

Embedded control with movement detection: Utilizing MPU9250, MQTT, Node-RED, and InfluxDB and machine learning in esp32.

Nikolaos Mouzakis

May 21, 2025

1 Introduction

This project integrates an ESP32 microcontroller with an MPU9250 inertial measurement unit (IMU) sensor to detect device movements and control 2 LEDs and PWM outputs accordingly. It implements a mechanism to persist and restore device state via MQTT messaging, Node-RED flow processing, and data storage in an InfluxDB time-series database.

The key functionalities are:

- Reading accelerometer and gyroscope Y-axis data from the MPU9250.
- Applying threshold-based logic or machine learning to detect movement and determine LED and PWM states.
- Publishing device status data over MQTT.
- Restoring device LED and PWM states after reboot using InfluxDB queries and Node-RED.

2 MPU9250 Sensor Data and Movement Detection

2.1 Sensor Overview

The MPU9250 is a 9-axis IMU sensor combining a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer. For this project, only the accelerometer Y-axis and gyroscope Y-axis values are utilized.

2.2 Data Acquisition

The ESP32 reads raw sensor values for accelerometer Y (ay) and gyroscope Y (gy) at periodic intervals.

2.3 Threshold-Based Movement Detection

Threshold values are defined empirically to decide when the ESP32 should change the states of its LEDs and PWM output.

2.4 Controlling LEDs and PWM

Based on movement detection, the device sets the LEDs and PWM.

2.5 Status Publishing

The ESP32 publishes its current LED and PWM states on the MQTT topic `esp32/status` as a formatted string:

```
LED1:OFF LED2:ON PWM:15
```

This message serves to provide the real-time status of device outputs.

3 MQTT Communication Architecture

The ESP32 communicates with the Node-RED server via MQTT. Topics used include:

- `esp32/status` — Periodic status updates including LED and PWM states.
- `esp32/restore_request` — Sent by ESP32 at startup requesting last stored device state.
- `esp32/restore_state` — Published by Node-RED after querying InfluxDB with the restore state data.

3.1 Restore request and response flow

Upon powering on, on startup, the ESP32 immediately publishes a `get_latest` message on `esp32/restore_request`. This triggers Node-RED to query the latest device state stored in InfluxDB and respond by publishing a formatted restore state message on `esp32/restore_state`.

The ESP32 listens for this restore message and applies the LED and PWM settings accordingly before continuing normal operation. Restoration can be observed in the following Figure, where the Esp32 boots and assigns immediately the latest saved state on the LEDs.

```

AccZ: -7.57
ax:156.90 ay:-3.27 az:-7.57
GyroX: -0.02
GyroY: 0.11
GyroZ: -0.14
gx:-0.02 gy:0.11 gz:-0.14
mx: 46.85 my: 18.52 mz: 103.31PWM level now: 88
ay=-3.27, gy=0.11, LED1=OFF, LED2=ON, PWM=88
ets Jul 29 2019 12:21:46
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:4688
load:0x40078000,len:15460
ho 0 tail 12 room 4
load:0x40080400,len:4
load:0x40080404,len:3196
entry 0x400805a4
operating
Connecting to wifi..
WiFi connected - IP address: 192.168.88.42
connecting to MPU...
Connecting to MQTT...
MQTT connected
Sent restore request
Received message on topic: esp32/restore state
Payload: LED1:OFF LED2:ON PWM:88
RECEIVED RESTORE MESSAGE
Restored state from InfluxDB

```

Figure 1: Latest state restoration as observed over UART

4 Node-RED and InfluxDB Integration

4.1 Node-RED Flow Design

Node-RED acts as the middleware for the bridging of MQTT and InfluxDB:

1. **MQTT In node** listens on `esp32/restore_request` and detects the `get_latest` message.
2. **InfluxDB Query node** performs a time-series query fetching the most recent LED and PWM state values.
3. **Function node** formats the InfluxDB query response into the MQTT payload string.
4. **MQTT Out node** publishes the formatted message on `esp32/restore_state`.

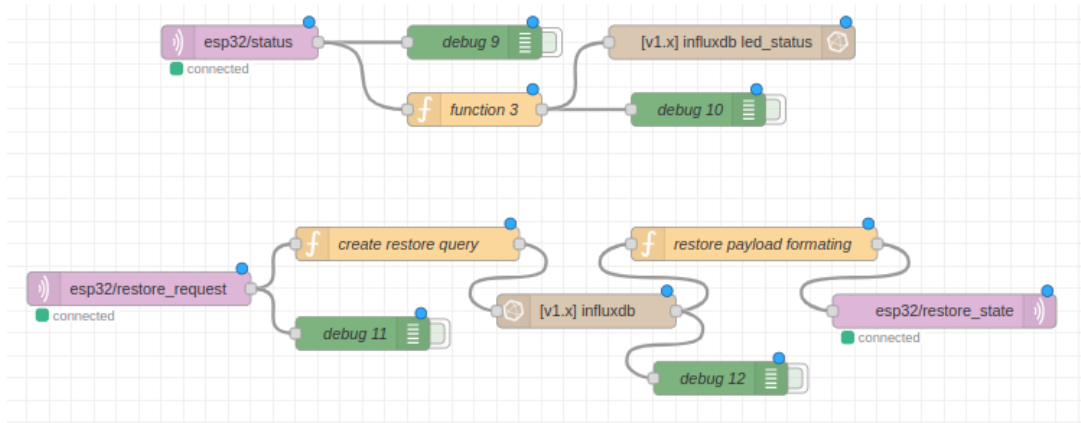


Figure 2: Node-Red data and processing flow

4.2 InfluxDB Query

As for the InfluxDB, version 1.6.4 has been utilized and installed as

```
echo "deb https://repos.influxdata.com/ubuntu bionic stable" | sudo tee /etc/apt/sources.list.d/influxdb.list
sudo curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -
sudo apt get update
sudo apt-get install -y influxdb
```

After installation, a database is created to be used for this project.

```
influx
> create database esp32db
```

In `function3` of the Node-Red then, when run the following Javascript code in order to push data in the same database.

```
let parts = msg.payload.split(" ");
let led1 = parts[0].split(":")[1] === "ON" ? 1 : 0;
let led2 = parts[1].split(":")[1] === "ON" ? 1 : 0;
let pwm = parseInt(parts[2].split(":")[1]);
```

```
msg.payload = {
  LED1: led1,
  LED2: led2,
  PWM: pwm
};
msg.measurement = "led_status";
return msg;
```

On the InfluxDb out node in Node-Red is also specified and set the measurement's name as "led_status" and on the server configuration the respective host ip, port number and database name as "esp32db".

The InfluxDB query used in `create_restore_query` node is:

```
msg.query = 'SELECT LAST("LED1"), LAST("LED2"), LAST("PWM") FROM "led_status"';
return msg;
```

This query retrieves the most recent LED1, LED2, and PWM values stored.

In the `restore payload forming` node of Node-Red response is formatted to be given as an input on the `restore_state` topic as:

```
let row = msg.payload[0]; // first row of result

let LED1 = row.last === 1 ? "ON" : "OFF";
let LED2 = row.last_1 === 1 ? "ON" : "OFF";
let PWM = row.last_2;

msg.payload = `LED1:${LED1} LED2:${LED2} PWM:${PWM}`;
msg.topic = "esp32/restore_state";
return msg;
```

4.3 State Restoration

Upon receiving the restore payload, the ESP32 parses the string and applies the saved LED and PWM states, ensuring the desired recovery.

4.4 UI visualization

For the UI visualization 2 switches reflecting the current state of LED have been employed and a slider for representing the common PWM applied to them.

A new function named `function4` have been used to do the same operation as `function3` with the difference of returning 3 outputs, one by one to be passed on the switches and the slider.

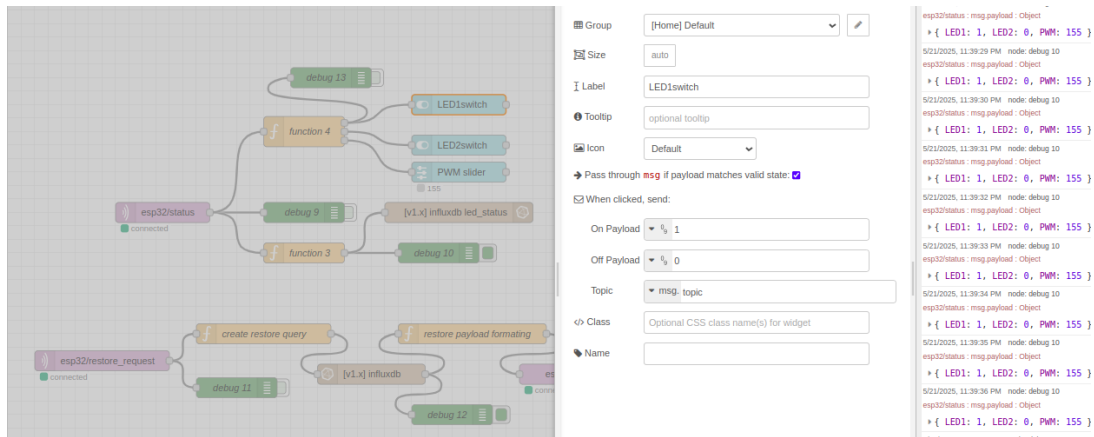


Figure 3: configuration show for slider of led1.

A screenshot of the visualization is presented below, having the 2 switches and the slider, and for debug purposes the plots of the incoming data from accelerometer and gyrometer.

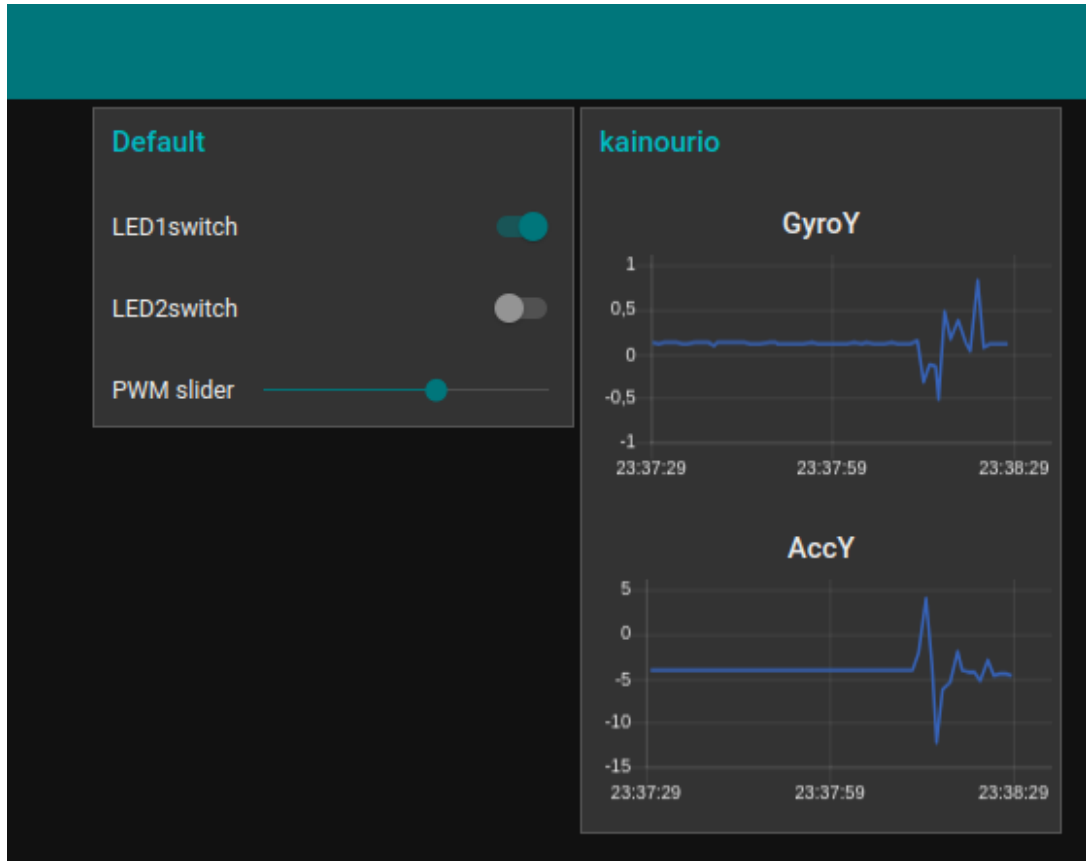


Figure 4: Node-Red UI.

5 Summary of Data Flow

Component	Topic / Action	Description
ESP32	Publish esp32/restore_request	Sends <code>get_latest</code> on startup
Node-RED	MQTT In on esp32/restore_request	Detects restore request
Node-RED	InfluxDB query	Fetch last saved LED/PWM states
Node-RED	Function node	Format restore state payload
Node-RED	MQTT Out on esp32/restore_state	Publishes restore state
ESP32	MQTT Subscribe esp32/restore_state	Receives and applies restore state
ESP32	Publish esp32/status	Periodic status updates

6 Motion detection via machine learning

In the second part, the goal is to move from the threshold supported movement detection and achieve the operational functionalities by employing machine learning.