23rd International Conference on Renewable Energies and Power Quality (ICREPQ'25)
Tenerife (Spain), 25th to 27th June 2025
ISBN-13: 978-84-09-66511-2

# On the IoT architecture for real-time monitoring and control of offshore wind turbines using multithreaded low-cost microcontrollers

B. Sánchez Centeno[1], M. Fernández de Diego[1], M. Santos Peñas[2] and S. Esteban San Román[3]

[1] Computer Sciences Faculty
University Complutense of Madrid
28040 Madrid (Spain)

[2] Institute of Knowledge Technology
Complutense University of Madrid
28040 Madrid (Spain)

[3] Faculty of Physics,
Complutense University of Madrid
28040 Madrid (Spain)

**Abstract.** This paper presents an IoT architecture for real-time monitoring and control offshore wind turbines using multithreaded low-cost boards. The system aims to address the challenges of offshore wind energy, particularly for deepwater and floating turbines, by developing an affordable solution that enhances reliability, maintainability, safety and efficiency. The architecture consists of two levels: a low-level system for local data collection and control and a high-level system for real-time monitoring and command execution. A key innovation is the use of a simple, low-cost ESP32 microcontroller to handle dedicated algorithm loops for turbine operation. Prototype components, microcontroller specifications, software implementation and graphical user interface design are detailed below, demonstrating the feasibility and advantages of this approach for the offshore wind industry. By reducing costs and improving remote monitoring capabilities, this architecture has the potential to drive investment and accelerate the deployment of renewable energy.

**Key words.** IoT, low-cost microcontrollers, multithreading, data monitoring, offshore wind energy.

## 1. Introduction

Faced with the health and environmental damage caused by fossil fuels, the challenges of extraction and transportation, and the coercion exerted by fossil fuel producers, Europe must transform its energy grid. The transition to renewables is more necessary than ever, with the goal of making gas, oil and coal residual in the electricity market. Wind energy presents a significant potential, particularly with the expansion of deepwater turbines. These offer higher efficiencies and reduced environmental impact [1]. However, they also introduce new challenges, such as complex control operations, due to the harsh marine environment [2].

Over the past decade, offshore wind energy has grown by nearly 30% annually [3]. This expansion has been driven by industry investments and technological advancements, enabled by larger sizes and the absence of topographical constraints. Wind currents at sea are stronger and more consistent, making offshore installations particularly advantageous in terms of performance. However, competitiveness with conventional energy sources was not achieved until 2017. High distribution and maintenance costs, exacerbated by strong winds, waves, and ocean currents, continue to hinder development [4].

Bottom-fixed wind turbines are considered economically and technically viable in waters up to 50 meters deep [5]. This constraint limits installations to coastal areas, often leading to conflicts with urban centers, high-traffic fishing zones or protected habitats. But advancements in floating wind turbine technology are addressing these limitations. This emerging offshore solution allows deployment in waters up to 1.000 meters deep, unlocking previously inaccessible deep-water sites. Although still in its early stages, floating wind technology holds exceptional potential for the industry's future.

The objective of this work is to develop a low-cost system for monitoring and controlling offshore wind turbines. The key innovation of this research lies in affordability: demonstrating that a simple, low-cost microcontroller can handle a dedicated algorithm loop for the well-functioning of an offshore wind turbine.

Reducing costs will drive investment and accelerate deployment, specialized hardware and architectural designs will enhance reliability and maintainability, and advanced control and monitoring strategies will improve safety and efficiency.

The project is structured into two levels. The low-level system focuses on local data collection and control using inertial or voltage measurement sensors to regulate variables such as pitch and load, forming an autonomous control loop. The high-level system enables real-time turbine monitoring and facilitates command execution within the control loop.

## 2. Architecture

The low-level architecture consists of a physical prototype connected to a microcontroller that hosts the control cycle and the communication with the server. The high-level architecture consists of MATLAB ecosystem services that display data from the server and allow bidirectional communication.

### *A. Prototype*

The prototype consists of several components that simulate an offshore wind turbine and serve to test the developed software.

1) *Generator*: The brushless DC motor is the component in charge of generating a three-phase signal. This signal has an amplitude and frequency that determine the rotational speed.
2) *Pitch actuator*: This component consists of a micro-servomotor that is responsible for adjusting the pitch angle of the wind turbine blades. The pitch adjustment controls the speed and efficiency of the generator, as it modifies the angle of attack of the blades with respect to the wind. Different pitch positions can be simulated depending on the energy or safety needs of the wind turbine.
3) *Load actuator*: This system is composed of a set of 4 relays that provide up to 16 different states. Its main function is to regulate the electrical resistance of the generator motor, which directly influences the power drawn from the generator and the load applied. Different load conditions can be simulated, mimicking the behavior of the generator against different levels of power demand or linear wind momentum.
4) *Inertial measurement sensor*: An inertial measurement unit (IMU) is anchored to the floating base of the prototype and its function is to measure the linear and angular accelerations generated by the vibrations and the simulated wave. This information is used to evaluate how variations in the base motion affect the performance of the wind turbine and how operating conditions could be optimized in a real marine environment.

In addition, the prototype has a small integrated display that provides real-time local monitoring. This display allows for quick and easy viewing of basic information on component status and generator performance, facilitating immediate evaluation and adjustments without the need to connect external devices.

### *B. ESP32 microcontroller* [6]

All these components are interconnected by means of a PCB board that integrates a two-core ESP32 microcontroller. See Figure 1.

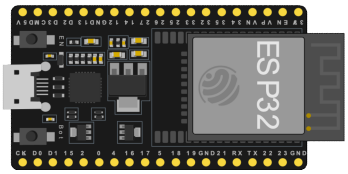

Fig. 1. ESP32 board.

The ESP32 microcontroller features two Xtensa LX6 processors. Thanks to its dual-core architecture and its native operative system FreeRTOS, each core can execute separated tasks that can be prioritized and dynamically distributed, maximizing the use of the available hardware. For example, one core can be dedicated to control sensors and actuators and the other to manage connectivity.

It is able to handle serial communication protocols, with up to 38 input and output pins, and supports Wi-Fi and Bluetooth. Its operating voltage varies between 3,3V and 5V and its power consumption is highly efficient: in idle state its current consumption is less than 5µA, while in full operation it reaches a maximum of 0,5A.

These features, together with its small size and resistance to extreme temperatures, make the ESP32 a central element in the development of IoT technologies, industrial automation and advanced embedded systems.

### *C. MATLAB*

The MATLAB ecosystem is in charge of data management outside the prototype. The database server is hosted on the ThingSpeak platform. The graphical interface that interprets and modifies this database has been developed in MATLAB. Through the interface, it is possible to visualize the collected data, make real-time adjustments and generate detailed performance analysis.

## 3. Embedded Software implementation

The code is loaded into the microcontroller built into the turbine prototype and runs it indefinitely from the moment it is connected to power. A real wind turbine could run it continuously from the installation by interposing scheduled safety restarts. The flowchart of Figure 4 is intended to show the skeleton of the program in a simplified way to achieve greater expressiveness. It makes direct use of the shared queues that communicate both processes bidirectionally by message passing.

Before we begin, it is worth remembering that the free ThingSpeak license limits the period between data uploads to 15 seconds and the granularity of sampling to 1 second. The period will be extended from 15 to 20 seconds to avoid coordination problems, and these limitations will be bypassed by jointly uploading, in JSON text format, sets of 20 data every 20 seconds.

Using FreeRTOS libraries, the program starts by declaring the two queues in charge of communicating the processes of both threads. They correspond to two unidirectional channels constantly accessed by both processes. This implementation ensures that the operational control is never interrupted.

On the one hand, the state channel starts from the control process and goes to the communication process. This design is presented in Figure 2. It has the capacity to hold a sufficient number of state instances so that it does not fill up in the usual time it takes to write the state to the ThingSpeak server. In this case, it is made to coincide with the limitation period between uploads so that if a long upload delay occurs and the channel fills up, the program discards the corresponding data at intervals of a given duration.

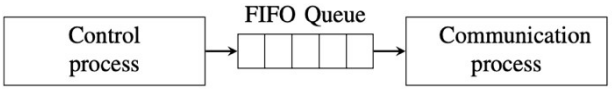


Fig. 2. States FIFO Queue.

On the other hand, the command channel starts from the communication process and goes to the control process. This design is presented in Figure 3. It has the capacity to store just a single instance of commands and any write action overwrites the stored instance. Thus, if eventually two different commands are received before the control reads them, the second one prevails as it is interpreted as a correction of the first one.

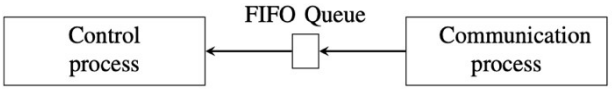


Fig. 3. Commands FIFO Queue.

It then associates a function to each of the two processors on the board. The control process function is associated to processor 1 of the board. It has an instance of the turbine control class and executes its methods directly. The implementation forces each iteration to last exactly 1 second. The communication process function is associated with processor 0 of the board. It has an instance of the communication class and executes its methods directly. The timing of this process is asynchronous to ensure that the downstream and upstream instructions are not corrupted. This logic does not compromise system security because it remains completely isolated from the control process.

The message passing between processes implements details to be reviewed. Sending to the status channel is performed in the control process at the rate of one send per second. If the channel is not saturated, it adds the element to the queue; otherwise, it empties it and discards a complete interval. The reception is performed in the communication process during the necessary iterations until an upload interval is completed. If the channel is empty, the process waits indefinitely for a status. Sending to the command channel is performed in the communication process, in the same loop in which states are received, but after at least 1 second since the previous send. It does not matter if the channel is empty or full, the command is always overwritten over the only position in the queue. The reception is performed in the control process every second if the queue is not empty. If the channel is empty, the process ignores this part.

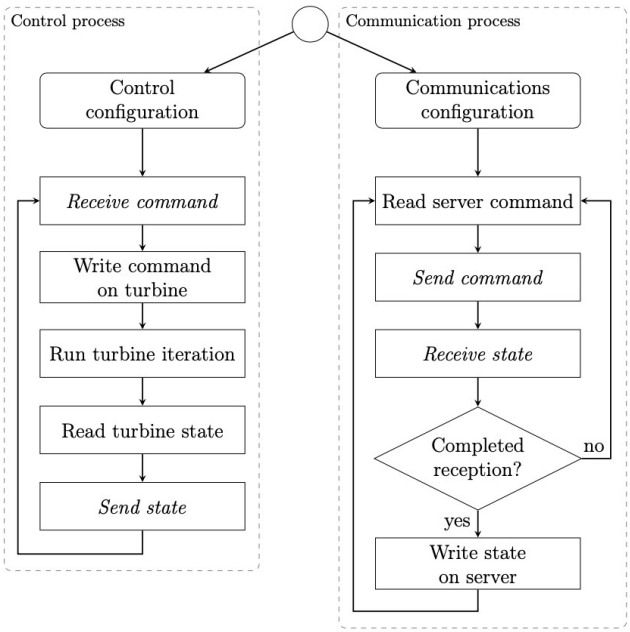


Fig. 4. Flux diagram.

## 4. Graphic User Interface implementation

The objective is to implement an interactive application that allows to display a large amount of information about the status of various systems and to send different commands, establishing bidirectional communications. In addition, scalability and adaptability to new types of turbines with different characteristics must be taken into account.

Given the complex context of this application, a solution based on object-oriented programming (OOP) and the Model-View-Controller (MVC) design pattern is proposed. This pattern is used to separate the user interface from the event management and communications, while isolating these from the business logic, i.e., from the processing and functionality of the system data [7]. This modular implementation facilitates the modification or replacement of any component with no need to make changes to the others, which also allows a simple extension of the system, either by adding new views or new functionalities to the model independently.

The model encapsulates information and functionality, notifying the listening components when you make changes. It is independent of the user interface, so it has no instances of the view classes or the controller. Its main function is to store information about the farm, such as name and description. In addition, it manages a set of instances representing all the turbines in the farm and monitors the total power generated by the active turbines at any given time. The turbine instances are responsible for accessing ThingSpeak and, secondarily, a weather server.

The view takes the information from the model and displays it to the user via a graphical interface. It caters to user interactions and invokes controller methods. For the implementation of the graphical user interface (GUI), App Designer, a MATLAB extension that allows creating graphical interfaces interactively, has been used.

The controller manages the user's interactions with the view and translates them into requests to the model, when necessary. It also performs periodic updates to the data without the need for the user to explicitly request them, maintaining dynamic behavior between the different views.

Fig. 5. A layered view of the turbine farm and turbine interfaces.

## 5. Use cases

Let's examine system automatic and manual modes behavior through the following use cases.

### *A. Auto mode*

1) *System configuration*: Immediately after powering the system, the blades are placed in a stop position with a pitch angle of 0º while waiting for the control and communication threads components to be configured and loaded. These include all peripherical sensors and actuators, the integrated display, the Internet connection and the ThingSpeak protocols. A few milliseconds after switching on the power, the autonomous control process will have started with the strategy implemented. Thanks to this speed and its initial stop state, the turbine is not at risk from loss of control at the start of the cycle. The communication process will start a few seconds later, as the connection configuration involves longer waiting times.

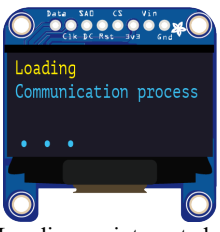

Fig. 6. Loading on integrated display.

2) *Slow period*: At this moment, the wind flow still has zero speed. The autonomous control loop starts by setting the pitch of the blades at 45º and the load indicator at 0 —with 500Ω resistance between the generator terminals— to favor the start-up. This first thrust is necessary to overcome the mechanical friction of the components. The communication process starts receiving data from the control and sending data to the dataserver. ThingSpeak receives the first data packets. The integrated display now shows the data that the communication process has received from control as in Figure 7, updated with 1Hz frequency.

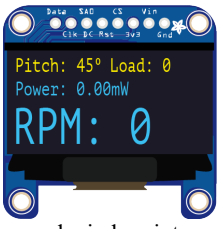

Fig. 7. Zero speed wind on integrated display.

3) *Starter period*: As the wind speed is set to low, the blades begin to rotate. At this point, as the revolutions increase, the pitch controller brings the blade angle closer to the optimum —set at 10º— and the load controller increases the indicator that reduces the electrical resistance. This control ensures that, even at the lowest wind speed, the turbine reaches the production angular velocity.
4) *Production period*: The RPM has reached the production threshold, set at 20% of the nominal RPM. The algorithm sets the load and activates the PID pitch control to keep the RPM within this threshold. The pitch control manages to stabilize them with an impressive error of less than 3% despite small variations in the wind flow generator. See first picture of Figure 8 for the nominal working state. As the wind speed increases to moderate, the revolutions increase slowly. The pitch PID controller distances the angle of attack from the optimum angle and brings it closer to a more inefficient position of 45º. The revolutions slightly decrease and the system, with its new pitch angle, manages to stabilize at the production threshold. The second picture of Figure 8 shows how the pitch has had to increase the angle of attack to maintain the prototype in the desired range of RPM.

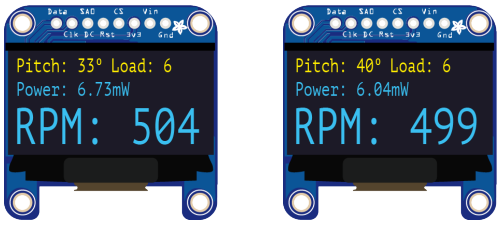

Fig. 8. Low and moderate wind speed on integrated display.

5) *Emergency stop*: Finally, the wind speed is increased to high. Since the pitch control had already reduced its range of action by stabilizing at the moderate speed, the revolutions increase rapidly and eventually, a peak may exceed the desired range. The system deactivates the pitch controller and activates the generator's electrical load controller as the last resource. Although this method is able to stop the increase in RPM, it is not enough to slow it down and bring it back into the working range. At this point, the strategy is to perform an emergency stop: flag the blades and increase the load resistance as much as possible. This strategy causes to completely stop the rotation of the generator without compromising the mechanism or structure of the turbine. The display shows an alert to indicate the stop. To illustrate this phase, the first line of the display blinks with the emergency message. This effect, and the fast reduction in RPM, can be seen in both pictures of Figure 9.

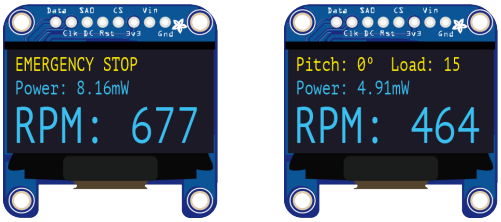


Fig. 9. Emegency stop progression on integrated display.

## B. *Manual mode*

1) *Load modification*: Increasing the load indicator may slightly reduce the revolutions. Nevertheless, it is interesting if you want to increase the power output of the turbine generator. The higher the indicator, the higher the power output. Figure 10 shows the result.

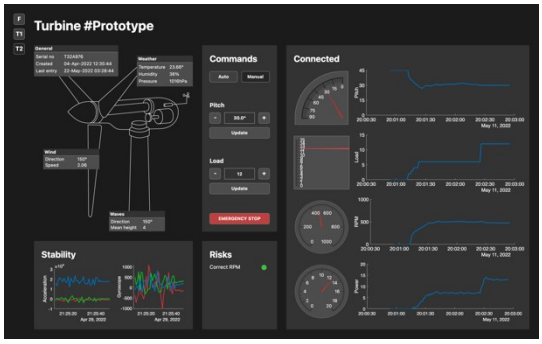


Fig. 10. Load change on interface.

2) *Pitch modification*: Varying the pitch angle determines changes in RPM much more significantly. Setting the pitch at the optimum angle can increase the RPM by more than 200% and the pitch at the minimum angle stops the operation. After an stability period, Figure 11 shows the result of setting the pitch at its optimum angle.

3) *Emergency stop*: When the RPM have gone out of the normal range, performing load or pitch controls may pose electrical or structural risks to the system. See the evolution of RPM and power decrease in Figure 11, after the RPM peak caused by the previous command.

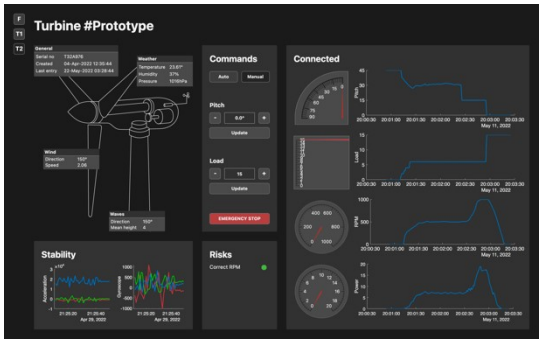


Fig. 11. Pitch change and emergency stop on interface.

## 6. Conclusions

This work has addressed the problem of generating a software architecture capable of autonomously directing the control strategy of a wind turbine prototype, displaying its status, and sending commands based on the behaviour observed in real time. This represents a step towards more accessible, efficient, and reliable offshore wind energy systems, with potential for wide-ranging applications in the renewable energy sector.

Traditional systems often rely on programmable logic controllers (PLCs) or high-end embedded platforms with greater computational resources and deterministic real-time behaviour. In contrast, low-cost microcontrollers like the ESP32 are enough for many control and monitoring tasks [8], especially in early developments. The modularity of the proposed architecture facilitates rapid prototyping and iterative improvements. Although the interface is developed in MATLAB, it could also be integrated with tools like LabVIEW for real-time visualization in both research and industrial applications.

Some highlights of the implementation include the adoption of a microcontroller with multiple processors to support a multi-threaded program capable of executing autonomous control and handling bidirectional communications simultaneously. This architecture is rarely used in low-cost, small-scale prototype IoT systems. However, certain limitations should be addressed. The current prototype may need further adaptation to fully represent offshore floating turbine dynamics. It would also be beneficial to enhance accelerometric measurements through mathematical transformations such as Fast Fourier Transform or Kalman Filters. Also, the system's reliance on ThingSpeak's free tier imposes data transmission limits.

Possible applications and future developments include achieving a system capable of supporting various control algorithms. This would allow for testing advanced control algorithms and machine learning techniques to optimize turbine performance [9].

## Acknowledgement

This work has been partially funded by the Spanish Ministry of Science and Innovation, under the MCI/AEI/FEDER project number PID2021-123543OBC21.

## References

[1] «Offshore renewable energy,» European Comission. Accessed on: Feb. 12, 2025. [Online]. Available: https://energy.ec.europa.eu/topics/renewable-energy/offshore-renewable-energy_en.

[2] T. Salic, J. Charpentier, M. Benbouzid, and M. Boulluec, «Control Strategies for Floating Offshore Wind Turbine: Challenges and Trends,» Electronics, vol. 8, 10 2019.

[3] S. Reed, «A New Weapon Against Climate Change May Float,» The New York Times, no. 4, para. 9, June 2020. Accessed on: Feb. 12, 2025. [Online]. Available: https://www.nytimes.com/2020/06/04/climate/floating-windmills-fight-climate-change.html.

[4] M. Tomas-Rodriguez, M. Santos, «Modelado y control de turbinas eólicas marinas flotantes,» Revista Iberoamericana de Automática e Informática industrial, vol. 16, p. 381, 09 2019.

[5] Dutton,Alastair Simon Piers; Sullivan,Charlene Coyukiat; Minchew,Elizabeth Oakes; Knight,Oliver; Whittaker,Sean, "Going Global: Expanding Offshore Wind To Emerging Markets," World Bank Group, Washington D.C., October 2019. Accessed on: Feb. 12, 2025. [Online]. Available: http://documents.worldbank.org/curated/en/716891572457609829/Going-Global-Expanding-Offshore-Wind-To-Emerging-Markets.

[6] Espressif Systems, "ESP32-WROOM-32", 2021. Accessed on: Feb. 12, 2025. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.

[7] J. Correas, «El patrón Modelo-Vista-Controlador,» in *Technology of Programming*. Department of Computer Systems and Computation, Complutense University of Madrid. 2018.

[8] K. Gunasekaran, N. Anbuselvan, J. S. Priya and C. Ravindra Murthy, "IoT based Wind Energy System Monitoring for Maximum Power Generation," 2023 8th International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2023, pp. 396-400.

[9] Sierra-García, Jesús Enrique, and Matilde Santos. "Exploring reward strategies for wind turbine pitch control by reinforcement learning." Applied Sciences 10, no. 21 (2020): 7462.