Automatic Fish Feeder using ESP32

CORNEA HOREA – IONUT

Table of Contents

1. Introduction

The purpose of this project is to create an automated fish feeder using an ESP32 microcontroller, enabling the user to remotely control and monitor feeding schedules via a web interface. The motivation behind this project is to provide a practical solution for aquarium owners who want to ensure consistent feeding times, especially during their absence.

Objectives

The primary scope of this project includes:

- Enabling both manual and scheduled feeding.
- Displaying feeding history for tracking purposes.
- A user-friendly interface accessible through a Wi-Fi connection.

This project stems from a well-known concept, but the customization options, such as scheduling via a web interface and a visual feed history, set it apart from existing solutions which only allow you to schedule the feeding time from the machine itself.

2. Bibliographic research

The concept of an automatic fish feeder has been implemented in various forms, both commercially and in DIY communities. Below are some examples for commercial fish feeder alternatives:

Commercial Solutions

Eheim Everyday Fish Feeder: This is a widely used automatic fish feeder that dispenses dry food at pre-set times. (*Figure 1*)

- **Features:** LCD display for schedule programming, battery-powered, reliable mechanism.
- **Limitations:** No network connectivity or remote control; the schedule can only be adjusted manually.
- **Relevance:** The physical feeding mechanism inspired the servo motor-based rotation design for dispensing food in the ESP32-based solution.

Figure 1 - Eheim EveryDay Fish Feeder

Reference number : 7

Fish Mate F14 Feeder: Specially designed for multiple daily feedings, supporting up to 14 meal portions. (*Figure* 2)

- **Features:** Easy-to-program timer, powered by AA batteries, works well for vacation use.
- **Limitations:** Only supports specific portion sizes; lacks customizable feeding intervals.
- **Relevance:** Demonstrates the importance of a flexible feeding schedule and variable feeding amounts, which were considered in this project as potential improvements.

Figure 2 - Fish Mate F14

Reference number: 8

DIY Arduino-Based Solution (*Figure 3*)

- Many DIY projects using Arduino offer cost-effective solutions for automatic fish feeding.
- Typical setups include a servo motor for dispensing food and an RTC (Real-Time Clock) module to manage schedules.
- Popular community projects include tutorials with basic features like a push-button for manual feeding and LCD screens to display the time and feeding status.

- **Key advantages:** Flexibility to implement additional features, lower cost compared to commercial feeders.

- **Challenges:** Require significant manual configuration, limited user interfaces, and sometimes rely on external libraries for NTP time synchronization.

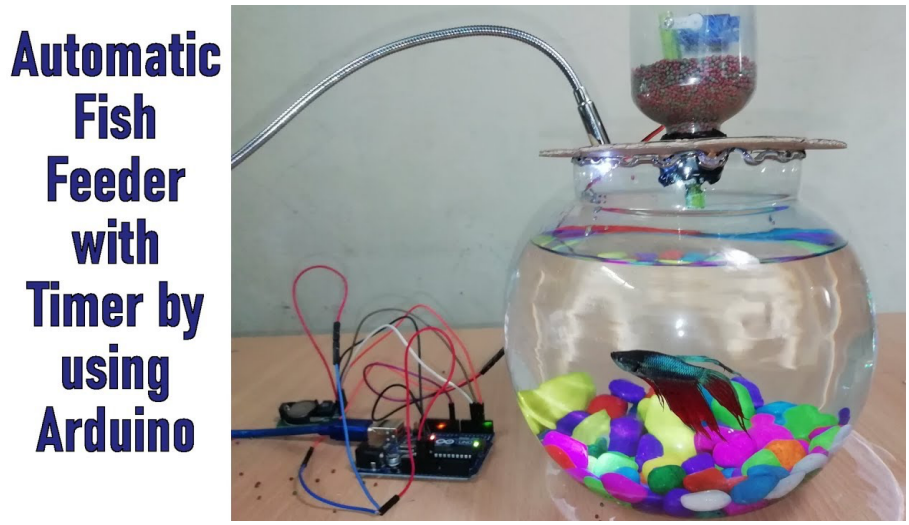

Figure 3 - Arduino based Fish feeder

Reference number: 9

Criteria	***Eheim Everyday Feeder***	***Fish Mate F14***	***DIY Arduino Project***	***ESP32 Fish Feeder (Proposed)***
Power Source	*Battery*	*Battery*	*USB/Battery*	*USB or external power/Battery*
Remote Control	*No*	*No*	*Limited with add-ons*	*Yes, via Wi-Fi*
Web Interface	*No*	*No*	*No*	*Yes*
Scheduling	*Fixed intervals*	*Fixed intervals*	*Customizable*	*Fully customizable*

Cost	*High (~€30-50)*	*Moderate (~€25)*	*Low (~€10-20)*	*Low (~€10-20 for components)*
Ease of Use	*Easy*	*Easy*	*Moderate (programming required)*	*Moderate (programming required)*

Key Takeaways for the Proposed Solution

The comparison shows that the proposed ESP32 fish feeder combines the best aspects of commercial feeders (ease of use and reliability) with the flexibility of DIY solutions. By using a web interface, the project eliminates the need for physical programming and offers a more intuitive experience. Additionally, incorporating NTP synchronization ensures that the feeding schedule remains accurate without the need for external RTC modules or frequent reprogramming.

3. Proposed solution and Implementation

Overall Description

The proposed solution is a device based on the ESP32 microcontroller, which combines hardware and software to create an automated fish feeder. The solution allows users to set an automatic feeding schedule or trigger manual feeding via a responsive web interface. Additionally, the feeder keeps a log of the last five feed events, ensuring transparency and accountability for the feeding process.

The device operates as follows:

- *The ESP32 microcontroller connects to a local Wi-Fi network and hosts a web server accessible from any device connected to the same network.*
- *The feeder synchronizes with an NTP (Network Time Protocol) server to maintain accurate time.*
- *A servo motor rotates to dispense food when a feeding event (manual or scheduled) is triggered.*
- *Users can view and update the feeding schedule and monitor the last five feed events via a user-friendly web interface.*

Theoretical Description of the System

The system's core functionality relies on the following:

- ***Time Synchronization:*** *The ESP32 fetches the current time from the pool.ntp.org server to ensure accurate scheduling. The Romanian time zone is configured using an offset.*

- ***HTTP Web Server:*** *The ESP32 hosts an HTTP server to serve the main control webpage and API endpoints (/feed, /setschedule, /getstate).*

- ***Servo Control:*** *The servo motor receives pulse width signals from the ESP32 to control its rotation angle, with a full 180-degree rotation to dispense food and a return to 0 degrees after feeding.*

Hardware Setup

Components Used:

- **ESP32 Development Board:** The main microcontroller responsible for Wi-Fi connectivity and running the web server.
- **Servo Motor:** A small servo motor (SG90) controls the release of food by rotating a predefined angle.
- **Power Supply:** The ESP32 and servo motor are powered via USB (5V) or an external power adapter.
- **Breadboard Wires:** Used to connect the components securely.

Pin Connections:

- **ESP32 GPIO2:** Connected to the servo's control pin.
- **5V :**Powers the servo motor.
- **GND:** Shared ground for the ESP32 and servo motor

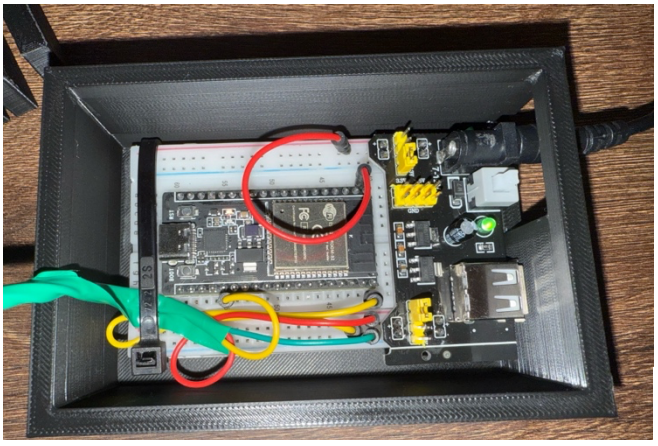

Figure 5 - actual Circuit Connection

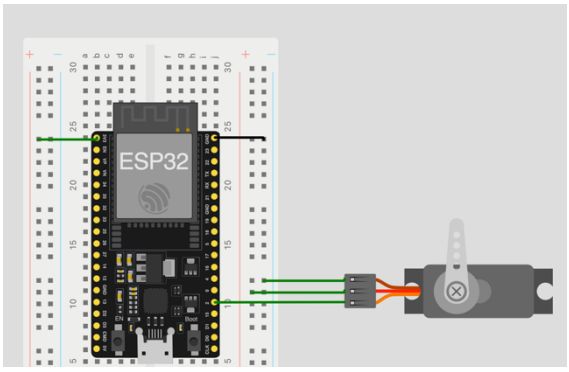

Figure 4 - Circuit Connection using Wokwi

Reference number: 10

Hardware Customization

To enhance the robustness and aesthetic appeal of the project, custom 3D-printed components were created:

- **ESP32 Casing:** A protective case was designed to encase the ESP32 microcontroller, preventing dust accumulation and minimizing the risk of accidental damage to the pins and connections.

Figure 6 - 3D printed ESP32 Casing

- **Servo Enclosure:** The servo motor was housed in a 3D-printed enclosure to secure its position and prevent movement during operation. This ensures precise and repeatable rotations without misalignment and also fixes the servomotor to the aquarium glass.

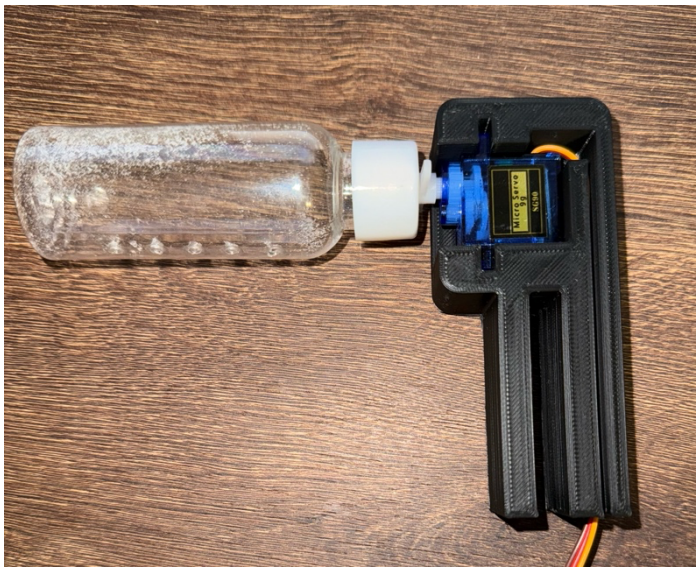

Figure 7 - 3D printed Servo Enclosure

Software Implementation

The software implementation consists of several key components:

1. **Web Interface (HTML/JavaScript):**

The ESP32 serves a single-page web application using HTML and JavaScript. This interface provides:

- **Manual Feed Button:** Allows the user to trigger a manual feeding event.
- **Schedule Form:** Enables the user to set the day, hour, and minute for automatic feeding.
- **Current Schedule Display:** Shows the current feeding schedule in a readable format.
- **Feed History Section:** Displays the last five feed events (both manual and automatic).

Web Page Features:

- Built with **Bootstrap 5** (4) for responsive design and a clean UI.
- JavaScript functions (updateUI(), feedNow(), setSchedule()) handle asynchronous updates via HTTP requests to the ESP32 endpoints.

2. **ESP32 Code Structure:**
   - setup() **Function:** Initializes the Wi-Fi connection, configures the NTP time, and sets up the web server routes.
   - loop() **Function:** Continuously listens for client requests and checks the feeding schedule.
   - handleRoot() **Function:** Serves the main webpage.
   - handleFeed() **Function:** Handles manual feeding requests by triggering the servo motor and logging the event.
   - handleSetSchedule() **Function:** Receives the day, hour, and minute parameters from the web interface and updates the schedule.
   - handleGetState() **Function:** Returns the current schedule and feed history in JSON format.
3. **Servo Motor Control:**
   - The servo motor is controlled using pulse width modulation (PWM) signals to achieve precise rotation.
   - The angle of rotation is set to 180 degrees during feeding, simulating a complete "dump" of fish food.
   - After a 1-second delay, the servo returns to its original position.
4. **Time-Based Feeding Check (**checkFeedingSchedule()**):**
   - The function compares the current time to the stored schedule.
   - If the current day, hour, and minute match the scheduled values, the servo is triggered for automatic feeding.
   - A safeguard (hasFedThisWeek) prevents multiple feedings within the same time window.

5. **Preferences Library Integration for Persistent Schedule Storage**

   To ensure that the feeding schedule persists even after power loss, the Preferences library (2) provided by ESP-IDF was integrated. This library allows data to be stored in non-volatile storage (NVS) and retrieved upon reboot.

   - **Benefits:**
     - Persistent storage ensures that the feeding schedule remains saved after a power outage.
     - No need for user reconfiguration after the system reboots.
     - The Preferences library offers efficient reads and writes, preserving flash memory lifespan.

Debugging and Improvements

During the development and testing phase, the following debugging and improvements were made:

- **Wi-Fi Connection Issues:** The initial connection process sometimes failed due to signal strength or incorrect credentials. This was mitigated by implementing a 30-second retry loop and displaying status messages.
- **Servo Motor Calibration:** The servo's pulse width was adjusted to ensure it dispensed an appropriate amount of food without over-rotating.
- **Time Synchronization:** The NTP configuration was tested for delays and fallback mechanisms. The system was adjusted to avoid unnecessary delays by reducing polling intervals for the NTP server.
- **Persistent Schedule Storage:** A critical improvement was the addition of the Preferences library for storing feeding schedules. Before this upgrade, the system would lose the feeding schedule on reboot. The Preferences library resolved this issue, ensuring that data is saved in NVS and retrieved automatically when the system restarts.

4. Testing and Validation

Problems Encountered and Solutions

During the implementation of the fish feeder project, several issues were encountered, requiring debugging and adaptations to ensure proper functionality:

1. **Wi-Fi Connectivity Issues:**
   - **Problem:** The ESP32 failed to connect to the Wi-Fi network in some cases due to weak signal strength.
   - **Solution:** A retry mechanism was added to attempt reconnections up to 30 times before displaying a failure message. Additionally, tests were conducted closer to the router to confirm that poor signal strength was the cause.

2. **Servo Motor Over-Rotation:**
   - **Problem:** The servo motor initially rotated past 180 degrees, causing the food dispenser to jam.
   - **Solution:** Adjusted the PWM signal range and recalibrated the servo using the ESP32Servo library to ensure it rotated exactly 180 degrees.
3. **Time Synchronization Delay:**
   - **Problem:** Delays in NTP synchronization caused the system to take up to 15 seconds to fetch the correct time.
   - **Solution:** The retry interval was reduced, and error messages were added for better debugging. A fallback to the last saved time was considered but not implemented in this version.
4. **Web Page Responsiveness:**
   - **Problem:** The web page took too long to update after a feeding event.
   - **Solution:** Increased the polling frequency of the updateUI() JavaScript function from 15 seconds to 10 seconds to ensure near real-time updates.

Calibration and Environmental Changes

- **Location and Signal Strength Tests:**

Distance from Router (meters)	**Connection Stability (Success Rate)**	**Comments**
1	*100%*	*Excellent connection*
5	*95%*	*Stable but slight delay*
10	*70%*	*Periodic disconnections*

- **Web Interface Latency:**

Polling Interval (seconds)	***Average Page Update Time (seconds)***	***Comments***
15	*3.5*	*Slow responsiveness*
10	*2*	*Moderate responsiveness*
5	*1*	Excellent responsiveness, but increased load

Phases of Development

The project went through several phases, evolving from a simple servo control test to a fully integrated system:

1. **Phase 1: Servo Motor Control Test**
    - Initial tests involved rotating the servo manually via code without any schedule or web interface.
2. **Phase 2: Web Server Setup**
    - The web server was implemented to serve the control webpage.
    - Required optimization of the HTML structure.
3. **Phase 3: Time Synchronization and Scheduling**
    - The NTP server was added to enable accurate feeding schedules.
    - Debugging of time offsets was performed to ensure correct handling of Romanian time zones.
4. **Final Phase: Integration and Validation**
    - All components were integrated, and the system was tested under real conditions.
    - The servo motor was tested for consistency in food dispensing, and the web interface was refined for ease of use.

5. Conclusion

The primary goal of creating an automated fish feeder with manual and scheduled feeding capabilities was successfully achieved. The system provides an easy-to-use web interface for controlling feeding events, displays a history of the last five feedings, and ensures accurate time synchronization.

Adaptations and Tests Summary

- Implemented a retry mechanism for both Wi-Fi and NTP connections.
- Improved the web interface for faster updates and mobile responsiveness.

Practical Applications and Benefits

- The solution provides convenience for aquarium owners who travel or have inconsistent schedules.
- It is easy to set up and requires minimal maintenance.
- The web-based control eliminates the need for physical adjustments or external displays.

Possible Improvements

1. **Security Enhancements:**
   - Add password-protected access to prevent unauthorized use of the web interface.
2. **Backup Battery:**
   - Integrate a battery backup to maintain feeding schedules during power outages.
3. **Customizable Feeding Portions:**
   - Implement a mechanism to adjust the portion size by controlling the servo rotation duration.
4. **Mobile Notifications:**
   - Add push notifications to inform the user when a feeding event occurs or if there are connection issues.

6. References

1. *Wi-Fi Web Server Example for ESP32:* https://randomnerdtutorials.com/esp32-web-server/
2. *ESP32 Preferences Library:* https://docs.espressif.com/projects/arduino-esp32/en/latest/tutorials/preferences.html
3. *NTP Time Synchronization:* https://forum.arduino.cc/t/setting-rtc-using-ntp/887850
4. *Bootstrap 5 Library for Web Design:* https://getbootstrap.com/docs/5.0/getting-started/introduction/
5. *Debugging :* https://chatgpt.com
6. *3D objects modelling software:* https://www.autodesk.com/products/fusion-360/overview?term=1-YEAR&tab=subscription
7. *Eheim Everyday Fish Feeder:* https://eheim.com/en_GB/aquatics/accessories/feeding/autofeeder/autofeeder
8. *Fish Mate F14:* https://fishmate.co.uk/Fishmate-F14-Aquarium-Feeder?srsltid=AfmBOoqsTqgE-AE3dZwrEWB9UNvqf_T_FYh6dx5NFIvvxM1fHkxsHdFN
9. *Automatic Fish Feeder Mechanism with Timer by using Arduino:* https://www.youtube.com/watch?v=nCuCHAS6Evk
10. *Wokwi ESP32 simulator :* https://wokwi.com