Synchronized Data Acquisition System Report

Project: Renewable Energy from Ocean Waves (Capstone #30)

Role: Electronics & Control Leader

Author: Matthieu Gomez

Date: December 12, 2025

Table of Contents

1. Introduction & System Architecture

1.1 Context

The objective of the Point Absorber project is to design and test a small-scale wave energy converter. To validate the efficiency of the system and compare it with theoretical models, precise data acquisition is required. We need to measure the mechanical input (buoy motion) and the electrical output (generator power) simultaneously.

1.2 The Synchronization Challenge

A major challenge in such systems is **data synchronization**.

- Using two separate loggers (one for the camera, one for the circuit) results in time skews, making efficiency calculations $\frac{P_{elec}}{P_{Mech}}$) inaccurate.
- Windows OS latency makes PC-based timestamping unreliable for high-frequency electrical measurements.

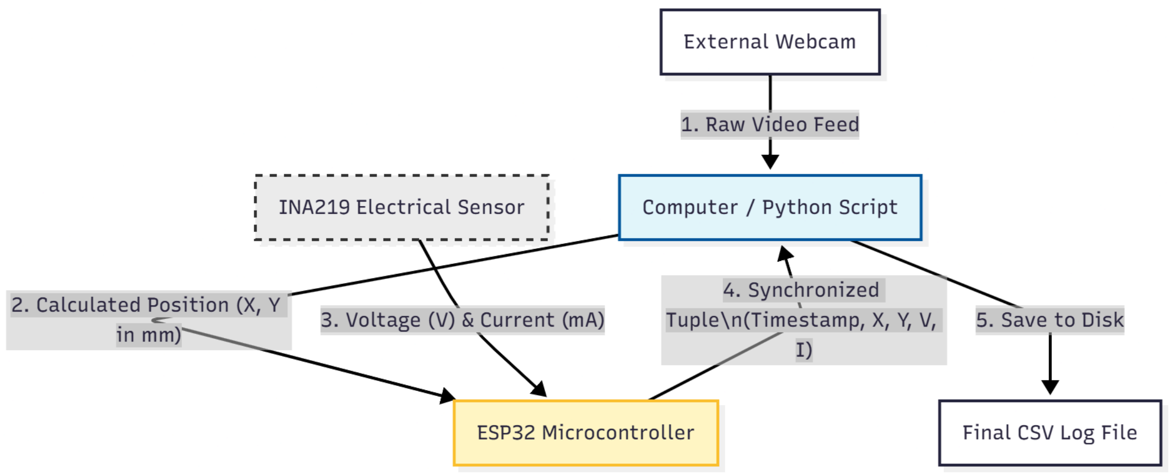
1.3 Solution: The "Synchronized Loop" Architecture

We developed a closed-loop architecture where the Microcontroller (ESP32) acts as the **Time Master**.

1. The PC processes images and sends the position ($X, Y$) to the ESP32.
2. The ESP32 waits for this position data.
3. Upon reception, the ESP32 *immediately* reads the electrical sensors and applies its own hardware timestamp.
4. The ESP32 sends the full synchronized tuple back to the PC for logging.

1.4 System Diagram

The following diagram illustrates the data flow and connections:



2. Mechanical Measurement Methodology (Computer Vision)

To measure the buoy's displacement without adding mechanical friction (which would damp the system), we opted for a non-contact Computer Vision approach using **ArUco markers**.

2.1 The 3-Marker System

Instead of a single marker, we implemented a robust **3-marker system** to handle camera vibrations and setup variations.

- **Reference Marker 1 (ID 10):** Acts as the physical Origin $(0,0)$.
- **Reference Marker 2 (ID 11):** Placed at a fixed, measured distance from Ref 1. Used for **Dynamic Calibration**.
- **Mobile Marker (ID 12):** Placed on the moving part (spring/buoy).

2.2 Dynamic Calibration Algorithm

Hardcoding a "pixel-to-mm" ratio is error-prone (if the camera moves slightly, data is invalid). Our algorithm recalculates the scale factor for *every single frame*:

$$Ratio\ (mm/px) = \frac{Known\ Distance\ (mm)}{Pixel\ Distance(Ref_1, Ref_2)}$$

This ensures that even if the camera vibrates or focus changes, the measurement remains accurate in millimeters.

2.3 Error Handling & Expected Precision

- **Robustness:** If the mobile marker is temporarily occluded or exits the frame, the system outputs a specific error value (-1.0) instead of freezing or logging false data. This allows for easy data cleaning during post-processing.
- **Precision:** With a standard HD webcam positioned at ~60-80cm from the setup, the expected measurement precision is approximately $\pm$ **0.5 mm to 1.0 mm**. This sub-millimeter accuracy is achieved through OpenCV's sub-pixel corner detection algorithms.

3. Electrical Measurement & Hardware Setup

3.1 Components

- **Microcontroller:** ESP32-DevKitC (Selected for its high processing speed and dual-core capabilities).
- **Sensor:** INA219 High-Side DC Current/Voltage sensor.
  - Communication: I2C (SDA: GPIO 21, SCL: GPIO 22).
  - Max Voltage: 26V (High enough for our generator).
  - Max Current: 3.2A.

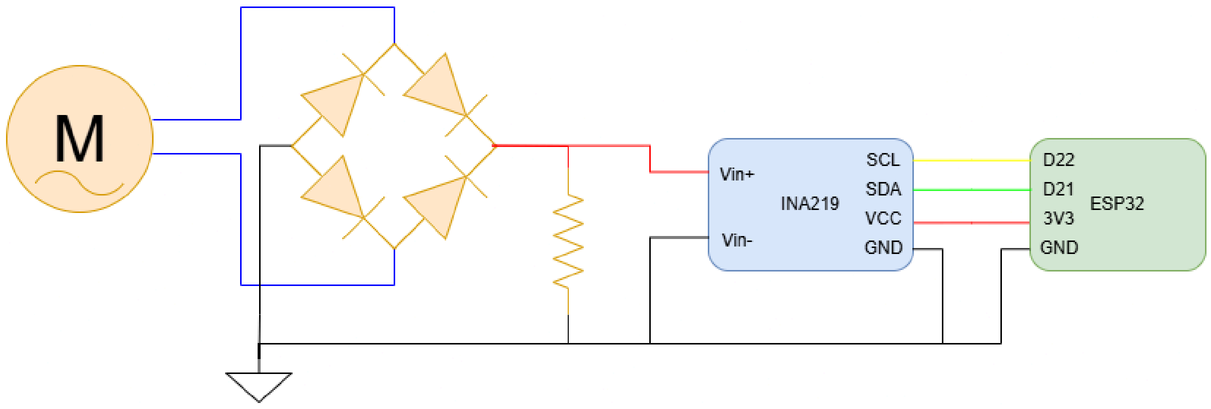
3.2 Wiring Diagram

The electrical circuit ensures a common ground between the power stage (generator) and the logic stage (ESP32).

- **VCC/GND:** Powered via ESP32 3.3V rail.

- **SDA/SCL:** Connected to GPIO 21/22 with internal pull-ups.
- **Common Ground:** The generator's rectifier output (-) is linked to the ESP32 GND to provide a stable reference voltage.

Detailed Wiring:



4. Software Implementation

4.1 Python Script (calibrated_logger.py)

The script handles the heavy lifting of image processing.

- **Initialization:** Opens the Serial port with rtscts=False and waits 4 seconds to allow the ESP32 to reboot and stabilize.
- **Buffer Cleaning:** A critical routine clears the serial input buffer to remove boot logs (ets Jul...) before starting acquisition.
- **Loop:**
  1. Detects markers.
  2. Computes the dynamic ratio.
  3. Sends coordinates formatted as "X.XX,Y.YY\r\n" to the serial port.
  4. Listens for the echo (Full Log) from the ESP32 and writes it to CSV.

4.2 ESP32 Firmware (master_data_logger.ino)

The firmware is optimized for speed (115200 baud) and robustness.

- **Parsing:** Uses Serial.parseFloat() to efficiently extract X and Y values from the incoming string.
- **Sensor Read:** Calls ina219.getBusVoltage_V() and ina219.getCurrent_mA().
- **Formatting:** Assembles a comma-separated string (Timestamp,X,Y,Volts,Amps) and terminates it with a newline character (println).

5. Operational Guide: How to Run an Experiment

5.1 Hardware Setup

1. **Placement:** Fix the camera facing the setup. Ensure Markers 10, 11 (Fixed) and 12 (Mobile) are visible.
2. **Calibration:** Measure the physical distance between the centers of Marker 10 and 11. Update KNOWN_DISTANCE_MM in the Python script.
3. **Connection:** Connect the ESP32 to the PC via a high-quality **Data USB Cable**.

5.2 Software Launch Sequence (CRITICAL)

Due to COM port exclusivity, the launch order must be strictly followed:

1. **Plug in** the ESP32.
2. **Open Arduino IDE** to check the Port number (e.g., COM3).
3. **CLOSE ARDUINO IDE** completely (or at least close the Serial Monitor). If the Serial Monitor is open, Python will crash with PermissionError.
4. **Run the Python Script:**
   ```
   python calibrated_logger.py
   ```
5. **Wait:** The script will pause for 4 seconds ("Cleaning buffer...").
6. **Verify:** A video window will appear. Green squares should outline the markers. Data logs should scroll in the terminal.

5.3 Troubleshooting

Symptom	Probable Cause	Solution
PermissionError: Access is denied	The COM port is locked by another program.	Close Arduino IDE, PuTTY, or any other serial monitor.
"RX: -1.0, -1.0" in Logs	Markers are not detected.	Check lighting. Ensure all 3 markers (10, 11, 12) are in the frame.
No "PONG" / No Data Received	Serial driver issue or bad cable.	1. Use a Data USB cable. 2. Reinstall CP210x drivers. 3. Check Device Manager for COM port number.
Values stuck at 0.0V / 0.0mA	Wiring issue.	Check the 4 wires to INA219 (VCC, GND, SDA, SCL). Ensure the Load is connected.

5.4 Data Retrieval

- When finished, press **'q'** on the video window to stop.
- A file named log_experience_YYYYMMDD_HHMMSS.csv will be saved in the script folder.
- This file can be directly imported into MATLAB or Excel for analysis.

6. Future Improvements

- **Wireless Data Transmission:** Replace the USB Serial link with **Wi-Fi (MQTT or WebSocket)**. This would allow the PC to be placed further away from the water tank, reducing the risk of water damage.
- **Active Load Control:** Implement a PWM control loop on the ESP32 to automatically adjust the load resistance (simulating different PTO damping coefficients) based on the wave amplitude detected by the vision system.