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

Vittorio Rossetto
0001136868
Alma Mater Studiorum - University of Bologna
Bologna, Italy
vittorio.rossetto@studio.unibo.it

***Abstract*—The Smart Wall Art project seeks to merge art and technology to create dynamic, interactive experiences that adapt to the user's environment. This project is motivated by the desire to make art more engaging, interactive, and emotionally resonant, enabling it to actively respond to the daily rhythms of life. It also emphasizes creativity, exploring how IoT technology can enhance our emotional connection with visual experiences**

I. Introduction

The Smart Wall Art project aims to develop an interactive digital art system that responds to environmental conditions in real time. By combining microcontroller-based sensing, data transmission, time-series storage, and dynamic visual rendering, the project demonstrates how IoT technologies can be applied to create personalized and adaptive user experiences. The artwork displayed on screen changes based on ambient light, temperature and humidity, creating a system where the visual output is directly influenced by the physical environment.

This project aim is to explore the integration of environmental data with digital media. In addition to responding to sensor data, the system also supports optional user feedback through a Telegram bot, enabling further personalization based on trending preferences.

This paper presents the full implementation of the Smart Wall Art system. It details the hardware setup using an ESP32 microcontroller connected to motion, temperature, humidity, and light sensors. The microcontroller collects sensor data and transmits it using HTTP, with runtime configurability provided via MQTT. A Python-based data proxy receives the data and stores it in an InfluxDB time-series database, which can be visualized using Grafana. A separate Python application renders dynamic artwork on screen, adapting visual elements to current environmental conditions. Additionally, a forecasting module is included to predict future sensor readings using time-series analysis. Optional features include user feedback collection and visual style adaptation based on user ratings provided on the Telegram bot.

II. Projet's Architecture

The Smart Wall Art system is designed as a layered architecture that links physical sensors, a communication backbone, data storage, and visualization components. Its goal is to transform environmental conditions and visitor presence into interactive artwork while logging sensor data for later analysis in Grafana dashboards.

A. Hardware and Sensor Layer

At the foundation of the system lies the hardware responsible for capturing environmental and interaction data. This layer provides the raw input that drives both the visualization and monitoring components.

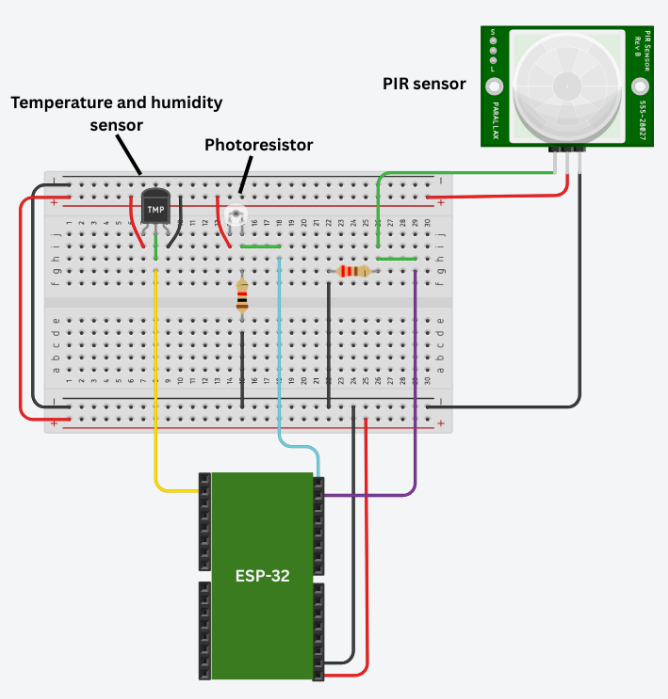

Fig. 1. Circuit design, to be linked with a computer to display live images

- **ESP32/ESP8266 microcontroller** provides WiFi connectivity and acts as the central node for sensor data collection.
- **PIR motion sensor** detects presence (digital HIGH/LOW).
- **DHT11 (KY-015)** measures temperature and humidity.
- **LDR (light sensor)** provides ambient light levels (analog 0–1023).

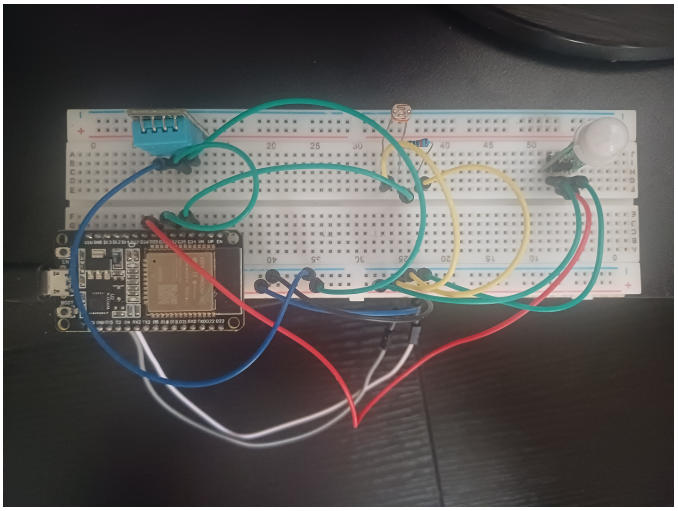

Fig. 2. Picture of the hardware

The microcontroller packages sensor readings into JSON messages and publishes them to the messaging layer.

B. Messaging Layer

This layer ensures that collected sensor data can be transmitted to multiple consumers in real time. By using a publish/subscribe protocol, it decouples the hardware from downstream applications.

- Implemented using an **MQTT broker** (Mosquitto).
- Topics:
  - `smartart/sensor` for environmental data (light, temperature, humidity).
  - `smartart/motion` for motion events.

C. Data Proxy Layer

The data proxy acts as a bridge between real-time messaging and long-term storage. It transforms MQTT messages into a consistent format suitable for database insertion.

- A python script subscribes to MQTT topics.
- Normalizes data and writes it to **InfluxDB**.
- Ensures consistent schema across measurements.

D. Storage Layer (InfluxDB)

This layer provides persistent, queryable storage for all sensor data. By leveraging a time-series database, the system can store and retrieve historical measurements.

- Stores time-stamped sensor data.
- Optimized for time-series queries.
- `sensor_data` only contains data with `motion = 1`, hence data that produced images. Example:

time	location	light	temp	hum	motion
19:20:01	room1	340	22.3	48	1
19:20:15	room1	355	22.4	47	1

- `all_sensor_data` instead contains all registered sensor readings, indipendently from the motion status, which is not considered. Example:

time	location	light	temp	hum
19:20:01	room1	340	22.3	48
19:20:15	room1	355	22.4	47

- `visual_ratings` contains each rating recieved in the *Telegram Bot*. Example:

time	rating	timestamp	user_id	visual time
19:20:01	3	19:20:01	131587	16:40:32
20:25:15	5	20:25:15	152692	19:36:01

E. Visualization and Interaction Layer

On top of the stored and transmitted data lies the visualization and interaction layer. This is where sensor input is transformed into ”Visual Art”.

- **Generative Art Engine** (Python, Pygame):
  - Subscribes directly to MQTT topics.
  - Maps sensor inputs to visual parameters:
    - Light → background brightness.
    - Temperature → color palette.
    - Humidity → transparency and density.
    - Motion → triggers new abstract artwork.

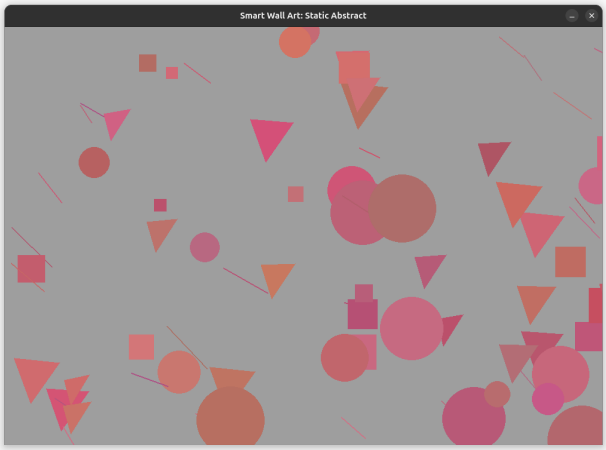

Fig. 3. Example of an image created by the script with *'light': 472, 'temperature': 31.531811459882068, 'humidity': 64.32045161097754*

- **Grafana Dashboard**:
  - Connects to InfluxDB as data source.
  - Provides historical trends and real-time monitoring.
  - Enables curators to track visitor activity and environment.

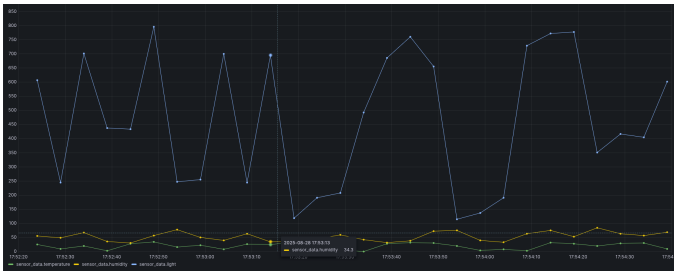

Fig. 4. Grafana panel for the application's DataBase

F. Forecasting Module

The system also incorporates a forecasting component. This module, implemented as a Python script, queries historical sensor values from InfluxDB and applies time-series modeling (ARIMA) to predict future temperature, humidity, and light levels. Forecast accuracy is evaluated using MAE (Mean Absolute Error) and MSE (Mean Squared Error) metrics, and the results can be visualized as plots of predicted versus observed values. By integrating forecasting into the architecture, the project extends its functionality beyond reactive art generation, enabling predictive insights to support active monitoring.

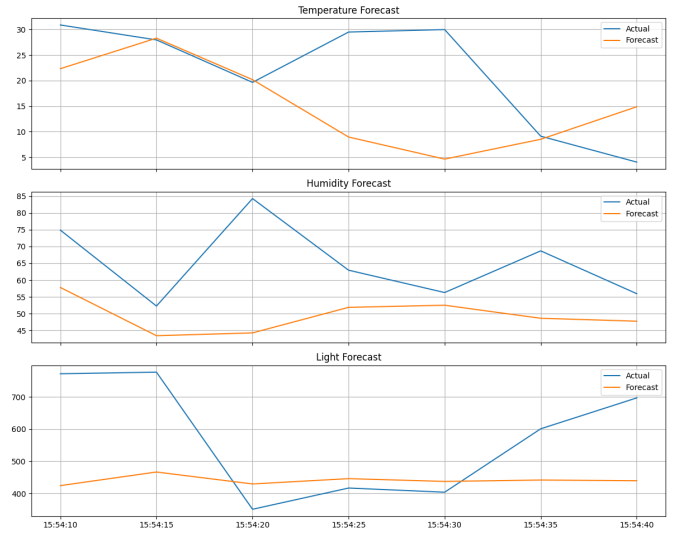

Fig. 5. Plots comparing forecasted data and registered data. produced from generated (hence partially random) data.

G. Telegram Bot Module

To incorporate user feedback into the system, a Telegram bot was developed. The bot allows users to rate the visuals generated by the art engine on a scale from 0 to 5 stars. It retrieves the most recent visual linked to a motion event and presents an inline keyboard for rating. Each submitted rating is stored in InfluxDB through an API, where it is associated with the timestamp of the corresponding visual. This module adds a direct engagement channel, enabling users to influence how the system evolves over time.

H. User Rating Model

Building upon the feedback collected by the Telegram bot, a rating model was implemented to adapt the visual output. Sensor data entries are matched to their corresponding user ratings, and the combined dataset is used to train a machine learning model. A Random Forest regressor was chosen to predict user satisfaction (rating) based on environmental conditions such as light, temperature, and humidity. The trained model is saved and can be used by the art engine to bias its figure generation towards configurations that historically received higher ratings. This module closes the feedback loop, making the system increasingly personalized and responsive to user preferences.

I. High-Level Data Flow

The overall flow of data through the system is summarized in the following diagram:

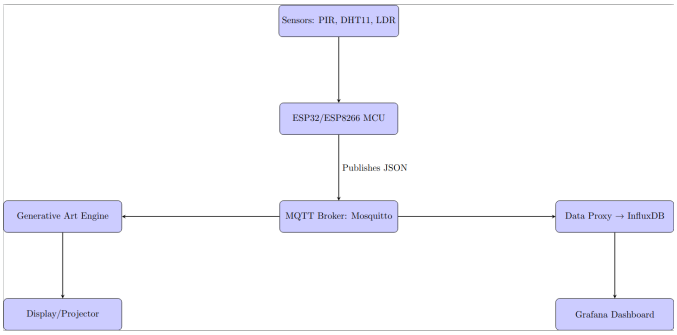

Fig. 6. Flow diagram for the application

J. Scalability and Extensions

Finally, the architecture is designed with scalability in mind. This allows the system to support additional sensors and locations.

- Support multiple locations by tagging sensor data with `location`.
- Deploy remotely via cloud MQTT brokers for distributed installations.
- Add interactive controls through dedicated MQTT topics (`smartart/control`).

III. Project Implementation

The implementation of the Smart Wall Art project followed a modular and incremental approach. Each component was first developed and validated individually, and later integrated into the complete pipeline. The following subsections describe the implementation of the main system layers.

A. Hardware and Sensor Setup

The foundation of the system consists of an ESP32 microcontroller connected to three sensors: a PIR motion sensor, a DHT11 sensor for temperature and humidity, and an LDR for ambient light intensity. The sensors were wired to digital and analog pins of the ESP32 and tested individually to verify stable readings. To avoid noisy outputs from the PIR sensor, a debouncing mechanism was implemented in the firmware.

B. Firmware Development

The ESP32 was programmed in the Arduino IDE to periodically sample temperature, humidity, and light values, while also monitoring motion events. Data are exchanged using HTTP for measurements and MQTT for configuration.

Data Communication:

- **HTTP:**
  - Environmental data are sent as JSON via `POST` requests to the `/sensor` endpoint.
  - Motion triggers are reported as {`"motion":1`} via `POST` requests to the `/motion` endpoint.
- **MQTT:**
  - Used only for configuration, subscribing to `smartart/config` to make it possible to update parameters such as the sampling rate, this feature however still isn't implemented

Data Collection:

- The **DHT11** and **LDR** are sampled periodically (default: 1 s).
- The **PIR sensor** is event-driven, sending motion events independently of periodic data.

Network Management: Wi-Fi connectivity is handled by `WiFiManager`, and the device maintains an MQTT session for configuration updates while delivering sensor and motion data via HTTP.

C. Messaging Layer

For device-to-server communication, a Mosquitto MQTT broker was installed and configured on the host machine. The broker functioned as the central messaging hub, enabling multiple consumers to subscribe to shared data streams. Although the ESP32 devices transmitted measurements via HTTP, the middleware service subsequently republished these values to the broker under the topics `smartart/sensor` and `smartart/motion`. This ensured that all downstream services could consume a uniform MQTT feed. Correct operation of the broker was verified using a simulated data publisher `simulated_sensor_publisher.py` which was used to simulate sensor data and send them using MQTT protocol.

D. Data Proxy

A Python-based middleware service (`data_proxy.py`) was developed to bridge MQTT communication and persistent storage. It subscribed to both sensor and motion topics, processed incoming JSON payloads, and wrote them into the database. All sensor readings were continuously stored in a measurement called `all_sensor_data` to preserve a complete history, used for forecasting. When motion events occurred, the most recent sensor values were combined with the motion flag and stored as a unified entry in the `sensor_data` measurement. Each record was tagged with a location identifier. To enable higher-level interaction, an API (`visual_rating_api.py`) exposed HTTP endpoints. One endpoint retrieved the most recent motion-triggered entry (`latest_visual`), while another allowed users to submit feedback by storing ratings in a dedicated `visual_ratings` measurement, tagged with user identifiers and timestamps. Together, these services bridged low-level sensor streams with user-facing feedback data, ensuring that visual events could be both recorded and evaluated.

E. Database Layer

InfluxDB was used as the time-series database to store all sensor values. The database schema was deliberately kept simple to ensure extensibility for future rooms or sensor types. Data was validated by manually querying with InfluxQL to confirm proper insertion and retrieval.

F. Visualization: Generative Art

A Python application built with Pygame was developed to transform real-time sensor data into abstract visuals. The application subscribed directly to MQTT topics. Environmental parameters modulated visual features: light controlled background brightness, temperature determined the base color palette, and humidity influenced transparency and density. Motion events acted as triggers for regenerating a new image. This ensured the displayed artwork was both dynamic and contextually aware of its environment.

G. Visualization: Grafana Dashboards

To complement the generative art, Grafana was configured as a monitoring interface. By connecting Grafana to InfluxDB, dashboards were created to visualize historical trends of temperature, humidity, and light. This method make it possible for curators to look at real-time and long-term insights into the installation's environment.

H. Forecasting Module

A forecasting component was added to extend the analytical capabilities of the system. Implemented in `forecast_data.py`, this module queried historical values from InfluxDB, resampled the data, and applied ARIMA models to predict future light, temperature, and humidity. Forecast accuracy was assessed using MAE and MSE metrics, and the results were visualized through plots comparing predicted and observed values. This predictive feature demonstrates the system's potential not only for reactive visualization but also for anticipating environmental changes.

I. Telegram Bot

User engagement was integrated via a Telegram bot. The bot allowed users to provide ratings (0–5) on the generated art styles. The rating is automatically linked to the last generated art. Each rating was logged in InfluxDB, linked to the corresponding artwork's timestamp, thus including user feedback in the database.

J. User Rating Model

Finally, a machine learning module was developed to leverage user feedback. Using the combined dataset of sensor values and Telegram ratings, a Random Forest regressor was trained to predict user satisfaction based on environmental conditions. The trained model could be used by the art engine to bias figure generation toward patterns that historically achieved higher ratings, thus closing the feedback loop and making the system more personalized over time. For the model it was chosen a ranom forest regressor because:

- It can model complex, nonlinear relationships between sensor values and ratings, which are likely since the data are based on human perception.
- Reandom forests are less sensitive to outliers and noisy data, which are common in both real-world sensor readings and subjective ratings
- It usually works pretty well out-of-the-box, with low tuning required.

The model seemed the most fitting for an initial setup with a relatively small and scattered dataset. But in a future more expanded version other, more complex, models could be explored.

K. Integration and Testing

All components were integrated into a single workflow: the ESP32 sensors published to MQTT, the data proxy logged values into InfluxDB, Grafana visualized the data, and the art engine displayed visuals that responded to sensor inputs and motion. Forecasting and rating modules added predictive and adaptive layers. End-to-end testing confirmed that sensor events propagated correctly through the pipeline and that visual outputs updated as intended.

IV. Results

A. System Functionality

In the construction of this projet, we succesfully integrated the use of sensors, MQTT communication, and data storage. To reach the construction of an IoT application that proves the unexplored potential of IoT technologies also in ”artistic” fields

B. System Stability

Although the application appears to be relatively robust, several issues require further investigation and improvement, including:

- **PIR Sensor Reliability:** While partially addressed through code-level solutions, the issue persists, likely due to the sensor’s inherent quality and potential inconsistencies in the testing environment.
- **User Base:** For the AI model that relies on user ratings to reach tuning for the art generation, a sufficiently large user base is essential to obtain accurate data. Currently, the model is based solely on generated ratings, which makes it inherently less reliable, even though it was developed using a theoretically sound foundation.

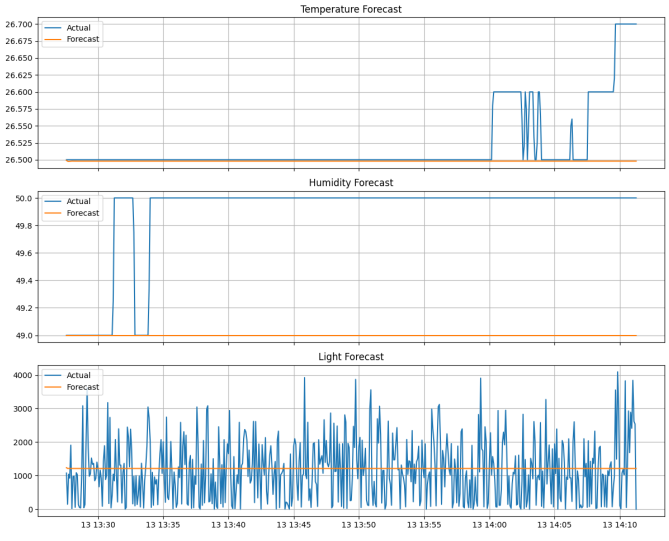

Fig. 7. Plotted forecast and real data referred in the paragraph

C. Evaluation of Forecasting Results

The forecasting script was applied to three sensor modalities: temperature, humidity, and light, with each time series resampled and interpolated to a uniform frequency of 5 seconds. To test the system, 2623 data points were evaluated per series, all harvested with consistent readings in the timespan of 4 hours.

a) Temperature.: The temperature forecasts show excellent accuracy, with a Mean Absolute Error (MAE) of 0.02 and Mean Squared Error (MSE) close to zero. The smooth progression of the interpolated series and the low error values indicate that the forecasting model is highly effective for this variable.

b) Humidity.: Humidity forecasting yielded an MAE of 0.89 and an MSE of 0.89. These values suggest moderate deviations between predictions and actual values, though still within an acceptable range for environmental monitoring. Compared to temperature, the model exhibits slightly higher variability in capturing humidity trends.

c) Light.: In contrast, light forecasts show significantly larger errors, with an MAE of 758.74 and an MSE of 860,235.44. This indicates high variability and difficulty in modeling light intensity, likely due to the strong fluctuations and abrupt changes observed in the raw series. These results suggest that further refinement or alternative models may be necessary for better results.

Overall, the evaluation highlights strong performance in forecasting temperature, reasonable accuracy for humidity, and clear limitations in modeling light intensity.

D. Future Challenges

- **Multiple Locations**: The system already includes the infrastructure to differentiate sensor readings from locations, wich could be implemented to further refine art generation or to deploy the system on multiple stations.

- **Different sensors**: Adding more sensors to reach an even deeper enviroment understanding could help refine art generation
- **Improving generation techniques**: Implementing other techniques for art generation such as generative AI and animated art, for an improved user experience
- **Enlarge User Base**: For a complete system, it is needed to reach a larger user base to create a solid foundation of ratings for the model training.