Host-Device Communication — Wi-Fi Wireless Control via WebSocket (Windows)

This guide is for Windows users who want **low-latency, real-time control** of their robot through Wi-Fi.

By using **WebSocket**, your PC and the robot driver board can exchange data in both directions instantly — without the delay or polling limitations of traditional HTTP requests.

> 📦 GitHub Repository:
>
> https://github.com/EffectsMachine/robot_driver_with_esp32s3_lite

---

1. Why WebSocket?

WebSocket is a **full-duplex protocol built on TCP**, allowing persistent, two-way communication between your computer (host) and the robot (device).

It’s ideal for robotics applications that require continuous state updates and fast command response — typically up to **60 messages per second**.

Feature	Description
✅ **Low latency & persistent connection**	The connection is established once, avoiding repeated handshakes like HTTP. Response latency can reach the millisecond level — perfect for motion control.
✅ **True bidirectional data flow**	Both sides can actively send data — commands from the PC and status feedback from the robot — with no need for polling.
✅ **Wide compatibility**	Uses the same ports as HTTP (80/ws, 443/wss), bypasses most firewalls, and is supported in all major languages (Python, JavaScript, etc.).

Feature	Description
⚠️ **Requires connection management**	You'll need to handle reconnection if Wi-Fi drops — unlike HTTP's one-shot model.
⚠️ **Slightly harder to debug**	Standard browsers can't test it directly; use tools like **Postman (WebSocket mode)** or scripts instead.

2. How It Works

The ESP32-based driver board runs a lightweight **WebSocket server** that shares the same port (`80`) as HTTP, with the fixed endpoint `/ws`.

Once powered on, the board automatically starts the WebSocket service.

Communication workflow:

1. **Connection** — The PC sends a handshake request and establishes a persistent link.
2. **Data exchange** —
   - **PC → Device:** Sends JSON commands (e.g., motion control, parameter updates).
   - **Device → PC:** Pushes status data (e.g., sensor values, execution feedback, battery level).
3. **Disconnection** — Either side can close the link, or it will drop automatically if the network disconnects (you may need to reconnect manually).

All commands use the same JSON format as HTTP, for example:

```
{"T":202,"line":1,"text":"webSocket msg","update":1}
```

3. Environment Setup

The WebSocket examples use the same environment as the HTTP examples.

Step 1. Install Python

Download and install **Python 3.10 or later** from

https://www.python.org/downloads/

Be sure to check **“Add Python to PATH”** during installation.

Step 2. Open the Example Folder

Navigate to:

```
..\Example for Robot Driver Lite\python_example\webSocket
```

Right-click inside the folder → **Open in Terminal**.

Step 3. Create and Activate a Virtual Environment

You only need to do this once (you can reuse the same environment for both HTTP and WebSocket examples).

```
python -m venv venv
.\venv\Scripts\activate
```

After activation, your terminal prompt should start with `(venv)`.

Step 4. Install Dependencies

```
pip install -r requirements.txt
```

---

4. Run the WebSocket Example

Step 1. Edit the Example

Open `webSocket_example.py` using any text editor (Notepad, VS Code, Sublime, etc.).

Update the **IP address** in the file to match your device:

- If your PC connects directly to the board’s hotspot:
  `ESP32_IP = "192.168.4.1"`
- If both are on the same LAN (e.g., via a router):

  Use the IP shown on the board’s OLED display.

You can also edit the **command payload** inside the `on_open()` function if you want to send custom instructions:

```
def on_open(ws):
    print("Connected to ESP32 WebSocket")

    command = {
        "T": 202,
        "line": 1,
        "text": "webSocket msg",
        "update": 1
    }

    ws.send(json.dumps(command))
    print(f"Sent: {command}")
```

Step 2. Run the Script

Make sure the board is powered on and on the same network as your PC, then run:

```
python webSocket_example.py
```

Step 3. Check the Output

- ✅ **Connection successful:**

  Terminal shows `Connected to ESP32 WebSocket` and the sent command.

- 📩 **Receiving data:**

  If the robot pushes feedback (e.g., sensor data), you'll see `recv: {...}` in the terminal.

- ❌ **Connection failed:**

  Check that the IP is correct and that the WebSocket service is running.

"connection refused" usually means the device isn't reachable or WebSocket isn't enabled.

To stop the script, press **Ctrl + C**.

---

5. Code Overview

The main functions inside `webSocket_example.py` are structured for easy customization:

Function	Purpose
`on_open(ws)`	Called once when the connection is established. Used to send initial commands.
`on_message(ws, message)`	Handles incoming messages from the robot (status, feedback, etc.).
`on_error(ws, error)`	Prints connection or runtime errors.
`on_close(ws, close_status_code, close_msg)`	Called when the connection closes — ideal for auto-reconnect logic.

Developers can extend this script to implement real-time robot control dashboards, telemetry viewers, or automatic command sequences.

---

6. Summary

Using WebSocket provides:

- **Low latency** and **continuous control**
- **Real-time feedback** from the robot
- **Simple setup** and **standard JSON messages**

Once connected, you can control movement, read sensors, and receive robot status updates in real time — all from your PC.