Smart Wall Art – Interactive Art That Changes Based on Environmental Input

MohammadAmin Moghadasi
Department of Computer Science
University of Bologna
Bologna, Italy
Mohammad.moghadasi@studio.unibo.it

***Abstract*—The Smart Wall Art (SWA) project integrates art and technology to create interactive digital artworks that dynamically respond to environmental conditions. By using sensors such as DHT11, LDR, and PIR, the system collects real-time data on temperature, humidity, light, and motion, which are then processed through ESP32 and adaptively visualized on a display. Data are transmitted via MQTT or HTTP to an InfluxDB database for storage, analysis, and visualization, while machine learning models improve predictive capabilities. User engagement is further incorporated through a Telegram bot, applying an epsilon-greedy strategy to adapt visual outputs based on feedback. This fusion of IoT, data analytics and interactive art enhances Human–Computer Interaction (HCI) and Human–Machine Interfaces (HMI), providing a personalized and immersive artistic experience.**

***Index Terms*—Smart Wall Art, MQTT, HTTP, ESP32, DHT11, LDR, PIR, InfluxDB, Decision Tree Regressor, Epsilon-Greedy Algorithm.**

I. Introduction

The rapid growth of the Internet of Things (IoT) has enabled the integration of sensing, data processing, and communication technologies into everyday applications, extending beyond industrial and scientific domains into creative and artistic fields. The Smart Wall Art project explores this intersection by developing an interactive digital artwork that adapts in real time to environmental conditions. Using sensors such as DHT11 for temperature and humidity, an LDR for light intensity, and a PIR for motion detection, the system continuously collects contextual data from its surroundings. This information is transmitted via lightweight communication protocols, namely MQTT and HTTP, and stored in an InfluxDB database for analysis and visualization. To enhance interactivity, the project incorporates predictive analytics through machine learning techniques and integrates user feedback collected via a Telegram bot, which employs an epsilon-greedy strategy to dynamically refine the visual outputs. By merging IoT technology, data analytics, and user-centered design, Smart Wall Art demonstrates how human-computer interaction (HCI) and human-machine interfaces (HMI) can evolve into immersive, personalized artistic experiences.

II. System Architecture

The system architecture of the Smart Wall Art project is designed to integrate sensing, processing, communication, and actuation into a unified framework that enables interactive and adaptive digital artwork. At its core, an ESP32 microcontroller collects environmental data from multiple sensors: the DHT11 for temperature and humidity, the LDR for ambient light, and the PIR sensor for motion detection. This data is displayed locally on an OLED screen and used to drive an RGB LED, reflecting environmental changes in real time. For connectivity, the system supports dual communication modes—MQTT for lightweight, real-time messaging and HTTP for request–response operations—ensuring flexible integration with cloud services. Sensor data is transmitted to a data proxy application, which stores and manages it in an InfluxDB time-series database, enabling visualization and further analytics. Runtime commands sent over MQTT allow for dynamic reconfiguration, such as adjusting sampling rates or motion thresholds, ensuring robustness and adaptability. This layered architecture establishes the foundation for interactive art by linking the physical environment with digital representation in a responsive and scalable manner. Figure 1 illustrates the overall project architecture, which encompasses both the hardware and software components.

III. Hardware Design

The hardware design of the Smart Wall Art system is centered on the ESP32 microcontroller, which manages data acquisition, processing, and communication. Three primary sensors are integrated into the setup: the DHT11 sensor for measuring temperature and humidity, the PIR sensor for detecting motion, and the LDR module for monitoring ambient light intensity. Together, these sensors provide real-time environmental data that drives the adaptive behavior of the system. The readings are displayed locally on an OLED screen (SSD1306, I²C protocol), allowing immediate visualization of the collected parameters.

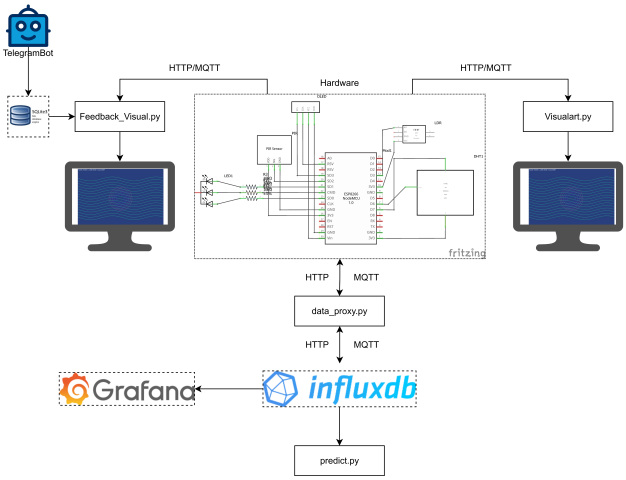

Fig. 1: System Architecture.

In addition to sensing, the system incorporates actuation through an RGB LED connected to GPIO 25, 26, and 27, which provides color-coded feedback based on environmental conditions such as light intensity. All components are powered by the ESP32's 3.3V output and connected on a breadboard for modularity and prototyping convenience. This configuration ensures reliable sensor readings, real-time responsiveness, and seamless integration between the physical environment and digital art representation

Equipment Name	Pinout	ESP32
OLED	VCC GND SCL SDA	3.3 GND IO22 IO21
DHT11	VCC GND DATA	3.3 GND IO04
PIR	VCC GND Out	3.3 GND IO15
LDR	VCC GND AO	3.3 GND IO34
RGB (LED)	GND R G B	GND IO27 IO26 IO25

TABLE I: Pin connections between equipment and ESP32

IV. DATA ACQUISITION

The data acquisition layer is responsible for sensing, display, and telemetry of environmental signals that drive the adaptive visuals of the Smart Wall Art system. An ESP32 microcontroller reads three sensors—*DHT11* (temperature, humidity), *LDR* (ambient light), and *PIR* (motion)—and publishes structured measurements in real

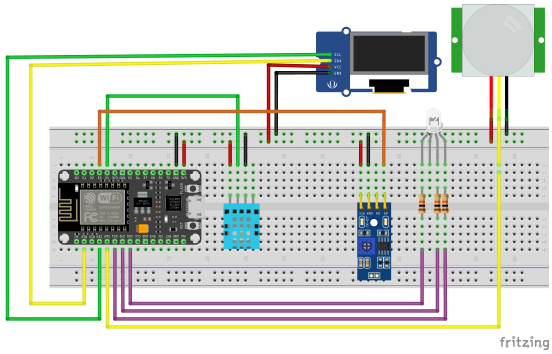

Fig. 2: Wiring Diagram.

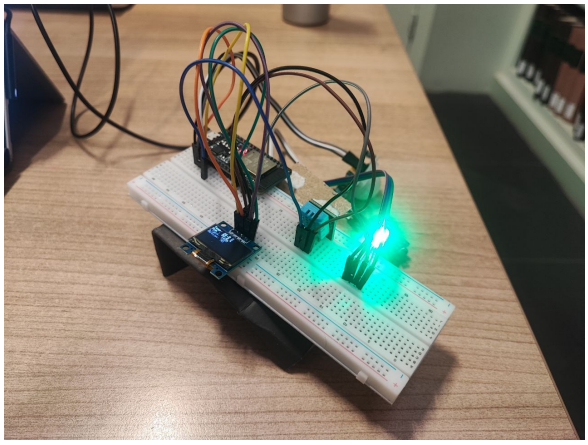

Fig. 3: Hardware Implementation.

time. Readings are shown locally on an *SSD1306 OLED* (I^2C), while an *RGB LED* provides immediate color-coded feedback.

A. Sensors, Actuators, and Local Display

- **DHT11** (GPIO 4): measures *temperature* and *humidity*.
- **LDR module** (ADC GPIO 34): measures *ambient light intensity*.
- **PIR motion sensor** (GPIO 15): detects *human presence/motion*.
- **OLED SSD1306** (I^2C: SDA GPIO 21, SCL GPIO 22): displays live values.
- **RGB LED** (PWM: R GPIO 27, G GPIO 26, B GPIO 25): instant visual feedback.

B. Acquisition Logic and Debouncing

To suppress false motion alarms, the firmware counts consecutive PIR `HIGH` samples and sets `motion=1` only when the count exceeds a configured threshold (*motion debouncing*). Light levels drive the RGB LED according to simple thresholds (example policy in Table II); these values can be tuned during experiments.

Condition	Light reading	LED color
Bright environment	> 3000	Red
Moderate light	> 1500	Green
Low light	≤ 1500	Blue

TABLE II: Example LED logic driven by LDR reading (tunable thresholds).

C. Telemetry and Runtime Control

At each sampling instant, the ESP32 builds a JSON payload

{`temp`, `hum`, `light`, `motion`, `device_id`, `ts_ms`}

and transmits it using a selectable transport:

- **MQTT** topic `smartart/sensordata` for lightweight, low-latency streaming.
- **HTTP** POST to `/ingest` for simple request–response delivery.

The device supports remote reconfiguration over MQTT:

- `smartart/cmd/sampling_rate` $\rightarrow$ sets sampling interval (s).
- `smartart/cmd/motion_alert` $\rightarrow$ sets PIR consecutive-hit threshold.
- `smartart/cmd/mode` $\rightarrow$ switches transport (`mqtt` or `http`).

D. Proxy and Storage Pipeline

On the server side, a lightweight Python *data proxy* bridges telemetry to an InfluxDB time-series database. The proxy:

1) Subscribes to `smartart/sensordata` (MQTT) and accepts `/ingest` (HTTP) JSON.
2) Validates presence of the four core fields `temp`, `hum`, `light`, `motion`.
3) Writes data to InfluxDB with `device_id` as a tag. If a valid timestamp in milliseconds ($\geq 1.6 \times 10^{12}$) is provided, it is converted to nanoseconds and stored as the event time; otherwise, the server's current time is used.

This architecture yields reliable, queryable streams for dashboards and downstream analytics (e.g., MAE/MSE evaluation, forecasting).

E. Message Schema (example)

```
{
"device_id":"esp32-smartart-01",
"ts_ms": 1736354123456,
"temp": 24.8, "hum": 48.2, "light":
2120, "motion": 1
}
```

V. DATA STORAGE

To ensure reliable collection, persistence, and analysis of environmental data, the *Smart Wall Art* system employs a cloud-based storage and visualization infrastructure. The sensor readings captured by the ESP32 (temperature, humidity, light, and motion) are packaged into JSON payloads and transmitted via MQTT or HTTP to the data proxy. From there, the data are stored in an **InfluxDB time-series database**, which is optimized for IoT telemetry due to its efficient handling of time-stamped data. Each record includes fields such as temperature, humidity, light, and motion, tagged with device identifiers,, and stored with nanosecond precision timestamps. This structure enables efficient querying, long-term storage, and time-based analysis of sensor signals.

The stored data is then visualized using **Grafana**, which provides an interactive dashboard for monitoring system behavior. Time-series graphs display trends for each environmental variable, allowing the identification of both patterns and anomalies. For example, light intensity graphs reveal variations throughout the day, humidity and temperature plots show gradual environmental changes, and motion graphs capture user presence events. Grafana's customizable dashboards also support threshold-based alerts and comparative visualizations, making the system not only a creative interactive art piece but also a functional monitoring solution.

VI. FORECASTING

The forecasting module of the *Smart Wall Art* project extends the system's functionality by predicting future sensor values using historical time-series data. This enables proactive adaptation of the interactive art system, anticipating environmental changes rather than only reacting to them.

A. Data Source

Sensor data is queried directly from the InfluxDB time-series database using Flux queries. The dataset consists of timestamped records of temperature, humidity, and light intensity collected over the past 24 hours. Records are preprocessed to remove missing values and aligned into a structured `Pandas` DataFrame for analysis.

B. Feature Engineering: Lagged Predictors

To capture temporal dependencies, lag features are added for each variable. With lag parameter $k = 2$, the system builds predictors using the previous two time steps of all three variables (temperature, humidity, and light). This approach incorporates both auto-lags (past values of the same variable) and cross-lags (past values of other variables), enriching the predictive power.

C. Modeling

For each target variable (temperature, humidity, light), a separate regression model is trained using the `DecisionTreeRegressor` from the `scikit-learn` library:

- Model depth limited to `max_depth=4` ensures a balance between accuracy and generalization.
- Data split chronologically into 80% training and 20% testing to preserve time ordering.
- The model learns nonlinear relationships between lagged inputs and current sensor values.

D. Evaluation and Results

Performance is assessed on the held-out 20% test window using **Mean Absolute Error (MAE)** and **Mean Squared Error (MSE)**. Results demonstrate the model's ability to approximate environmental dynamics within the constraints of limited model depth.

Visualization outputs include time-series plots comparing actual vs. predicted values:

- **Light Forecast**: Fluctuations are captured, though rapid variations introduce prediction gaps.
- **Temperature Forecast**: The model successfully tracks gradual environmental changes, with some smoothing of spikes.
- **Humidity Forecast**: Predictions align with general trends, though abrupt shifts are harder to capture.

All predictions are exported into a consolidated CSV file for further analysis and reproducibility.

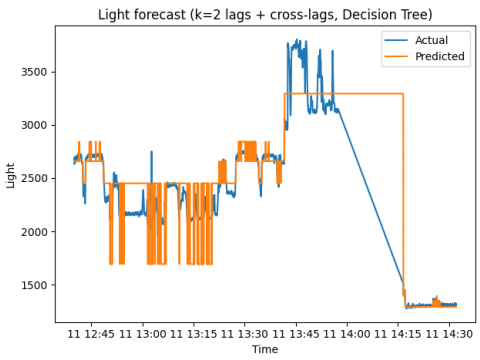

Fig. 4: Light forecast.

VII. SMART ART ACTUATOR

The Smart Art Actuator represents the final stage of the *Smart Wall Art* system, where raw sensor data is transformed into dynamic artistic visuals using a real-time rendering engine (implemented in Python with Pygame). This module closes the loop of sensing, processing, and visualization by adapting graphical elements to temperature, humidity, light intensity, and motion inputs.

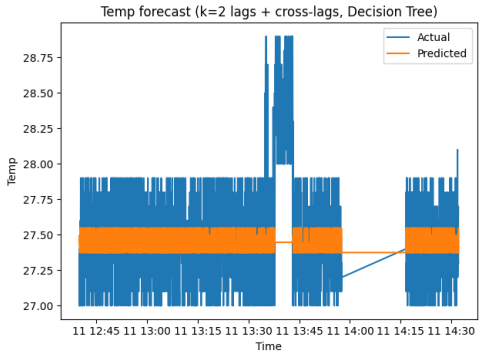

Fig. 5: Temperature forecast.

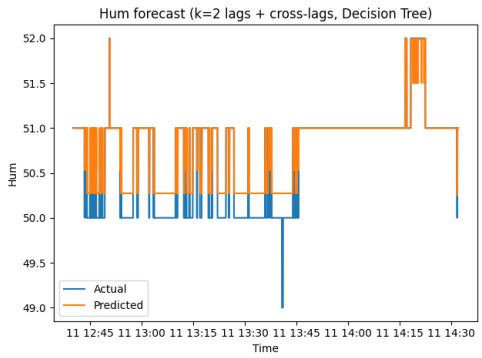

Fig. 6: Humidity forecast.

A. Temperature → Rings

Temperature values control the generation of **concentric rings** displayed at the center of the canvas. The number of rings is derived as:

$$\text{Ring Number} = 3 + \frac{\text{Temp} - 15}{5}$$

Each ring's radius and color evolve over time according to:

$$r_i = 40 + 26i + 12\sin\left(\frac{t}{30} + i \cdot 0.6\right)$$

$$\text{Ring Color}_i = \left(100 + 15i,\ 40 + 12i,\ 220 - 18i\right)$$

This ensures that higher temperatures produce more rings with larger radius and warmer tones, visually encoding environmental conditions.

B. Humidity → Waves

Humidity modulates **wave patterns** that flow across the screen. The amplitude of the sinusoidal waves increases with humidity:

$$\text{Wave Amplitude} = 8 + \frac{\text{Hum}}{3}$$

The motion of the waves over time is given by:

$$y(x,t) = A \cdot \sin\left(\frac{x+2t}{80}\right)$$

where A is the humidity-dependent amplitude. This produces smooth oscillations whose intensity reflects environmental moisture levels.

Ambient Light → Background

Ambient light levels influence the **background color intensity**, normalized to the ESP32's ADC resolution (12-bit):

$$\text{Normalized Light} = \frac{\text{Light}}{4095}$$

This normalized value is mapped to RGB shades, creating darker or brighter backgrounds according to the measured light.

Motion → Flash Effect

The PIR sensor detects motion events, which trigger a **flash overlay** lasting for 2.5 seconds. This is implemented as a semi-transparent white surface drawn over the canvas:

$$\text{Flash Duration} = 2500\,\text{ms}$$

The flash effect visually emphasizes user presence, reinforcing the interactive nature of the system.

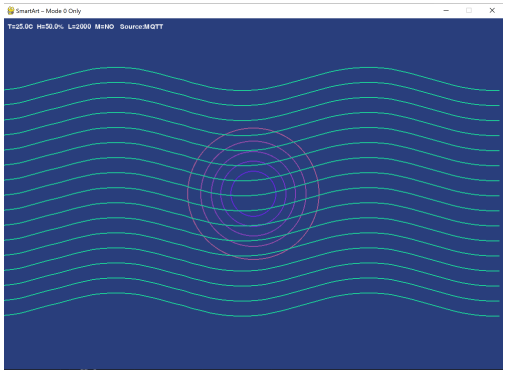

Fig. 7: Visual art.

C. Summary

By combining these mappings, the Smart Art Actuator translates raw environmental data into evolving digital art. Temperature governs rings, humidity controls waves, light adjusts the background, and motion triggers transient flashes. Together, these visual encodings create an immersive, real-time artistic experience that embodies the principles of Human–Computer Interaction (HCI) and Human–Machine Interfaces (HMI).

VIII. FEEDBACK AND USER ENGAGEMENTS

This module introduces a human-in-the-loop mechanism so that the system adapts not only to environmental signals but also to user preferences. A Telegram bot collects ratings and optional comments, which are stored in SQLite and consumed by a feedback engine that applies an ε-*greedy* policy to drive the visual palette.

A. Telegram Bot for Feedback Collection

The bot (built with `python-telegram-bot`) provides:

- **Ratings (0–5)** of the current visual style via inline buttons;
- **Optional short comments** ($\leq$200 chars) after a rating;
- **/myratings** to show a user's recent ratings;
- **Persistent storage** in SQLite (`feedback2.db`) with two tables:
  - `users(user_id, username, first_name, last_name, updated_at);`
  - `feedback(id, user_id, username, rating, comment, created_at).`

Schema migration helpers ensure columns exist across updates.

B. User Engagement Algorithm (ε-Greedy)

The `Feedback_Visual.py` process periodically reads the most recent N ratings and computes their average. A policy then selects how boldly to change the color palette:

1) **Read feedback:** every 2 s, fetch the last $N = 20$ ratings.
2) **Compute average:** $\overline{r} = \frac{1}{n}\sum_{i=1}^{n} r_i$ if ratings exist.
3) **Select ε:**

$$\varepsilon = \begin{cases} \varepsilon_{\text{NEUTRAL}} = 0.20, & \text{if no ratings exist;} \\ \varepsilon_{\text{EXPLORE}} = 0.50, & \text{if } \overline{r} < \text{THRESH} = 3.0; \\ \varepsilon_{\text{EXPLOIT}} = 0.10, & \text{if } \overline{r} \geq \text{THRESH.} \end{cases}$$

4) **Palette choice (ε-greedy):** with probability ε, *explore* a new palette from a predefined set; otherwise *keep* the current palette. A small jitter ± 12 is applied to each RGB channel to avoid repetition.
5) **Apply to visuals:** the selected palette colors (`bg`, `wave`, `circle`) are blended with environment-driven parameters from the actuator (rings/waves/background).

C. Integration with the Actuator

The engagement layer augments (does not replace) the actuator mapping: temperature still sets ring count, humidity sets wave amplitude, and ambient light drives background intensity. The feedback policy *modulates the*

color palette: if $\overline{r} < 3$ the system explores new palettes more often, whereas for $\overline{r} \geq 3$ it stabilizes around the successful palette with mild jitter.

D. Outcome

Combining Telegram-based feedback with an ε-greedy controller yields a hybrid adaptive art system that balances exploration of novel visual designs with exploitation of user-approved styles. This strengthens user engagement and advances the project's HCI/HMI goals.

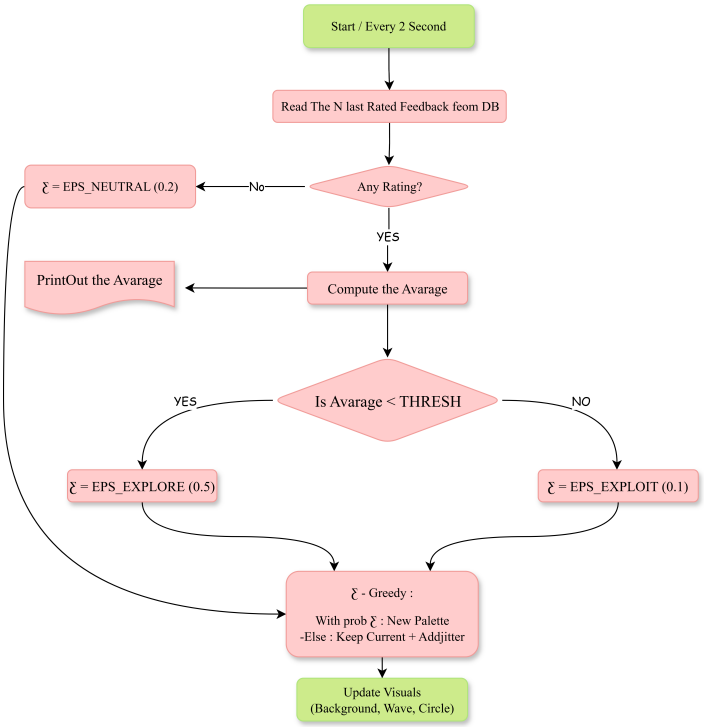

Fig. 8: Flowchart of the user engagement algorithm