluable support and guidance throughout the growth of this project. I am also deeply thankful to the entire team at the Jerónimo de Ayanz building, who provided me with a place in their lab, welcomed me into their group, and offered their assistance along the way.

Most importantly, I would like to extend my deepest appreciation to Daniel Alaez, the cornerstone of this project. As my advisor throughout the development of my final degree project, his contributions have been immeasurable. He not only designed the necessary electronic components but also led the development process and was always available whenever I needed help. Without his expertise and support, this project would not have been possible.

Abstract

Fan array wind tunnels (FAWTs) allow the generation of complex wind flows, with variability in space and time, for many applications. These include aeronautical engineering, civil engineering, architecture, and environmental sciences, among others. These tunnels can produce wind conditions tailored to specific requirements, for example, by recreating oscillating winds, speed gradients, or wind gusts. We propose an open-source Fan Array Wind Tunnel (FAWT) wireless control architecture that is ready to be used or modified, according to the user's needs.

The project required interdisciplinary knowledge in electronics and computer science, with a primary focus on the latter. Skills in networking, programming (Java and Python), and the integration of hardware components such as Raspberry Pi, Adafruit EMC2101, and NanoDAQ-LTS-32 were essential. Additionally, graphical user interface design played a crucial role. The resulting system went beyond the initial expectations, delivering a highly advanced FAWT and an adaptable remote control tool, continuously improved into a valuable resource for research and application.

Keywords: FAWT, open-source, wind tunnel, hardware, software, GUI

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CSV Comma-Separated Values

EtherCAT Ethernet for Control Automation Technology

FAWT Fan Array Wind Tunnel

GPIO General Purpose Input/Output

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

I2C Inter-Integrated Circuit

IDE Integrated Development Environment

MQTT Message Queuing Telemetry Transport

PWM Pulse Width Modulation

SSH Secure Shell

TCP Transmission Control Protocol

UART Universal Asynchronous Receiver-Transmitter

UAVs Unmanned Aerial Vehicles

UDP User Datagram Protocol

1 Introduction

We will never tire of looking up at the sky and seeing airplanes. Huge machines that cross the horizon transporting from cargo to people. However, a new phenomenon is increasingly appearing between the clouds when we set our sights on them: drones. Small, big, fast, slow, piloted, or automatic aircrafts, which are undoubtedly a symbol of technological revolution. These devices open up a world of possibilities in various fields, such as photography, agriculture, delivery, etc.

Drones, like we see in Figure 1, also referred to as Unmanned Aerial Vehicles (UAVs), and their recent democratization has led to a significant increase in scientific production linked to aeronautics.

Figure 1: Photograph of a drone flying (One Air Aviation, 2025)

A very common tool in any aeronautics laboratory is a wind tunnel. Wind tunnels facilitate the understanding of the principles governing flight dynamics, based on the complex Navier-Stokes equations. They allow simplified flow visualization without having to resort to expensive and complex computational fluid dynamics simulations.

Another application that has undergone strong development in recent years is civil engineering. Most of the uses in civil engineering consist of analyzing the effect of wind on structures (Cermak, 2003).

Recently, Fan Array Wind Tunnels, like the one on Figure 2 are gaining popularity over conventional wind tunnels. The principle of operation of these tunnels is based on replacing a single large, powerful fan with multiple, individually controllable fans. However, the introduction of a large number of control variables adds a great deal of complexity to the system, both from an electronic and a software point of view. Consequently, the parameter space is greatly increased.

Figure 2: Drone in front of a FAWT (Dantec Dynamics, 2024)

A key aspect in the design of these systems is scalability. By designing a modular framework, hardware and electronics can be scaled according to experimental needs, which is a common limitation of conventional wind tunnels. Also, the fan diameter and maximum power can be adjusted accordingly.

1.1 Objectives and Justification

As discussed in the previous section, FAWTs (Fan Array Wind Tunnels) are gaining popularity and becoming a powerful tool for drone testing, paving the way for future advancements. In this final degree project, we propose an open-source FAWT wireless control architecture that is ready to use or customize according to user needs: **JavaFAWT** (Java-based Fan Array Wind Tunnel).

This architecture is built upon readily available hardware, such as PC fans

and Raspberry Pi electronics, and operates on any machine capable of running Java and equipped with a wireless network adapter. The hardware, software, and electronics will be fully open source and thoroughly documented, allowing researchers, engineers, and users to scale, modify, and adapt them to suit their specific requirements. Besides the fact that the materials used are low-cost.

Our goal is to make this tool publicly available, designed with precision, while also providing the necessary knowledge for anyone to replicate it in their own laboratory. This initiative aims to introduce a wind tunnel to the market that meets the mentioned characteristics—something that does not yet exist.

To achieve this, we will work on two interconnected domains that are essential to most significant projects: **electronics** and **computer science**. Collaboration will play a key role in this process, with frequent communication with Daniel Aláez, who will primarily handle the electronics aspect.

Although this is a computer science-focused project, the role of electronics cannot be overlooked due to its fundamental importance. While we will not explore deeply the electronic details, they will be addressed and explained. Naturally, the main focus will remain on the computing aspect. Given its scope, this area will be divided into several key tasks:

1. **Server Implementation:** The first step is to develop a server that manages all communications. This server will act as the core application, enabling remote control of the fan array.

2. **Graphical User Interface:** The application will be developed in Java and will feature an intuitive graphical user interface, enabling user-friendly interaction.

3. **Protocol Selection:** For communication, we need to decide between the

two primary protocols: TCP or UDP.

4. **Client Development:** Once the server is operational, clients must be prepared to receive information from the application. These clients will also be equipped with the necessary tools to process this information and send it to the fans, enabling precise speed control, the ultimate goal of the application.

5. **Feature Enhancement:** Beyond the basics, the application will be enhanced with features to make it indispensable for FAWT control. Some of these features will include:
   - Real-time visualization of fan speeds.
   - Functional file processing capabilities.
   - Installation and display of pressure values at various points within the FAWT.
   - Configuration options for the fan array modules.

By the end of this project, we aim to deliver a robust, user-friendly, and fully documented tool that sets a new standard for FAWT control systems.

1.2 Technological Context

Historically, wind tunnels have proven a useful tool for aircraft design (Neal et al., 2004; Poisson-Quinton, 1968), optimization (Coutu et al., 2011) and testing (Klein & Murphy, 1998; Nicolosi et al., 2014). With the recent popularization of UAVs, studies have also been published on the performance of complete UAVs (Russell et al., 2016) or their components in wind tunnels (Brandt & Selig, 2011; Theys et al., 2014). The design and characteristics of traditional wind tunnels have also been analyzed by numerous authors, with a particular focus in low-speed wind tunnels (Barlow et al., 1999; Cattafesta et al., 2010; Mehta & Bradshaw, 1979).

The above-mentioned studies using conventional wind tunnels typically focus on static ideal conditions, far from real-world environments. Unlike large aircrafts where wind gust magnitude is relatively small compared to the mean flow, wind gusts can significantly affect small aerial vehicles during flight (Johnson & Jacob, 2009). As a solution, fan array wind tunnels have been proposed over the last few years (Ozono et al., 2006). This type of wind tunnel basically consists of an array of a large number of fans that may be arranged on various patterns on demand (Noca et al., 2021). One of the most significant applications of this technology is for testing the Ingenuity Mars Helicopter (Veismann et al., 2021) inside NASA's Jet Propulsion Laboratory (JPL) low pressure environmental chamber. According to (Dougherty et al., 2020), a small, near-silent FAWT has allegedly been used to aid in the neurobiological studies of mouse olfaction. Another relevant use case is the performance analysis of rotorcraft propulsion units in a combination of wind and icing conditions (Catry et al., 2021). The aforementioned studies highlight the wide range of applications of this technology.

Despite the large number of use cases, this technology is still experimental or has a high acquisition cost. Some of the few commercial solutions are closed to the purchase of a single fan configuration (fixed number of rows and columns) and are subject to the use of proprietary software.

1.3 Contributions

To try to overcome the disadvantages highlighted in the previous section, we propose an open-source FAWT wireless control architecture ready to use or modify according to the user's needs, namely JavaFAWT (Java-based Fan Array Wind Tunnel). This architecture is based on common hardware such as PC fans and Raspberry Pi electronics, and works on any machine capable of running Java and a network wireless adapter. The hardware, software and electronics are provided fully open source, and are well documented so that any researcher, engineer or

user can scale. Certainly, everything is up to modifications and adaptations to the user's needs. Providing the sector with a new and highly useful tool that can be utilized to its fullest potential.

2 Used Methodology

This project was not completed in a single day; it is the result of a long journey of exploration, development, trial, and error, culminating in innovative solutions. Some parts of the project could be done at home just by using a computer. However, there were others that required being in the lab. The lab was located in the *Jerónimo de Ayanz* building on the Arrosadia Campus of UPNA. Although the lab was not exclusively for our use, in the beginning, we had enough space to work. As the project began to take shape, we moved to another lab where we had plenty of room to store all the equipment and conduct the necessary tests.

I not only accessed the lab to configure parts of the fan wall or test the program with the actual fans, but also to meet with Daniel Alaez, my project supervisor.

2.1 Meetings

He has been the one guiding me throughout the project's development. Because of this, I regularly met with him in the lab to review the progress, identify areas for improvement, and discuss upcoming tasks. I kept track of all this information in a document to refer to whenever needed.

I consider these meetings to have been crucial to the successful completion of the project, especially because, in many cases, we were working in parallel, and frequent communication was essential for that.

2.2 GitHub

For the development of the app, a widely recognized version control system called GitHub was utilized. In addition to hosting different versions of the FAWT app,

the repository includes essential resources such as documents, tutorials, videos, images, examples, and detailed information about the hardware and software. The project is publicly available under the name JavaFAWT and can be accessed on GitHub.

Choosing to make the project public on GitHub allows anyone interested in it to explore, learn, and even contribute. This decision aligns with our commitment to collaboration and knowledge sharing. Using it has been highly beneficial for several reasons:

- **Version Control**: It enables seamless tracking of changes, ensuring we can revert to previous states if necessary.
- **Collaboration**: GitHub facilitates teamwork by allowing multiple developers to work on the same project simultaneously without conflicts.
- **Accessibility**: Being a public repository, it provides an excellent platform for sharing knowledge and encouraging external contributions.
- **Documentation**: GitHub acts as a centralized hub for all related resources, such as tutorials and hardware specifications, making everything easily accessible.
- **Reliability**: Hosting the project on GitHub ensures secure and reliable storage of the project and its history.

2.3 Gantt diagram

As mentioned earlier, the development of the project has been incremental, with many stages and tasks along the way. To illustrate this, a Gantt chart shown in Figure 3 has been created. It displays the various activities carried out throughout the development history.

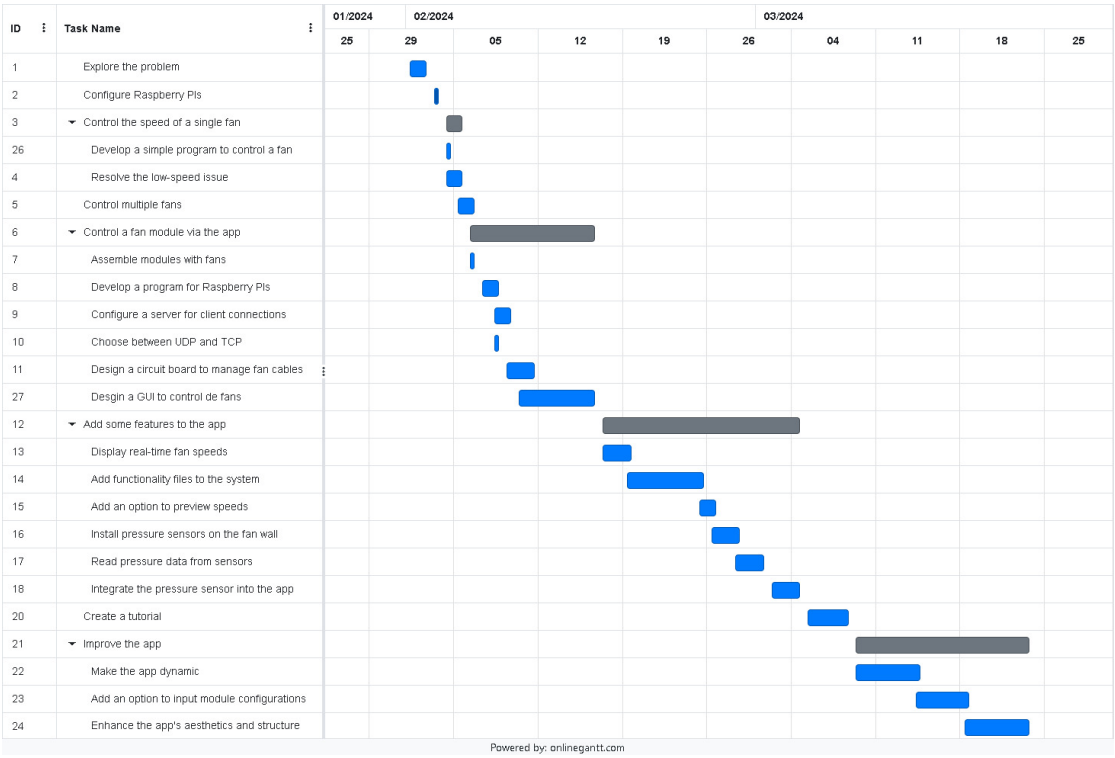

Figure 3: Gantt diagram of tasks

While the dates shown may not be entirely accurate due to holidays and days off from work, including these breaks would not contribute to understanding the diagram's essence. Therefore, the dates have been adjusted for clarity, focusing on the sequence and duration of the tasks.

The most important aspect of the diagram is the order and length of the activities. Some tasks, represented in grey, consist of subtasks and are not considered complete until all these subtasks are finished. On the other hand, the blue tasks are more independent. The chart highlights that some activities span a longer period than others, which is particularly significant. It also demonstrates how the project has been developed gradually—without the initial tasks, the later ones would not have been possible.

An interesting point contained in the Gantt chart is the *creation of a tutorial*. As we will see later this work, the use of the developed tools may feel overwhelming for someone without prior knowledge on the project. Therefore, at an advanced

stage of the project, it was decided to draft a tutorial to assist users in navigating the FAWT and its corresponding application, as well as all the preliminary configurations. This tutorial is also available on GitHub and proves to be highly valuable, especially in the beginning.

3 Hardware and Software

As shown in the Gantt chart (3), the first step to begin the project is to conduct research on what we aim to achieve, the resources available, and how we will use them to accomplish our goals.

As mentioned earlier, the project is a very ambitious idea due to the need to combine electronics with computer science. This presents many challenges and requires strict synchronization between both domains.

3.1 Technical Approach

When designing the FAWT, there are different strategies that can be considered, related to its structure and the control to be implemented over the fans. Figure 4 shows different FAWT designs that have been made.

Figure 4: Examples of FAWT designs (Dantec Dynamics, 2024)

3.1.1 FAWT Organization

Controlling each fan individually is costly and inefficient, whereas controlling a large wall of fans simultaneously can be overwhelming. Therefore, we decided to create fan modules. The FAWT will be composed of these modules, with each module containing six fans. This modular design allows for a scalable and

organized approach to constructing the wind tunnel.

It is important to remark that the configuration of the module is static, cannot be changed, and is the same for all of them. It contains the six fans in a 3×2 configuration, that means, three rows and two columns. In Figure 5, we can see 4 already built modules. In the image, the fan modules do not appear in the correct orientation because the photo was taken with the modules in a horizontal position, for a better view.

Figure 5: Photograph of four modules in the laboratory

Module Structural Unit:

The structural unit of each module has been designed to be manufactured in two distinct ways, depending on the available materials, tools, or budget. Figure 6 provides a comparison of the two proposed fabrication techniques.

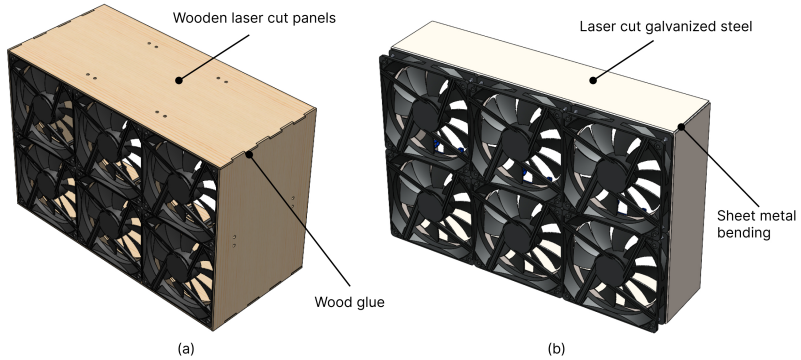

Figure 6: Module structural unit 3D assembly: (a) Wooden panel structure (b) Galvanized steel plate structure. Figure created by Daniel Alaez.

- **Option a** involves a wooden structure assembled from individual flat wood panels. This is the most cost-effective solution and can be cut into shape using a wood laser cutter or even a hand saw. The panels can then be assembled using puzzle-shaped joints and glued together. For added stability, a 3D-printed angle bracket is included for screwed reinforcement.

- **Option b**, while more expensive, offers a more professional and durable solution. It consists of a flat galvanized steel plate, which can be laser cut and formed using sheet metal bending. This method allows the entire structure to be fabricated as a single piece, eliminating the need for joints. The modules can then be assembled using conventional T-slot structural framing.

In both alternatives, the fans can be mounted to the front of the structure using standard PC screws supplied by the fan manufacturers. Similarly, the control board and the interface board can be secured to the back of the module, keeping wiring and connections concealed. The plate where the fans are mounted includes holes for routing cables to the back of the module, and 3D-printed parts are available to organize the cables neatly.

To ensure that the airflow generated by the fans is as straight as possible, a honeycomb flow straightener was added. This mesh is attached to the air outlet of the fans without interfering with their movement. It features small holes designed

to straighten the wind as it passes through the mesh, allowing us to create more controlled airflow patterns.

3.1.2 Communication Strategy

The objective is to design a structure composed of PC fans connected to a computing component via cables. Fan control will be wireless, implemented through a client-server architecture.

Wireless connectivity offers significant advantages over traditional wired methods such as Ethernet for Control Automation Technology (EtherCAT), Universal Asynchronous Receiver-Transmitter (UART), or Inter-Integrated Circuit (I2C). Unlike wired approaches, it eliminates dependency on the number or availability of physical ports and reduces the complexity and cost associated with cables. For instance, I2C requires multiplexers to overcome its connection limits, adding complexity and expense. In contrast, Wi-Fi enables seamless, scalable communication without these constraints. Moreover, the widespread adoption of Wi-Fi means that most low-cost development boards include built-in modules, further simplifying and reducing the cost of system implementation.

In this architecture, the client device will be directly connected to the fans, while the server will run an application with a user-friendly interface, allowing remote control of the fans. Client-server approach was chosen due to several reasons:

1. **Separation of Responsibilities:** In a client-server architecture, the system can be divided into two clearly defined components:
   (a) **The Server:** Responsible for managing the most complex tasks, such as the user interface, logic control, decision-making, monitoring of the fans, etc. The server will therefore handle the information sent to the Raspberry Pi.

(b) **The Client:** Responsible for receiving commands from the server and executing specific actions, such as controlling the fan speeds. This separation allows the client to focus on lightweight and specific tasks, while the server handles more arduous operations.

2. **Simplicity:** This architecture relies on well-defined protocols, typically using standards like Hypertext Transfer Protocol (HTTP), Message Queuing Telemetry Transport (MQTT), or WebSocket. This simplifies communication between the laptop and the Raspberry Pi, as well as implementation on both components.

   Additionally, if logic control needs updating, parameters need adjusting, or new features need to be added, these changes can be made on the server without modifying each Raspberry Pi individually. This facilitates long-term maintenance and updates.

3. **Scalability:** A single server can manage multiple clients. If additional devices or functionalities are needed in the future, the client-server architecture allows for easy expansion without significant changes. This is especially beneficial in our case, given the uncertainty of how many clients will eventually connect.

4. **Fault Tolerance:** If one of the client devices (e.g., a Raspberry Pi) disconnects or fails, the server can handle such situations and continue operating with the remaining devices. Moreover, the server can continuously monitor clients and detect potential issues.

5. **Flexibility:** Remote access is possible from various devices. The fan control system can be accessed from any device functioning as the server (e.g., the laptop) or even from other devices connected to the same network (or the Internet). This facilitates remote management and monitoring from different locations. Additionally, the server provides a richer and more complete user interface on the laptop, enabling the operator to intuitively adjust fan speeds.

With the communication architecture defined, we decided to use the TCP protocol over the two primary options available: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

1. **Connection:** TCP requires establishing a connection between the client and the server, unlike UDP. This enables verification by both parties that they are ready to communicate. While UDP skips this step and is faster, the speed difference has no big impact on the fans' performance. Thus, TCP is more beneficial in this context.

2. **Reliability and Error Control:** TCP ensures that data packets are delivered to the destination, retransmitting any lost packets if necessary. We want all bits to reach the fans without exception. Additionally, TCP includes error prevention mechanisms to ensure that the fans do not spin at unintended speeds.

3. **Overhead:** TCP introduces more overhead due to its control and management mechanisms, whereas UDP minimizes this risk by lacking such features. However, the amount of network traffic in our project is minimal, so this should not be a concern.

4. **Order:** Unlike UDP, which sends packets without ensuring their order of arrival, TCP guarantees that packets arrive in the correct order. Packet order is critical in our project, especially when creating specific wind flows where each fan must operate at a precise speed at a specific time. For this reason, TCP is preferable to UDP in this aspect.

Next, we need to select a computing component responsible for directly controlling the fans. However, before proceeding, we will examine the fans and their functionality.

3.2 Controlling a Single Fan

The fan model we are using is the b-Blaster 140x38 mm 12V (PWM version) (bgears, 2024) shown in Figure 7. Some characteristics of this model that led us to its election are the following:

- **Maximum air flow**: 308 cubic feet per minute.
- **Maximum speed**: 5,200 revolutions per minute.
- **Maximum power**: 24 watts.
- **Connection**: The b-Blaster 140×38 12V model includes a 4-pin connector for speed control via Pulse Width Modulation (PWM). The different colored wires for each pin are as detailed below:
  - **Blue:** Receives the PWM signal for the control.
  - **Yellow:** Sends a PWM signal to the system to monitor the fan's speed.
  - **Red:** Provides a 12V power supply.
  - **Black:** Ground.

Figure 7: b-Blaster 140 mm fan (bgears, 2024)

For our purposes, we are primarily interested in the red and black wires for power supply, and the blue wire to send the PWM signal, which allows us to control the fan's speed as desired.

After conducting some initial tests by supplying power and connecting the blue

wire to a signal generator, we managed to send a manual signal at 100% duty cycle. This resulted in a current of 1.8 amperes, which represents the maximum current the fan consumes when running at full speed. Consequently, this became our target value for controlling the fan at 100% speed. The next step was to choose the hardware component capable of achieving it.

Our first approach was to use the EMC2101 (System, 2020). For that, we utilized the Adafruit EMC2101 library (CircuitPython, 2024), which provides several functions to control the fans. However, when we sent a 100% signal to the fan through the EMC2101 board, it only consumed 1 ampere, well below the expected value. Similarly, we tested using an Arduino, but it only reached 1.5 amperes, which was also insufficient.

We discovered that the issue stemmed from the voltage and frequency of the PWM signal. The required frequency was 5 kHz, and the control signal voltage needed to be 6V. We were able to reach the 1.8 ampere consumption by using a voltage booster, which we incorporated between the PWM signal source and the fan. For this, we ultimately decided to use a Raspberry Pi to send this signal. Therefore, this would become the computing component acting as the client. Several factors influenced our decision to select the Raspberry Pi over other similar devices:

1. **Cost:** The Raspberry Pi is relatively inexpensive compared to other hardware components and offers excellent value for its price. This is especially important when deploying multiple client devices to control various fans or working with a limited budget.

2. **Size:** The Raspberry Pi is compact, lightweight, and easy to install. Its small size allows for seamless integration into constrained spaces, such as a wall of fans where space is limited.

3. **GPIO Pins:** One of the primary reasons for choosing this component is its

General Purpose Input/Output (GPIO) pins. As we will discuss later, the Raspberry Pi board includes several GPIO pins, which facilitate interaction with the fans by sending and receiving speed signals.

4. **Built-in Wi-Fi and Bluetooth:** The more recent versions of the Raspberry Pi, come with built-in Wi-Fi and Bluetooth capabilities. This allows the Raspberry Pi to connect wirelessly to the server (e.g., a laptop) without requiring additional accessories, simplifying network configuration and reducing extra costs.

3.3 The Client Controlling a Module of Fans

Among the wide variety of Raspberry Pi models, we decided to use the Raspberry Pi Zero 2W. This specific model is much smaller compared to others, giving us greater flexibility to position it on the FAWT without occupying too much space. Another advantage of Raspberry Pis is that the Raspberry Pi 2, 3, 4 and Zero share the same GPIO pin layout (Foundation, 2024). This means that if the exact model of Raspberry Pi is unavailable, another model can be used without altering the pin configuration or software. The only difference lies in how the Raspberry Pi is utilized and the space it occupies.

Regardless of the Raspberry Pi model chosen, it must be configured to be used as a client. Configuration involves installing an operating system and ensuring it connects automatically to the Wi-Fi network. We opted for Linux since the sole function of the client is to control the fans by sending commands, and Linux is simple enough for this purpose. Before establishing any connection with the server, we decided to test the control of a single fan with a preconfigured Raspberry Pi.

Once the Raspberry Pi is configured, it is advisable to assign it a unique hostname. This ensures that when the device powers on and connects to a known Wi-Fi

network, it can be easily identified using a software like *Advanced IP Scanner*. With its IP address, you can establish a Secure Shell (SSH) connection, allowing you to execute commands directly on its terminal. To do this, we developed a simple Python program. The idea of using Python came for the following motives:

- Python is included by default in Linux operating systems, so no additional installation is required.
- It is a simple and readable programming language with a broad range of tools.
- It supports client-server architecture with easy implementation.
- It provides access to numerous libraries, including *RPi.GPIO*, which we will use to control the fans.

The Python library *RPi.GPIO* (Contributors, 2024) offers a variety of functions to fully control the GPIO pins on the board, and consequently, any PC fans connected to those pins. However, this library works only when executed on a hardware with GPIO pins, such as a Raspberry Pi. Running it on a laptop or another device without these pins, would result in an error.

By using some methods of the library: *setMode()*, *ChangeDutyCycle()*, *setup()*, and *start()*, we can send PWM signals to the fans. Thus, we developed a Python program called *pwm_client.py*, which imports the *RPi.GPIO* library and uses these functions. This program acts as the client and runs on the Raspberry Pi. Additionally, it establishes a TCP connection with the server, connecting to the IP address and port specified in the parameters.

Once connected to the server, the Raspberry Pi's terminal displays the client number assigned to it. This number is provided by the server and is based on the order in which clients have established the connection. It serves as a unique

identifier for each client. Thanks to it, during execution, when a client receives the packets sent by the server, he can determine which section of the packet corresponds to him.

3.3.1 Connection between Raspberry Pi and Fan Module

In *pwm_client.py*, the GPIO pin numbering is configured to match the physical pin layout of the Raspberry Pi board. This board has various pins for different purposes. Figure 8 shows the ones of the Raspberry Pi Zero 2W. While there are 40 pins in total, we are only interested in those circled in red: 37, 35, 33, 31, 29, and 23. These, are configured as PWM outputs with a frequency of 5 kHz and initialized at zero speed. Six pins are used, corresponding to the six fans per module. Thus, one Raspberry Pi controls all the fans of a module.

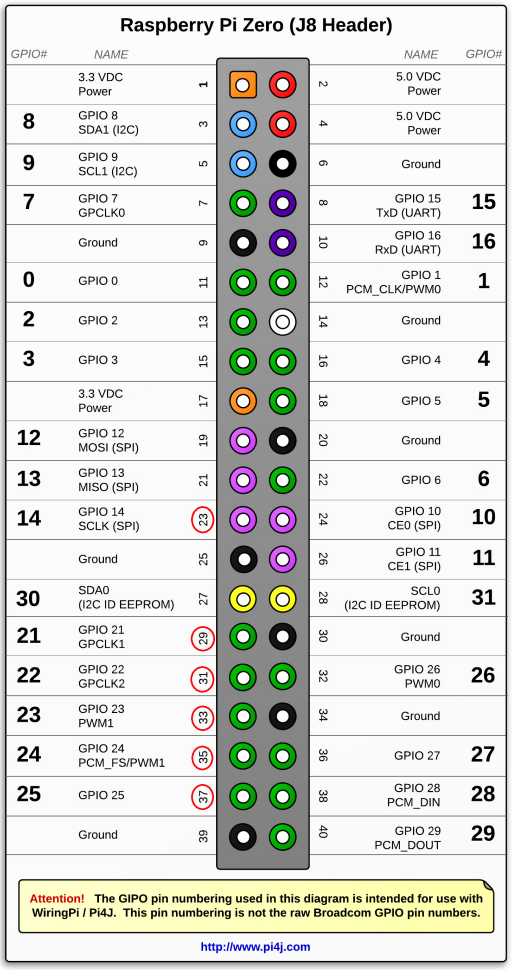

Figure 8: Pin layout of the Raspberry Pi Zero 2W (Pi4J, 2012)

Each GPIO pin corresponds to a specific fan in the module, based on its position. The pin-to-fan mapping is shown in the Table 1 and Figure 9:

Pin Number	37	35	33	31	29	23
Fan Number	1	2	3	4	5	6

Table 1: Pin-to-Fan Mapping

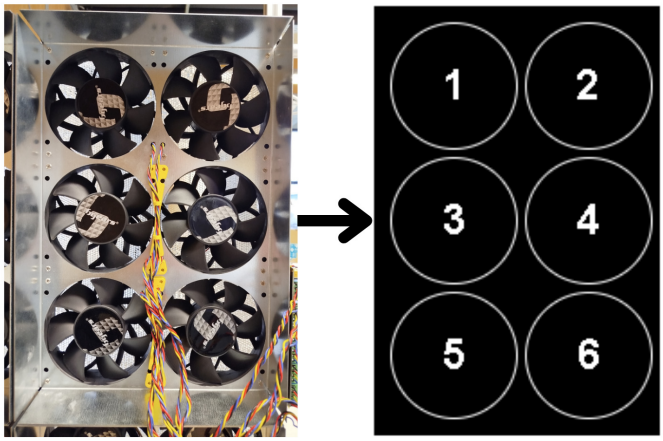

Figure 9: One module of fans and its numbering. Own elaboration

At the left side we see a photo of the real fans, whereas at the right it is shown the representation of this pin assignation. This assignation has been chosen by us, because we considered it was the best possible to favor the whole organization of the FAWT.

3.3.2 Control Mechanism

During execution, the client receives packets containing the new speeds for the fans. Each packet, a string of integers separated by commas, is parsed to extract the values. These values are then passed to the GPIO pins, allowing the fans to adjust their speed accordingly. Each packet contains the speed values for all the clients, but the client uses its assigned number to determine which values correspond to him in particular.

Initially, we considered sending separate packets to each client. This approach would reduce packet size, potentially make communication faster, and eliminate the need for client numbers, since each packet would inherently belong to its recipient. However, the packet sizes are relatively small. The packets are transmitted using the standard TCP protocol without additional IPv4 options. This means each packet includes a TCP header (20 bytes), an IP header (20 bytes), and a payload. To determine the total size, we sum these three components. The

packet structure is illustrated in Figure 10.

Figure 10: Structure of the packet

With only one client, the payload can range between 11 bytes (e.g., "1,2,3,4,5,6") and 23 bytes (e.g., "100,100,100,100,100,100"), as each character occupies one byte. For simplicity, we assume the maximum payload size of 23 bytes. With two clients, the size increases to 47 bytes (e.g., "100,100,100,100,100,100,100,100,100,100,100,100"), and so on for additional clients.

In a 100 Mbps network (a common standard) and considering TCP/IP headers, the transmission time for a packet can be calculated using the formula:

$$\text{Transmission time} = \frac{\text{Packet size (bits)}}{\text{Network speed (bits/second)}}$$

Table 2 summarizes the packet sizes and corresponding transmission times for varying numbers of clients, based on the above formula.

Number of Clients	Packet Size (bytes)	Transmission Time (µs)
1	63	5.04
2	87	6.96
3	111	8.88
4	135	10.8
5	159	12.72
6	183	14.64

Table 2: Packet sizes and transmission times for different numbers of clients

From Table 2, we observe a slight increase in transmission time as the number of clients grows. Yet, these differences are negligible in practice. The time required for a fan to adjust its speed upon receiving a PWM signal far exceeds the packet's transmission time, rendering the latter irrelevant. Using a single packet per operation also simplifies the server's task, as discussed later.

Moreover, each fan is uniquely identified by its pin, allowing the server to associate specific values with each fan in the packet. The Raspberry Pi terminal also displays the speed values read by the client, providing real-time feedback to the user.

3.4 More about the hardware

Before diving into the application developed for the remote control of the FAWT, let's finish explaining the hardware that makes up the project.

The FAWT can be structured with the desired number of modules. Therefore, there are many ways to distribute this groups of fans. It depends on the number of modules you have, the available space, the type of wind flows you want to create, etc. In our case, we created only 4 groups. Enough to perform the necessary tests and fit in the laboratory. We then distributed them in a 2×2 structure. We could also have done, for example, a 4×1 or a 1×4 arrangement, but it doesn't make much sense when the goal is to create a wall in front of which a drone should stand. Fortunately, the drone we wanted to test was small enough to fit in our FAWT dimensions, so we decided to keep it that way. But imagine you have a much larger aircraft (or flying machine); you would need a bigger FAWT too.

Once the structure was fully assembled with the 4 modules, we realized there were a lot of cables. It was quite a mess, so we decided to introduce a number of supplementary components to facilitate things like cable management. This allowed us to organize everything a little better. However, using breadboards to

connect the fans with the Raspberry Pi was still quite chaotic. It was then that Daniel Alaez designed an interface board that supplied power and connected the fans to the control board. The FAWT looks like Figure 11:

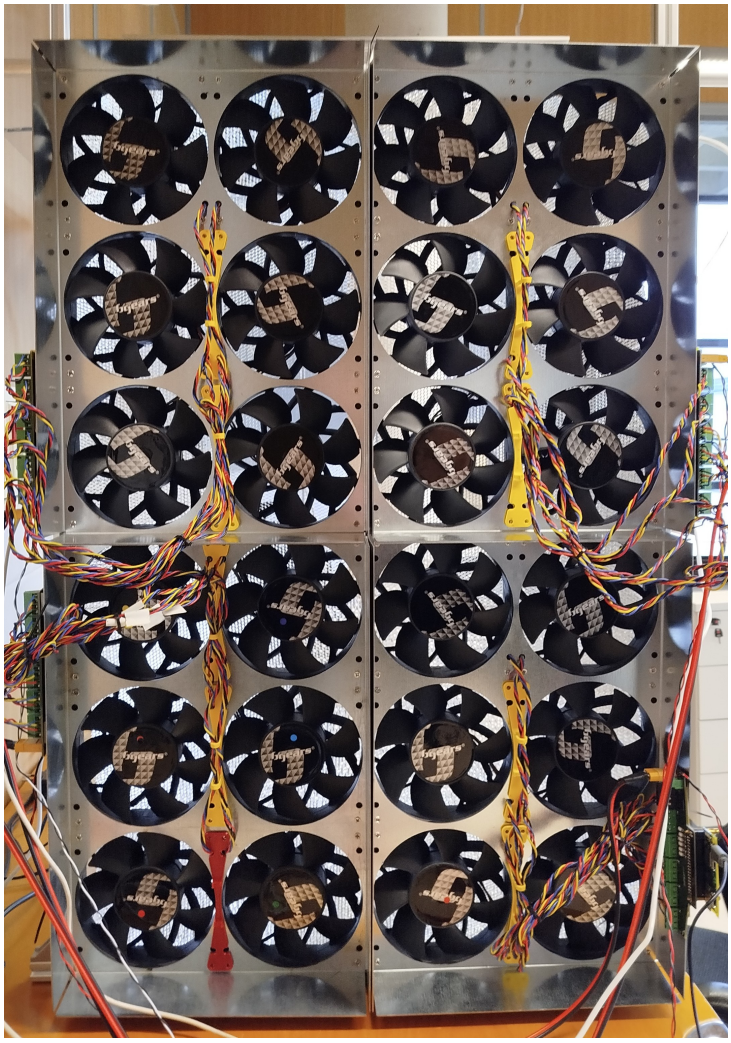

Figure 11: Photograph of the FAWT

This is the FAWT that we have been using all along the project. It consists of some modules, and the hardware components of these modules are as follow:

- The fans themselves.
- The control unit that acts as a client to receive commands from the control station (server).
- The interface board.

- The support and mounting structure itself.
- Supplementary components for organization.

These can be seen in a very visual way in Figure 12, which shows a sketch of the components that make up the FAWT and how they communicate with each other.

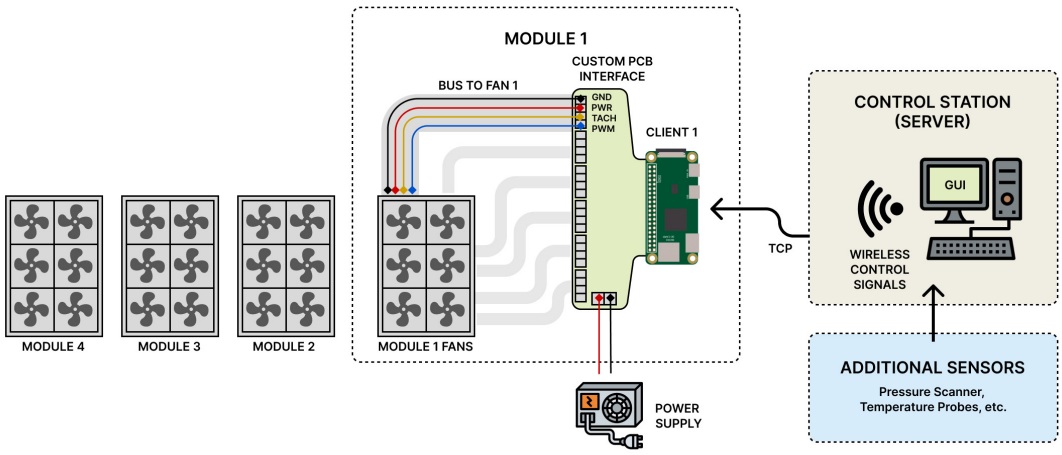

Figure 12: JavaFAWT architecture overview. Figure created by Daniel Alaez.

In summary, the server sends a packet to the client via the TCP connection. The client, i.e., the Raspberry Pi, receives it, which is a string composed of integers separated by commas. Depending on the client number, this Raspberry Pi will get different values. Let's imagine its client number is 2. Since each client consists of six fans, the first 6 values in the packet correspond to client number 1, while the ones from 7 to 12, correspond to this second client. Therefore, the Python program running on the client stores these speed values. Next, using functions from the *RPi.GPIO* library, it sends a PWM signal to each fan based on the speed specified in the packet. This signal is sent through the GPIO pins, which are connected to the blue wires of the fans. The fans receive the information and adjust their speed to match the new value. For all of this to work, the red wires of the fans are receiving power, and the black wires are grounded.

Depending on budget and desired application, the size of the FAWT can be scaled

up to as many modules as required. Thanks to the wireless connection, there is no need for physical wires to connect the server with each client, facilitating scalability. The different components that make up the proposed architecture are described below.

3.4.1 Interface Board

The interface board acts as the intermediate point between the Raspberry Pi (client) and the actual fans it controls. The board is designed to support the 3 most common Raspberry Pi variants: A+, B+ and Zero. In addition, this board is responsible for distributing the power supply to the six fans.

Since the Raspberry Pi gives a PWM control signal at 3.3 V by default, it is a requirement to raise this control voltage to the optimal fan control voltage. This is why the interface board is equipped with a second power input that elevates the voltage of each PWM control signal from the Raspberry Pi to the selected voltage.

For the fans chosen in the assembly (b-Blaster 140 mm), the optimum control voltage is 6V, which should be supplied via the control signal power connector. Given the low processing requirements, the inexpensive Raspberry Pi Zero 2W was selected as the client responsible for receiving the wireless control signals and converting them into PWM outputs for the motors. Figure 13.a shows an annotated image of the interface assembly with the Raspberry Pi Zero 2W. Figure 13.b shows a 3D visualization of the board with the different connections. At the right of Figure 13, we see a photograph of one of the boards that is part of our FAWT, with all its connections.

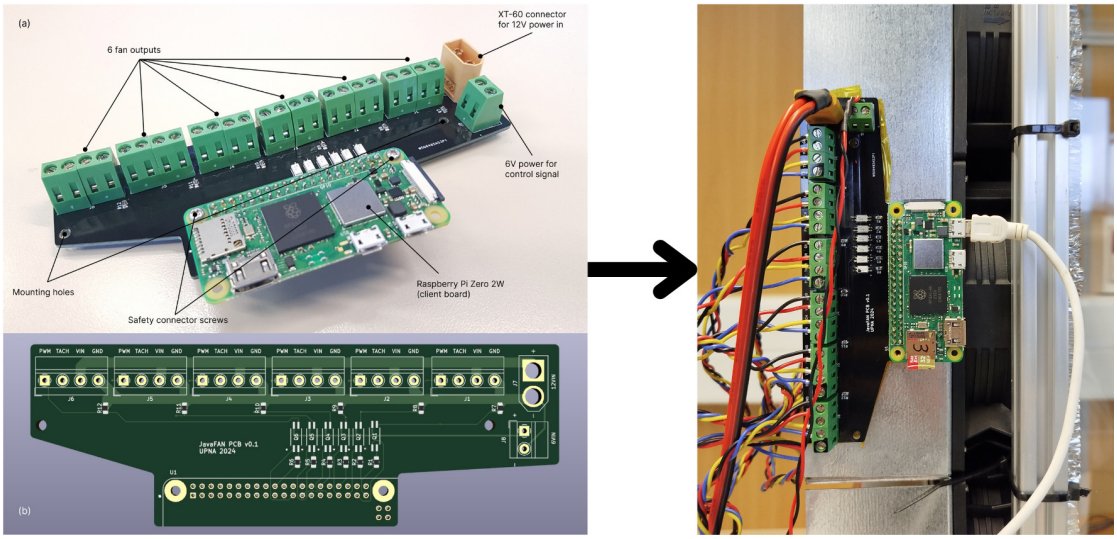

Figure 13: Interface board: (a) Annotated photograph of the assembly with a Raspberry Pi Zero 2W (b) 3D rendering of the PCB including connections. Figure created by Daniel Alaez.

For enhanced safety, the pins connecting the Raspberry Pi to the interface board can be anchored using spacers and screws through the mounting holes common to both boards.

3.4.2 Fan selection and Power supply

The main criterion for fan selection is the volumetric air flow rate metric. A power-efficient, reliable, widely available, low-cost fan has been sought. In addition, it should incorporate a PWM control cable and ideally a tachometer. Although there is a wide variety of alternatives on the market, the Bgears b-Blaster 140 mm in its 12 V variant was the chosen one, as previously mentioned, also for its dimensions.

Although a smaller diameter allows for finer scale flow control, it also reduces the frontal area of the tunnel, requiring more modules and their subsequent components to achieve the same frontal area. In any case, the structural unit of the module is provided in parametric design format, so that its dimensions can be easily adjusted to suit different fan sizes.

Regarding power supply, the fans have been dimensioned so that each module can easily be powered by a standard 12 V PC power supply. With a maximum power of approximately 33 W per fan, the total power of each module is just under 200 W, allowing even legacy PC power supplies to be reused. This saves assembly costs and gives a second life to older electronic components.

To ensure that the same control signal is provided to all tunnel fans, it is recommended to feed the control power port of the interface board from a single power supply for all modules. Thus, in the event of a slight voltage drop in the control power supply, it will affect all tunnel fans equally rather than a single module. Since the power consumption of the control signal is negligible, any voltage regulator can be used on a 12 V signal from any of the power supplies used to lower the voltage to 6 V and supply all the interface boards in parallel.

4 Development of the Application

After successfully managing fan speeds by sending packets, we need a simple yet effective application to help users control the fans effortlessly. Consequently, the application should be as visual and intuitive as possible.

4.1 Preliminary Decisions

The chosen language to develop the application is Java, for several compelling reasons (CodeJava, 2024; Training, 2024):

- Java offers platform independence, allowing the application to run seamlessly on any system with a Java Virtual Machine (JVM). This makes it ideal for supporting multiple operating systems, such as various versions of Windows and Linux.
- It provides a comprehensive set of networking libraries, including the *java.net* package, which simplifies the implementation of TCP/IP communication and socket management.
- Its concurrency features, such as threads and the *java.util.concurrent* package, facilitate smooth handling of multiple fan control commands and simultaneous TCP connections.
- Java's object-oriented approach allows for modular, maintainable, and reusable code. For example, each fan can be represented as an object with attributes like speed, buttons, or state flags.
- The Swing and JavaFX libraries enable the creation of cross-platform Graphical User Interfaces (GUIs), which are crucial for developing a consistent and functional control panel. While the GUI design may show minor visual differences across platforms, its overall functionality remains

the same.

- Java boasts a mature ecosystem with extensive documentation, libraries, and community support. This ecosystem makes it easier to integrate additional features, such as file browsing or drag-and-drop functionalities.

Once we were sure Java was the right programming language, we chose *NetBeans* as the Integrated Development Environment (IDE), primarily due to its frequent use throughout the computer engineering degree and its robust feature set (NetBeans, 2024):

- NetBeans is tailored for Java development, offering strong support for Java Standard Edition (SE) and Enterprise Edition (EE), along with tools for working with networking libraries like *java.net* for TCP applications.
- Its user-friendly interface provides a gentle learning curve for new developers. Built-in wizards and templates simplify the creation of projects, including those involving networking components.
- The IDE is cross-platform, supporting Windows, macOS, and Linux, which aligns with Java's philosophy of platform independence.
- NetBeans supports integration with frameworks and tools like Apache Maven, which can be beneficial if the TCP application evolves to include additional features.
- Its drag-and-drop GUI builder simplifies the design of user interfaces, making it easier to create control panels or monitor dashboards.
- As an open-source tool, NetBeans benefits from a vibrant community that provides plugins, documentation, and support, simplifying troubleshooting and enabling feature expansion.

4.2 Server and Connection

A single server has been designed to enable simultaneous wireless access and control of the pool of fans. This architecture allows the server to scale dynamically and control an almost unlimited number of fans. The server is an open-source Java application capable of running on various computers and operating systems. A Wi-Fi network is required on both the server and the clients to establish the necessary TCP connection.

To provide flexibility, we designed a preliminary interface that allows the user to select the desired port for the connection. A screenshot of this interface is shown in Figure 14. Users are expected to input valid port numbers based on basic networking knowledge. It is essential to understand that ports must be positive integers and that some ports, such as 22 (SSH), 25 (SMTP), 50 (Mail), or 80 (HTTP), along with many others, are reserved and should not be used. If an invalid port is entered, the application will not display an error, but the connection will fail, showing an error on the client's terminal. This feedback is sufficient to warn the user to correct the port number.

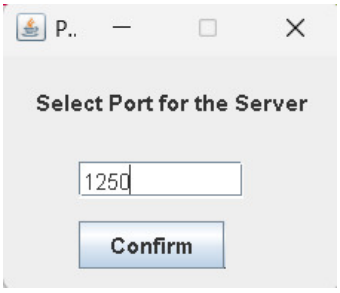

Figure 14: Interface for selecting the port number

After the port number is inputted, the server performs a scan of the machine's available IP addresses, selecting one that is not the local host. This ensures that external clients can establish a connection. While some servers may have multiple IP addresses, the server does not prioritize or select the best one; it only ensures

that the chosen address is not 127.0.0.1, which is restricted to local connections. Local host addresses limit connectivity to clients running on the same machine, excluding external devices like Raspberry Pis.

When a client establishes a connection, a unique number x is assigned to him based on the order of connection (e.g., 1, 2, 3, etc.), becoming client x. The client's information (IP, port, and number) is stored for later use. A list of currently connected clients is displayed in the interface in Figure 15, enabling the user to verify when all clients are ready.

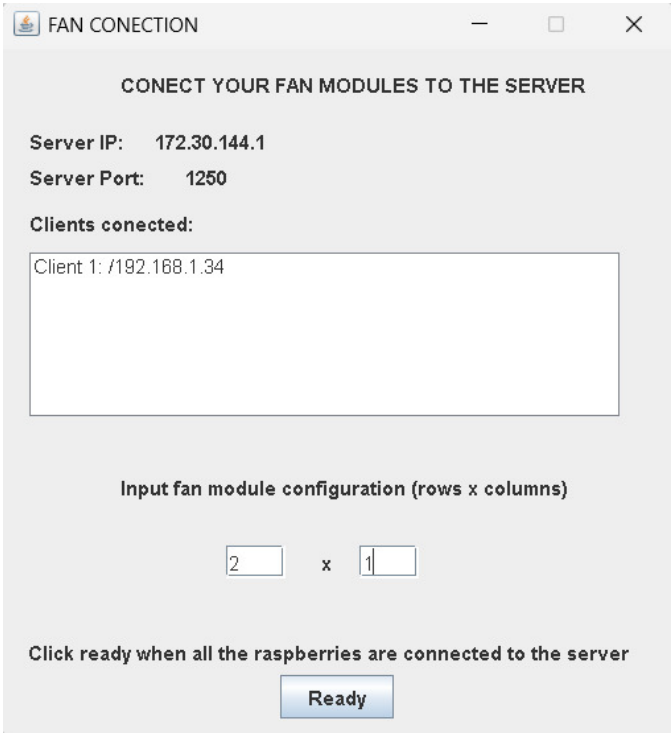

Figure 15: One client connected to the server

The same interface includes an essential feature of the FAWT control application: the ability to specify the dimensions of the fan wall, measured in terms of modules. For instance, if the FAWT has 4 rows and 3 columns, these values can be configured before running the main program. It is important to ensure that the configuration matches the number of connected clients. For example, setting a configuration of 3×3 implies 9 fan modules, but if only 4 clients are connected, only 4 modules

will be controlled, leaving the remaining 5 inactive.

Furthermore, the order of client connections determines the assignment of modules to clients. The assignment follows a systematic pattern: starting from the top-left module, moving right across the row, and proceeding down to the next row, continuing until the bottom-right module. To clarify, consider the Table 3 as an example of some connected clients:

Client Number	IP Address	Origin Port
1	192.168.1.1	2334
2	192.168.1.2	3218
3	192.168.1.3	1198
4	192.168.1.4	6735

Table 3: Example of Connected Clients

The number on each module identifies the client controlling it. If we input a configuration of 2×2, we will get the assignment shown on the left of Figure 16, and if we input a 3×3 configuration, we will get the one shown on the right.

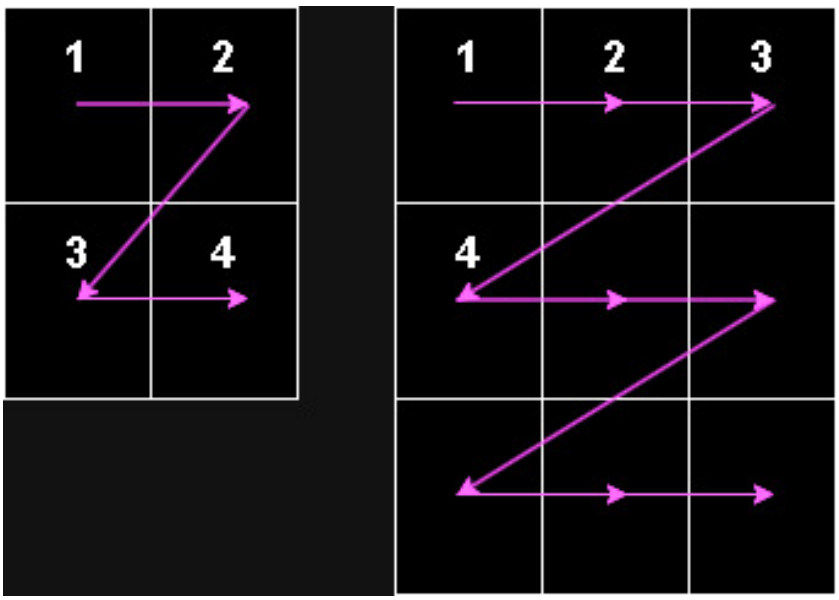

Figure 16: Client assignment for different module configurations. Own elaboration

As shown in the 3×3 configuration on the right, some modules remain unassigned because only 4 clients are connected. This kind of FAWT configuration is impractical, highlighting the importance of matching the number of clients to the selected configuration.

It is worth noting that the option to choose the configuration was not originally included in the application. Initially, the app defaulted to a 2×2 configuration with 4 modules. While this setup worked perfectly for our FAWT and similar builds, we had not anticipated its use by external researches until participating in the IMAV 2024 in Bristol. After that experience, we decided to introduce a feature to make the program flexible enough to work with different FAWT designs. Implementing this update required more time than expected, as it involved substantial changes to the codebase.

4.3 FAWT Control GUI

Imagine that all desired clients have already established a connection with the server, and the configuration has been successfully inputted. At this point, we want to stop accepting new clients and proceed with the application. So we transition to the main frame responsible for controlling the fans.

This frame has undergone the most significant number of modifications and surrounds the core logic of the application, aside from the connection-handling frames, which are relatively simpler in comparison. Initially, the primary goal was to enable users to directly adjust the speed of the fans through the application. This functionality had to be implemented in an intuitive and user-friendly GUI.

To achieve this, we used JButtons, with each button representing a specific fan. This layout was designed to allow users to easily update the speed of individual fans. The arrangement of the JButtons corresponds to the physical order of the

fans in the FAWT as we will see later in Figure 17. This design ensures that the representation in the app mirrors the real-world configuration, making it easier for the user to identify and control specific fans. Users can select fans individually by clicking on their corresponding buttons, set the desired speed using a JSlider, and send it (by clicking the *update* button). Additionally, a dedicated button allows all fans to be stopped simultaneously, if necessary.

While this basic functionality was a significant milestone, it presented several challenges during implementation. Upon achieving it, we recognized a substantial advancement in the app's development. However, the only way to observe the fans' spinning speeds was by physically looking at the real FAWT. This approach proved impractical when using the app remotely, as distinguishing the spinning speed of the fans visually is very difficult.

To address this, we decided to integrate real-time feedback into the app itself. A new frame was designed to display live data, using red-colored spheres to represent each fan. The radius of a sphere corresponds to the fan's spinning speed. A fan spinning at a higher speed will have a bigger sphere. Figure 17 illustrates an example with four active modules, where the fans are being controlled. At the right we can see the real FAWT.

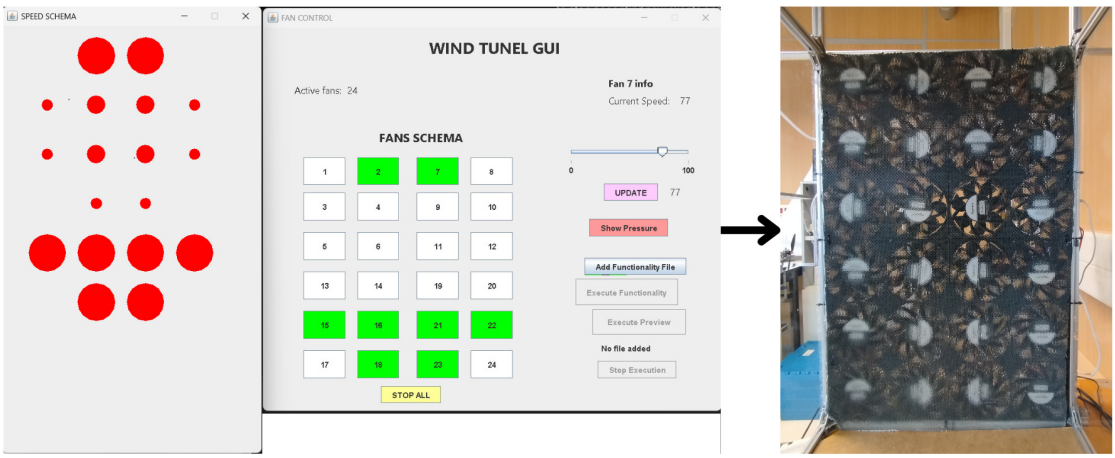

Figure 17: Old version of speed and control frame, and the real FAWT

- **Left window of the app**: This window displays the live spinning speeds of the fans. At a speed of 0, no sphere is visible. As the speed increases, the sphere grows proportionally, reaching its maximum size at a speed of 100. It is crucial to maintain the positional correspondence of the fans between both windows, ensuring the user can easily correlate their controls with the live display.

- **Right window of the app**: It is the user interaction Graphical User Interface (GUI) for controlling the fans. In the screenshot, some fans are selected, and their speed[1] is about to be updated via the JSlider.

- **Photograph of the FAWT**: This is a photo taken in the lab of the wall of fans we are using for the project. In this case, the photo is taken from behind, as this is where the wind flow is generated. The order of the fans is the same as the one we see in the application. Although it's not very clear in the picture, the wall of fans is equipped with the honeycomb flow straightener. Moreover, it is fully enclosed within a methacrylate structure to contain the wind inside.

4.3.1 Functionality Files

In the previous screenshot (17), you may have noticed some additional buttons on the control frame that we have not yet discussed. What is the purpose of the *Add Functionality File* button? And what exactly is a functionality?

Imagine you are testing a drone in the FAWT using the application. You want all fans to start spinning at zero speed. After one second, they should increase to 10, then to 20, and so on, until reaching 100. After that, the same pattern should repeat for a specified number of cycles. Manually updating the fans' speed for each step is tedious. Instead, it is much more efficient to create a file containing

[1]The fan speed ranges from 0 to 100, expressed as a percentage of fan's maximum capacity. For example, a speed of 30 corresponds to 30% of the fan's maximum speed.

these instructions and have the app read from it. Figure 18 shows an example of such a file.

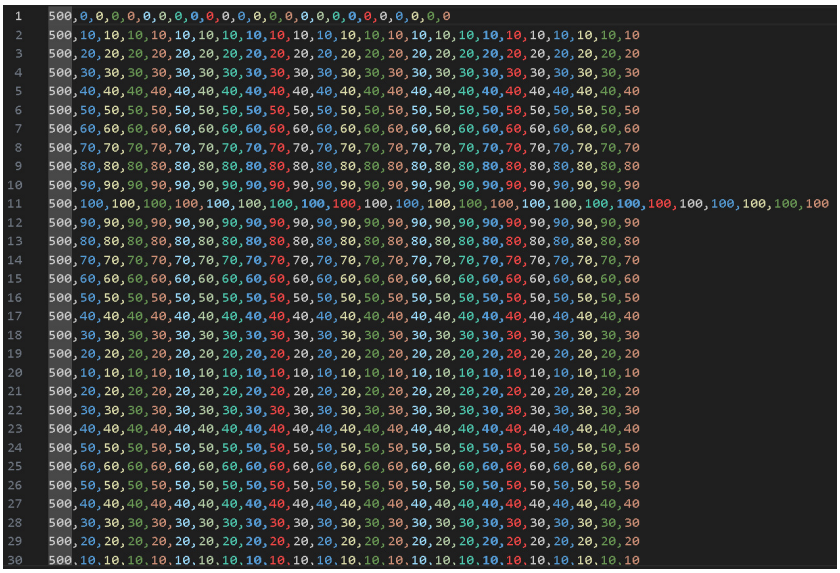

Figure 18: Example of a functionality file

The file is in Comma-Separated Values (CSV) format. It is a simple and easy-to-implement format. It has high compatibility with many programming languages, including Java and Python. Additionally, it takes up very little space, allowing for the storage of large amounts of data. Although it consists of lines of integers separated by commas, we treat it as if it is organized into rows and columns.

- The first column represents the time (in milliseconds) the app waits before sending the next set of values to the clients. In the example, this interval is consistently 500 milliseconds.
- From the second column onward, the values represent fan speeds. These follow the same structure as the messages sent to the clients for updating their speeds. Specifically, the first six values correspond to the fans of client 1, the next six to client 2, and so on.

Once the file is uploaded into the app, the functionality buttons will be enabled. These allow users to execute the functionality and observe the live fan speeds on the

live speed window. This approach enables the creation of custom airflow patterns simply by assigning desired speeds to the fans and specifying the duration for which those speeds should be maintained. It opens up a new realm of possibilities for testing and significantly enhances automation capabilities.

Preview Functionality:

Sometimes, we may want to test a specific functionality but are hesitant to execute it on the actual fans due to concerns about how they might respond. For this cases, we developed a preview mode. In this mode, the functionality is executed in a simulated environment, displaying the speeds in a new window without sending any commands to the clients. This allows users to evaluate the functionality visually within the app and decide whether to proceed with it or make some adjustments.

4.3.2 Pressure Files

Another feature we decided to add to the application is the integration of a pressure sensor. The use of such tools brings significant benefits to the project.

1. **Flow Uniformity Analysis:** Measuring pressure at various points allows for the evaluation of the airflow uniformity generated by the fans. This is essential for identifying areas with inconsistent flows or dead zones (where the flow is low or nonexistent).

2. **Identification of Structural Issues:** The sensor can help detect anomalies, such as blockages, air leaks, or defects in the fans. This allows for quick solutions before they significantly affect the system's performance.

3. **Design Optimization:** With precise pressure data, adjustments can be made to the placement, tilt, or speed of the fans to maximize system efficiency and minimize unwanted turbulence. This provides valuable feedback data

that can be used for various purposes. One of them is the automatic generation of wind flows using artificial intelligence. By having the fan speed values and the pressure readings provided by the sensor at the positions where we have chosen to place them, we can attempt to adjust the speeds to achieve the most customized and specific wind flows possible. This is not an idea to be deeply explored in this project, as it is completely beyond its scope. However, it is a task that should be considered for anyone who wants to continue working with the designed FAWT in the future.

We can find a wide variety of pressure sensors on the market, depending on their operating principle. Some of the most well-known examples are:

- **Piezoresistive Sensors:** These use the deformation of a piezoresistive element to measure pressure. They are accurate and suitable for measuring absolute, manometric, or differential pressures.
- **Piezoelectric Pressure Sensors:** These are based on piezoelectric materials that generate an electrical charge proportional to the applied pressure.
- **Capacitive Sensors:** These measure pressure by detecting changes in the capacitance of a flexible diaphragm when deformed under pressure.
- **MEMS Sensors (Microelectromechanical Systems):** These integrate a miniature circuit with piezoresistive or capacitive elements to measure pressure.

Among all these types, the chosen sensor was the *NanoDAQ-LTS-32*. This is a differential piezoresistive pressure sensor that uses MEMS (Microelectromechanical Systems) technology. Some of the reasons that led us to choose it are as follows:

1. **Advanced MEMS Technology:** The use of MEMS technology enables the sensor to be highly sensitive and efficient, ensuring reliable measurements

even in environments with fluctuating pressures or varying temperatures.

2. **Low Power Consumption:** Designed for continuous monitoring applications, this sensor consumes very little energy, making it ideal for systems that operate for extended periods or in conditions where energy efficiency is a priority.

3. **Long-Term Stability:** It offers high stability in its measurements, minimizing the need for frequent re-calibrations, which reduces maintenance costs and downtime.

4. **Resistance to Harsh Conditions:** It is designed to operate in a wide range of temperatures and pressures, ensuring reliable performance even in demanding industrial environments.

5. **Airflow Optimization:** By providing detailed pressure difference data, it allows for the quick identification and correction of issues such as turbulence or uneven flows in the system, contributing to overall efficiency.

6. **Cost-Effective:** Compared to similar technology sensors, the NanoDAQ-LTS-32 offers an excellent balance of price, accuracy, and functionality, making it an economical choice for projects requiring high precision.

In our case, we have decided to place the sensor's reading points, one for each fan as we see on Figure 19. In total, we use 24 sensors (we have 24 fans in our FAWT) out of the 32 reading points available. This is sufficient for our needs.

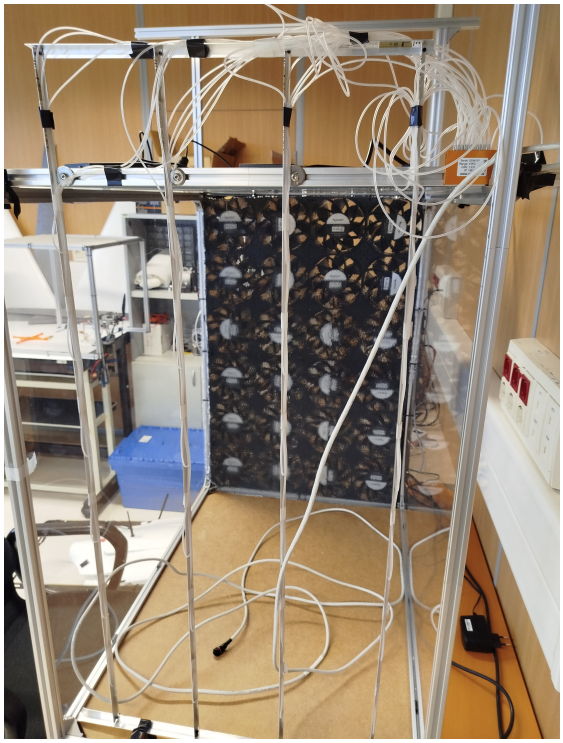

Figure 19: Photograph of the pressure sensor in the FAWT

The NanoDAQ-LTS-32 is the orange component located at the top right, from which all the white tubes come out. Each of these tubes is a pressure sensor, and its reading will be placed in front of one of the fans. The vertical lines are used to position these readings and place them in the desired location. This way, each fan has a pressure sensor placed in front of him to make the corresponding measurements.

Pressures in the Application:

But the question is, how is this technology implemented in the fan control application? It is done through the *Show Pressure* button, which we still need to discuss. This button allows for the real-time display of pressure values being read by the sensor.

To achieve this, the sensor must first be connected to the application. This is done using the window shown in Figure 20, where the IP address and port of the

pressure sensor are requested. Once entered, when the connection is established, the sensor readings at each point will be displayed on the screen.

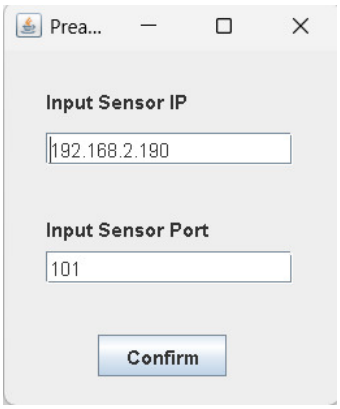

Figure 20: Interface for connecting a pressure sensor

These readings can also be accessed from the web browser, by entering the same IP address on the URL. It is recommended to access this page to configure certain parameters that will facilitate the analysis of the pressure values we will read. These configurations are completely optional but are very helpful. Some of them include, for example, changing the IP address or the port of the sensor. Modify the reading frequency for it to be more frequent or less, which is very helpful. As well as many more advanced configurations available. After customizing the pressure device, Figure 21 shows what is being read and displayed on the application .

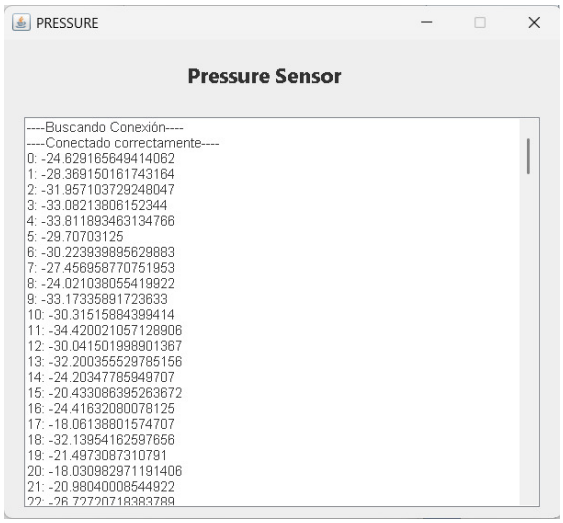

Figure 21: Preassure sensor readings

The numbers correspond to the sensors. Each time a pressure reading is performed in the application, all the information that has been read will be saved in a text file for subsequent analysis. This could serve for the idea we previously mentioned about integrating artificial intelligence, by processing this kind of files.

Application Demonstration:

After the development of this version of the application, a video was recorded. This video showed the program's functionality by capturing the screen of a laptop running the application. There, multiple Raspberry Pi devices are connected to the server, and the changes in fan speeds are displayed in the speed visualization window. Although the video is not particularly professional, it provides a clear idea of how the application functioned at this stage of development. While the real-time reaction of the fans is not visible, their sound can be heard in the background. The video is titled *Execution of the FAWT Remote Control Program* and can be found on YouTube.

4.4 Improving the Application

At this point, the application was already quite comprehensive, allowing significant control over the FAWT. However, there was still room for improvement. One notable enhancement was the ability to configure the fan wall dynamically based on the number of desired modules.

4.4.1 Dynamic Application

This improvement required transitioning from a static layout of four modules arranged in a 2×2 grid to a dynamic configuration selectable during client connection. Although this may seem trivial at first glance, it involved substantial changes to the application. The first step was to modify the connection frame,

enabling users to specify the desired module configuration. This flexibility accommodates users with varying resources, from setups with only two modules to those with significantly more. The main changes implemented in the application for this feature included:

- Relocating existing buttons to prevent interference with the fan modules.
- Adding new buttons to enhance the application's functionality.
- Replacing manually inserted JButton elements in the GUI with a script that dynamically generates and arranges JButton components based on the specified configuration.
- Modifying the fan speed visualization window to reflect the configured layout accurately.

Below, we provide an example of this configuration. Figure 22 is the connection frame that is configured for a 3×3 layout with only five clients connected. Figure 23 is the control frame.

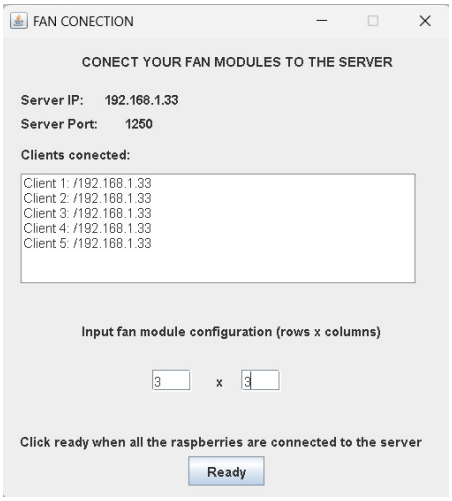

Figure 22: Connection frame with 5 clients connected in a 3×3 configuration

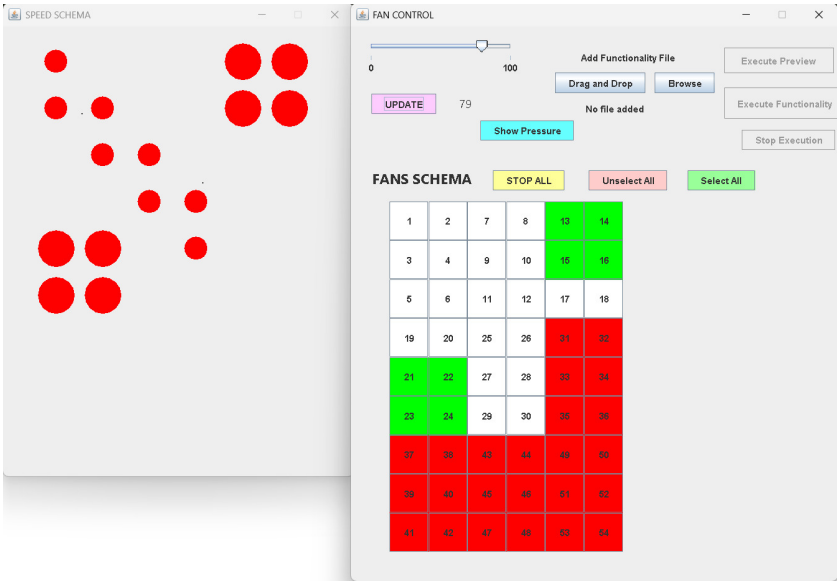

Figure 23: Dynamic version of the control frame

As shown, the program generates a 3×3 module configuration. However, since only five clients are connected, the remaining modules remain inactive due to insufficient client connections. This is why it is important to connect all the necessary clients.

This version of the application is significantly more advanced due to its adaptability. However, challenges remain for larger FAWT setups, which may exceed the screen's capacity. For now, the application works best with a moderately small number of modules. To address this, a new version was planned to accommodate larger setups and enhance the application's usability and professional appearance.

4.4.2 Final Improvements

Up until this point, the app consisted of two primary windows (omitting the connection part): the one to control the fans and the speed visualization one. If users wanted to preview a functionality, an additional window was created each time. Similarly, a separate panel was used to display the values read by the pressure sensor. This resulted in a cluttered application with too many individual windows and panels. To address this issue, it was decided to retain the initial connection

windows but consolidate the functionalities of the main windows into a single, compact, and less confusing layout.

Additionally, scroll panes were introduced, ensuring that regardless of the module configuration, users could view and control all the fans seamlessly. In the past version, if a client introduced a large layout like 8×8, most of the fans were impossible to see due to the limited space in the panel. But not anymore. Below, Figure 24 is a screenshot of this new frame with a 2×3 configuration and six connected clients.

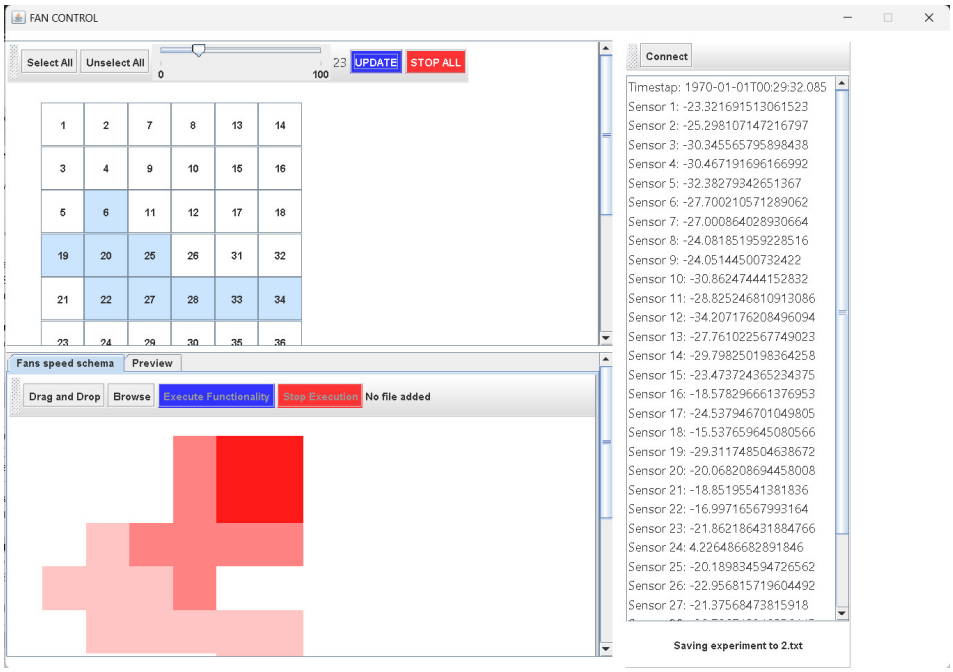

Figure 24: Last version of the control frame

Let us outline the main changes in the application compared to the previous version:

- As previously mentioned, what used to be separate windows; the fan speed control, real-time speed visualization, and pressure sensor readings are now integrated into a single one. The layout has been optimized to make the best use of the space available.
- The speed visualization has been completely revamped. Previously, speeds

were represented using spheres, where a bigger sphere indicated a higher fan speed. However, it was determined that there was a more intuitive way to represent this information. Speeds are now displayed as squares, with the intensity of the red color representing the speed. A higher fan speed corresponds to a more intense red.

Figure 24, shows some white areas (where the fans are stationary) and various shades of red intensity. For instance, the fans top right are spinning faster than those in the center, or at the left. This approach makes the visualization more intuitive.

- All buttons that previously floated around the GUI have now been organized into toolbars, giving the application a much more professional appearance. The button colors have also been adjusted to make them more visually appealing.

- At the time of the screenshot, the pressure sensors was connected. We can see its readings, as well as the timestamp, that will be updating every so often. All these values are being stored in a file named *2.txt*. So that we can analyze them later if necessary.

- The functionality handling has also been slightly modified. It is now accessible directly from the bottom-left window. If users wish to preview a functionality, they can switch to the *Preview* section, which allows to start and stop the functionality while observing the generated speeds.

- Finally, scroll panels have been added throughout the interface to ensure that no information is lost and to avoid using excessively large windows. This keeps the application size fixed and manageable.

With these final changes, the application was considered complete.

5 Experiment

At this point, it was decided to test the functionality of the application through an experiment. This step is essential because having a fan wall and a remote control application is meaningless if it cannot be tested. With this experiment, we aim to demonstrate that the application we have developed provides absolute control over what happens inside the wind tunnel. Furthermore, we want to highlight that this project opens a world of possibilities for what can be achieved.

For this study, we have our fully assembled and operational FAWT, where the pressure sensors are placed in front of each fan. Likewise, we have our remote control application for the fans, that has been developed during the project, and is ready to use.

5.1 Setup and Preparation

The idea is to connect the four Raspberry Pis of the FAWT to the application server, allowing us to control the four modules that make up the fan wall. Once this is done, we will select a 2×2 configuration and proceed to the main window.

Before starting the experiment, we will verify that the clients have been connected in the correct order and that the actual fans correspond to the layout displayed in the application. To do this, we will change the speed of each fan and check that the one we selected is indeed the one moving. If not, we will need to determine whether the issue lies in the clients being connected in the wrong order or if the problem goes further and is related to the fan order within the module. In the first case, the solution is straightforward: reconnect the clients in the correct order. In the second case, we will need to rearrange the fan cables on the interface board or make the necessary adjustments to ensure the fans match the layout shown in

previous Figure 9.

Next, we will create a functionality or speed file. In the experiment, we aim to test with the fan speeds. To do this, we will increase all the speeds simultaneously, assign different speeds, and leave some fans moving while others are stationary, to observe how the system reacts. To achieve this, we will use the functionality file shown in Figure 25.

```
2000,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2000,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
2000,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20
2000,30,30,30,30,30,30,30,30,30,30,30,30,30,30,30,30,30,30,30,30,30,30,30,30
2000,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40,40
2000,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50,50
2000,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60
4000,10,20,30,40,50,60,70,10,20,30,40,50,60,70,10,20,30,40,50,60,70,10,20,30
5000,40,0,40,0,40,0,40,0,40,0,40,0,40,0,40,0,40,0,40,0,40,0,40,0
5000,0,0,50,0,0 [illegible] 50,0,0,50,0,0,50,0,0,50,0,0,50,0,0,50
5000,0,0,0,70,0 Col 7: 0 0,0,0,70,0,0,0,70,0,0,70,0,0,0,70,0
1000,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Figure 25: Screenshot of the functionality file for the simulation

1. The file starts with all fans still.
2. After two seconds, the speed of all fans increases in increments of 10 until it reaches 60% [2], with each interval lasting two seconds.
3. Then, for four seconds, the fans will run at different speeds, with each fan spinning at the assigned values, some faster and others slower.
4. The next three lines leave some fans stationary while others spin. Each line assigns specific speeds to certain fans and stops the rest.
5. Finally, all fans are stopped, meaning the end of the simulation.

The last step of the preparation is to ensure that the pressure sensor is turned on and that its readings are reliable. This can be done via the web browser by entering its IP address or through the same remote control application. Additionally, the sensors need to be calibrated to make sure their readings show differential pressure values. This means they measure the pressure difference between the monitored

[2]We do not want speeds exceeding 60% on all fans to avoid generating excessively strong wind. Additionally, higher speeds are unnecessary for the specific goals of this experiment.

environment and the ambient atmospheric pressure, with a baseline of zero Pascals (Pa) under normal ambient conditions.

5.2 Execution

During the entire execution process, we will capture two types of recordings. The first will be a screen recording of the laptop running the remote program. The second will be a recording with a mobile phone camera pointed at the FAWT, aiming to observe the rotation of the fans.

We will start by connecting the pressure sensor to the application as described in previous sections. The sensor readings will begin to appear on the screen, and the system will indicate the text file where they are being saved. At this point, we will load the functionality (or speed) file into the system and execute it. Throughout its execution, we will observe changes in the speeds of the actual fans, which will correspond to what we see in real time in the application.

Once the process is complete, we will edit both videos and upload them as a single video to YouTube. The video is titled *Simulation of Remote Control of FAWT*. Watching the video is highly recommended, as it provides a much clearer understanding of the simulation conducted and the results obtained.

5.3 Idea for the Analysis

As mentioned earlier, the 24 pressure sensors are intentionally placed in front of each fan. Consequently, a sensor located in front of a fan that is turned off will detect less pressure compared to one in front of a fan spinning at maximum speed. However, since the fans (and therefore the sensors) are positioned very close to each other, what affects one sensor also impacts its neighboring sensor, but to a lesser degree.

With this idea in mind, we aim to observe the variations in the fans' speeds as well as the pressure variations detected by the sensors for each fan. All this is analyzed over the duration of the simulation. This demonstrates that our project also encompasses a significant area of data analysis. This opens the door to the incorporation of machine learning and artificial intelligence models, which could enable a more in-depth study of this data and provide more comprehensive insights.

6 Results

The pressure file generated with the simulation is very large. Nevertheless, we will show a few lines to illustrate how it is structured in Figure 26.

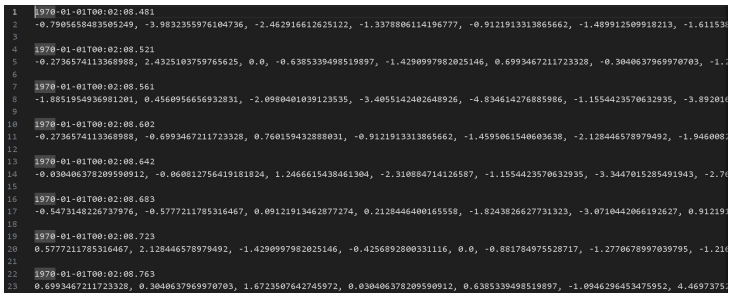

Figure 26: Some lines from the pressure file of the simulation

Each line starts with the timestamp, which indicates the exact moment when the reading was taken. Following this, the values read by each sensor are shown in Pascals. Each reading is approximated to 16 decimal places, which is why only some of the values are visible in the image. The readings are taken every 40 milliseconds, making the file quite large. This particular file contains around 1000 readings.

6.1 Data Processing

It is very difficult to work with a file of such size and characteristics. Therefore, a Python program was developed to process the file and modify it into one that we could work with. The changes made on the previous file returned the file represented in Figure 27:

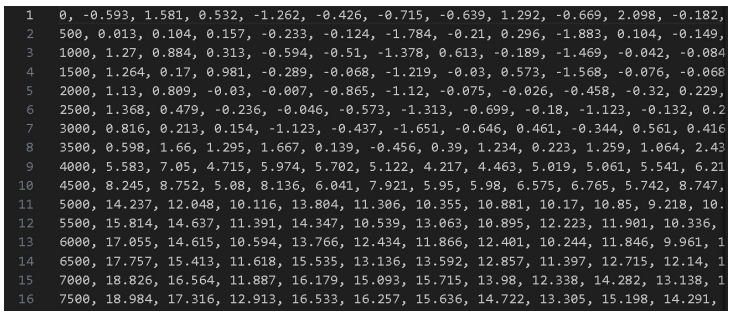

Figure 27: Some lines from the modified pressure file

- Each line of the file corresponds to a specific reading, reducing the number of lines.
- The number of decimals is reduced to 3, as this is more than sufficient for our needs.
- Lines of readings taken before the functionality had been executed, are removed. These lines were all very similar and did not add any value, as the experiment had not yet begun.
- To avoid redundancy in the data, readings within a 500 millisecond range are averaged. This also reduces the total number of lines and computation time.
- Instead of using timestamp values, they are replaced with values in milliseconds, starting at zero and increasing by 500 milliseconds. This structure matches the timing format in the speed file, making it easier to work with both files.

6.2 Plot with Results

After these changes, we decided to plot the results obtained from the experiment. The plot shows the variation in the speed of each fan and the pressure variation of each sensor (in front of those fans) over time. It is not as simple as it seems, taking into consideration all the information that needs to be displayed. There are 24 fans and 24 sensors, each with its values over time.

For this reason, we decided to use two separate plots. The upper one shows the speed variations of each fan, and the lower one shows the pressure variations. Each fan is marked with a color to identify it, which matches the color of its corresponding sensor. The resulting plot is shown in Figure 28.

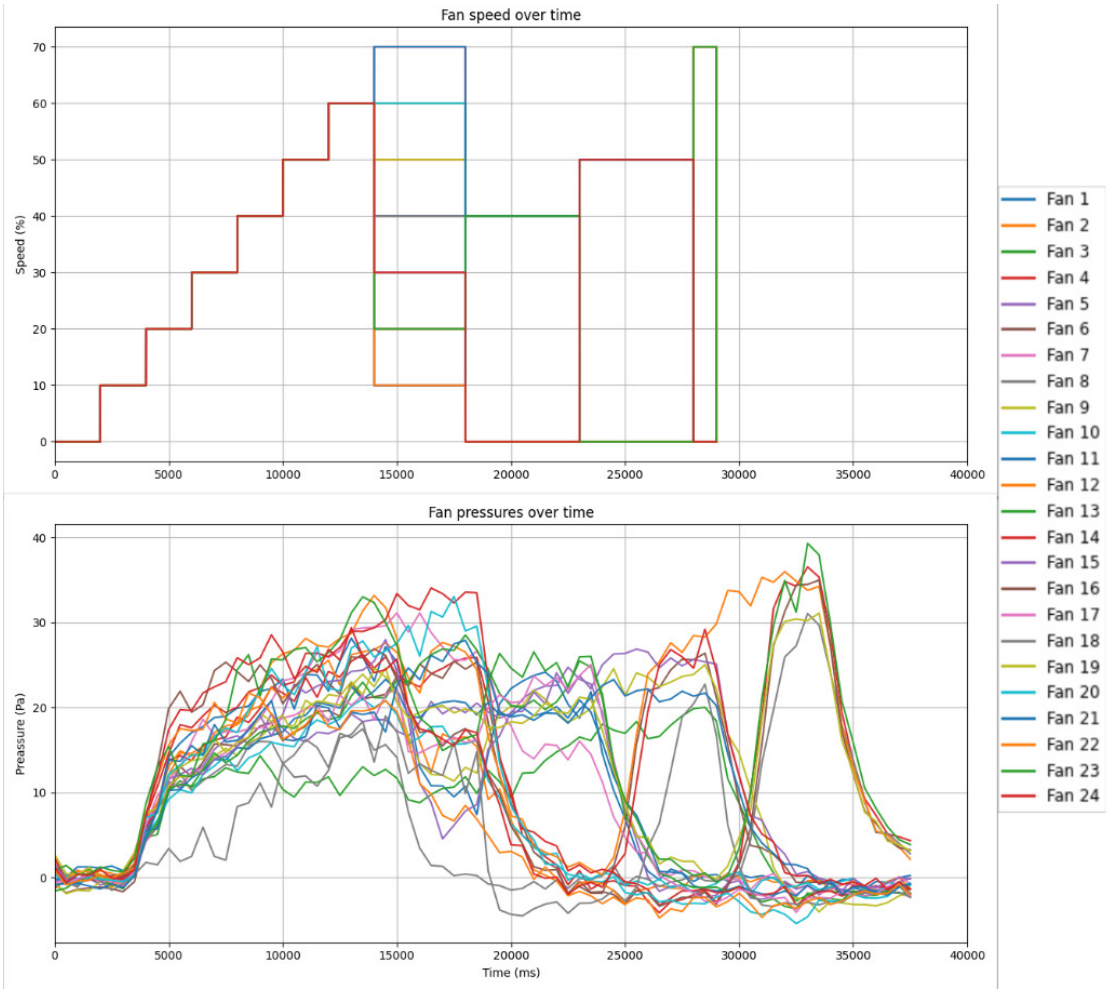

Figure 28: Plot with the experiment results. Own elaboration

6.3 Analysis of Results

In the upper plot, we observe that the fans initially remain stationary. Over time, their speed gradually increases until reaching 60% of their capacity, which occurs at 12,000 milliseconds. This increase is neither linear nor progressive, as the speeds increment in steps of 10, resulting in a staircase-like pattern. Since all the fans behave identically, the plot displays a single red line representing fan number 24,

the last one.

Looking at the lower plot, the pressures do not increase as rapidly. This is because the fans require time to accelerate; the process is not instantaneous. At 4,000 milliseconds, they begin spinning at a sufficient speed, and from that moment, we observe an increase in pressures. Each sensor reacts differently due to their varied positions, with some being more or less influenced by nearby fans. Additionally, some fans may produce more airflow than others even when receiving the same PWM signal.

The gray line, significantly below all the other lines in the pressure plot, exemplifies this behavior. Although both fan 8 (sensor positioned in front of fan 8) and fan 18 share the same color, analysis of the pressure data confirms that this line corresponds to fan 18. This fan, which appears to generate less airflow than the others, is located in the lower section of the FAWT, as shown in Figure 17 referenced earlier. While this is not ideal, we acknowledge that such variations can occur and must be considered. Fans are mechanical devices and, therefore, not perfect.

The pressure sensor readings increase steadily until 14,000 milliseconds, at which point each fan is assigned a different speed. This transition is evident in the upper plot due to the number of distinct lines observed during these 4 seconds. Similarly, in the pressure diagram, this variability is reflected as each sensor experiences different airflow patterns.

Following this phase, a new stage begins, during which specific speeds are assigned to some fans while others are stopped. In the upper plot, at 18,000 milliseconds, we observe that some lines reach a speed value of 40 (the assigned value), while others drop to 0. This pattern repeats later for speed values of 50 and then 70, forming shapes resembling rectangles. After this phase, all fans are stopped.

In the lower diagram, although less distinctly, we observe triangles with rising and falling slopes instead of rectangles. These triangular patterns start to appear from 18,000 milliseconds onward. The representation is not perfectly symmetrical, as the fans require some time to reach their assigned speeds. Nevertheless, the similarity between both diagrams is apparent.

Finally, when the fans are stopped, this is not immediately reflected in the pressure sensors. Instead, the pressures gradually decrease to zero, marking the end of the simulation.

7 Conclusion

The main objective of the project was to build a customizable FAWT capable of being wirelessly controlled. This posed a challenge, primarily due to the dual nature of its scope. On one hand, there is the electronics domain, which focuses more on the construction of the wind tunnel, and on the other, the informatics, more centered on the application. However, informatics was also required in certain parts of the tunnel. Therefore, synchronizing both domains was not an easy task, as it sometimes led to bottlenecks in one of the areas. To test certain functionalities of the application, some parts of the tunnel had to be completed, and vice versa. Nevertheless, thanks to frequent communication and teamwork, excellent synchronization was achieved in this regard.

This resulted in meeting the initial objectives, obtaining a fully usable fan tunnel ready for testing. Furthermore, this structure is easily and fully remotely controllable via an application. This app not only allows individual speed adjustments but also provides real-time fan representation, functionality file input (with preview), and pressure sensor readings. This last aspect is one of the most relevant ones of the project, which, interestingly, emerged during its development. In the initial plans, there was no indication that pressure sensors could be added to the wind tunnel. For this reason, it is also considered that, apart from achieving the initially proposed goals, the project exceeded expectations.

Additionally, we conducted an experiment where the FAWT was remotely controlled through the application. This trial demonstrated the experimentation capabilities of the developed product. The fans responded precisely to the commands issued by the application, generating the desired wind flows. Meanwhile, the sensors accurately captured these wind patterns, highlighting areas where the airflow had greater or lesser effects. This opens up a range of possible

tests and potential future improvements.

7.1 Shaping Tomorrow

Regarding the FAWT, the most significant improvement that can be made is its expansion. As we know, the tunnel currently consists of a total of 4 modules. While this is not exactly small, it is not as large as we would like it to be. Therefore, the idea of increasing the number of modules in the tunnel is something to take into account. This is particularly important given its primary goal: serving as a testing platform for drones. With the current dimensions, the drones that can be tested inside need to be very small. However, if we aim to broaden the range of drones to be tested, this change is essential. It is also important to consider that the FAWT cannot be excessively large, as it must fit within the laboratory space.

Regarding the application, there are always opportunities for improvement, especially when it is offered to users. At that point, requests for enhancements or errors we had not previously encountered start to come up, leaving room for future updates. One such possible update is the integration of emerging technologies, such as machine learning and artificial intelligence models. These tools would allow us to analyze the application's feedback more comprehensively, through real-time fan speeds and pressure sensor readings. By processing all this information with AI models, we could potentially generate more customized wind flows, thereby enhancing the FAWT's potential for testing drones within its structure.

References

Barlow, J. B., Rae, W. H., & Pope, A. (1999). *Low-speed wind tunnel testing*. John wiley & sons.

bgears. (2024). B-blaster product page [Accedido: 23 de diciembre de 2024]. https://www.bgears.com/b-blaster/

Brandt, J., & Selig, M. (2011). Propeller performance data at low reynolds numbers. *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 1255.

Catry, G., Ceyhan, O., Noca, F., Bosson, N., Bardazzi, L. J., Marquez, S., Jordaens, P. J., & Brandolisio, D. (2021). Performance analysis of rotorcraft propulsion units in a combination of wind and icing conditions. *AIAA AVIATION 2021 FORUM*, 2677.

Cattafesta, L., Bahr, C., & Mathew, J. (2010). Fundamentals of wind-tunnel design. *Encyclopedia of aerospace engineering*, 1–10.

Cermak, E. (2003). Wind-tunnel development and trends in applications to civil engineering [Accessed: 2025-01-06]. *Journal of Wind Engineering and Industrial Aerodynamics*, *91*(3), 355–370. https://doi.org/10.1016/S0167-6105(02)00396-3

CircuitPython, A. (2024). *Circuitpython emc2101 api documentation* [Accessed: 2024-12-29]. https://docs.circuitpython.org/projects/emc2101/en/latest/api.html

CodeJava. (2024). Java socket client examples (tcp/ip) [Last accessed: 23 Dec 2024]. https://www.codejava.net/java-se/networking/java-socket-client-examples-tcp-ip

Contributors, L. (2024). *Lgpio documentation* [Accessed: 2024-12-29]. https://rpi-lgpio.readthedocs.io/_/downloads/en/latest/pdf/

Coutu, D., Brailovski, V., Terriault, P., Mamou, M., & Mebarki, Y. (2011). Aerostructural model for morphing laminar wing optimization in a wind tunnel. *Journal of aircraft*, *48*(1), 66–76.

Dantec Dynamics. (2024). Dantec dynamics - measurement solutions for fluid mechanics and combustion [Accessed: 2025-01-16]. https://www.dantecdynamics.com

Dougherty, C., Veismann, M., Stefan-Zavala, A., Renn, P., & Gharib, M. (2020). Fan array wind tunnels: Of mice and mars (and machine learning too). *APS Division of Fluid Dynamics Meeting Abstracts*, T08–006.

Foundation, R. P. (2024). *Raspberry pi documentation* [Accessed: 2024-12-29]. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

Johnson, E., & Jacob, J. (2009). Development and testing of a gust and shear wind tunnel for navs and mavs. *47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*, 64.

Klein, V., & Murphy, P. C. (1998). *Aerodynamic parameters of high performance aircraft estimated from wind tunnel and flight test data* (tech. rep.).

Mehta, R. D., & Bradshaw, P. (1979). Design rules for small low speed wind tunnels. *The Aeronautical Journal*, *83*(827), 443–453.

Neal, D., Good, M., Johnston, C., Robertshaw, H., Mason, W., & Inman, D. (2004). Design and wind-tunnel analysis of a fully adaptive aircraft configuration. *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, 1727.

NetBeans, A. (2024). Apache netbeans [Accedido: 2024-12-23]. https://netbeans.apache.org/front/main/index.html

Nicolosi, F., Corcione, S., Della Vecchia, P., et al. (2014). Commuter aircraft aerodynamic design: Wind-tunnel tests and cfd analysis. *29th ICAS (International Council for Aeronautics and Astronautics) Conference*, 1–13.

Noca, F., Bujard, T., Visvaratnam, G., Catry, G., & Bosson, N. (2021). Flow profiling in a windshaper for testing free-flying drones in adverse winds. *AIAA Aviation 2021 Forum*, 2577.

One Air Aviation. (2025). One air flight academy - pilot training school [Accessed: 2025-01-16]. https://www.oneair.es

Ozono, S., Nishi, A., & Miyagi, H. (2006). Turbulence generated by a wind tunnel of multi-fan type in uniformly active and quasi-grid modes. *Journal of Wind Engineering and Industrial Aerodynamics*, *94*(4), 225–240. https://doi.org/ https://doi.org/10.1016/j.jweia.2006.01.010

Pi4J. (2012). Model zero w - pinout for raspberry pi zero w rev 1.0 [Accessed: 2025-01-16]. https://www.pi4j.com/1.2/pins/model-zerow-rev1.html

Poisson-Quinton, P. (1968). From wind tunnel to flight, the role of the laboratory in aerospace design. *Journal of Aircraft*, *5*(3), 193–214.

Russell, C. R., Jung, J., Willink, G., & Glasner, B. (2016). Wind tunnel and hover performance test results for multicopter uas vehicles. *American Helicopter Society (AHS) International Annual Forum and Technology Display*, (ARC-E-DAA-TN31096).

System, A. L. (2020). *Emc2101 fan controller and temperature sensor guide* [Accessed: 2024-12-29]. https://learn.adafruit.com/emc2101-fan-controller-and-temperature-sensor?view=all

Theys, B., Dimitriadis, G., Andrianne, T., Hendrick, P., & De Schutter, J. (2014). Wind tunnel testing of a vtol mav propeller in tilted operating mode. *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 1064–1072. https://doi.org/10.1109/ICUAS.2014.6842358

Training, L. (2024). Building java client-server applications with tcpb [Accedido: 2024-12-23]. https://luxoft-training.com/blog/building-java-client-server-applications-with-tcpb

Veismann, M., Dougherty, C., Rabinovitch, J., Quon, A., & Gharib, M. (2021). Low-density multi-fan wind tunnel design and testing for the ingenuity mars helicopter. *Experiments in Fluids*, *62*(9), 193.