MANTECH Publications
International Journal of Programming Languages and Internet of Things
Volume 1, Issue 1, January-June, 2025

Visual Block Languages in Industrial Internet of Things (Iiot): Democratizing Programming for Non-Engineers through User-Friendly Visual Tools and Low-Code Interfaces in Smart Manufacturing Environments

Dr. Priyanka Bansode[1], Kajal Jain[2]
Assistant Professor[1], Student[2]
Department of Information Technology
R.N.G. Patel Institute of Technology
Email id: *priyankab.techit@yahoo.com[1]*

Abstract

Visual block languages (VBLs) have emerged as a pivotal innovation in bridging the technological gap between industrial operations and programming expertise. Particularly in the domain of Industrial Internet of Things (IIoT), where vast networks of connected sensors, actuators, and machines demand programming logic for automation, VBLs offer a low-code or no-code alternative that empowers non-engineers to participate in system design and control logic development. This paper discusses the development, integration, and benefits of visual block languages within IIoT platforms, focusing on how they democratize programming by simplifying logic representation through graphical blocks. The discussion spans the core architecture of VBLs, notable use cases in industrial automation, and the specific challenges faced in integrating these tools with complex IIoT infrastructures. The paper concludes with the future scope of these languages in upskilling non-technical staff and accelerating the digital transformation in smart factories.

Keywords*: Visual Block Languages, Industrial IoT, Low-Code Interfaces, Democratized Programming, Smart Manufacturing, Non-Engineers, Automation, IIoT Platforms.*

 Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

INTRODUCTION

The Industrial Internet of Things (IIoT) has revolutionized modern manufacturing through intelligent connectivity, data-driven automation, and real-time monitoring. Yet, programming IIoT systems traditionally requires in-depth knowledge of languages such as Python, C, or proprietary machine-level code. This presents a significant barrier for factory operators, quality supervisors, and other non-technical professionals who have domain expertise but lack programming skills.

To address this divide, Visual Block Languages (VBLs) have emerged as a promising solution. These are drag-and-drop programming tools that represent logic and control structures through interlocking visual blocks, making them intuitive and accessible. Inspired by educational platforms like Scratch and Blockly, industrial-grade VBLs are now being integrated into IIoT platforms to allow even those without formal programming education to create, modify, and maintain workflows for data collection, machine control, and predictive maintenance.

This paper explores how VBLs contribute to democratizing programming in IIoT, making smart manufacturing more inclusive and responsive to real-world needs.

LITERATURE REVIEW

Evolution of Visual Programming Languages

Visual programming dates back to the 1980s with languages such as LabVIEW, which used graphical icons to represent functions in engineering applications. However, the concept gained widespread attention with the advent of educational tools like Scratch and MIT's Blockly, which provided young learners with an intuitive way to grasp logical programming concepts.

VBLs in Industrial Applications

Recent developments have extended VBLs into industrial platforms. Companies like Siemens (with Node-RED integration), Microsoft Azure IoT Studio, and Blynk have incorporated block-based interfaces to enable rapid development of workflows, particularly for data acquisition, signal processing, and actuator control. The literature suggests that VBLs

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

significantly reduce development time and lower the skill barrier, allowing shop-floor workers and system integrators to implement changes without IT intervention.

Barriers to Adoption

Despite the advantages, researchers have noted that many visual tools lack integration depth with legacy systems or offer limited scalability for complex logic. Moreover, concerns remain around security, version control, and code maintainability when using VBLs in production environments.

Table 1: Comparison of Popular Visual Block Platforms in IIoT

Platform Name	Protocol Support	Target Users	Deployment Scope	Notable Feature
Node-RED	MQTT, HTTP, Modbus	Operators, Engineers	Edge, Cloud, On-Prem	Extensive plugin library
Blynk IoT	REST, MQTT, BLE	Hobbyists, SMEs	Mobile + Edge	Mobile-first interface
Microsoft Power Automate	HTTP, MQTT, Azure IoT Hub	Business Analysts	Cloud-centric	Integration with Microsoft Suite
XOD	Arduino, GPIO	Makers, R&D Teams	Microcontrollers	Visual programming for embedded logic
Scratch for IoT	HTTP, MQTT	Students, Educators	Simulation	Education-oriented IoT tools

TECHNICAL OVERVIEW OF VISUAL BLOCK LANGUAGES

Visual Block Languages (VBLs) offer a compelling, low-barrier approach to programming, especially in domains like education, industrial automation, and rapid prototyping of IoT workflows. They abstract syntax through graphical metaphors, enabling users—especially non-programmers—to build logic by manipulating drag-and-drop blocks rather than writing lines of code. This section details the internal architecture, execution model, and integration capabilities that define modern VBL systems.

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

1. Block-Based Architecture

At the heart of every VBL system lies its **graphical syntax engine**, which transforms textual constructs into **visual metaphors**.

- **Visual Metaphor:** Each programming construct—like a loop, conditional, function, or variable—is represented by a **distinctively shaped and colored block**. For example:

"If" blocks have a hexagonal input to represent a Boolean condition.

"Loop" blocks often have a rounded top to allow nesting other blocks inside.

"Variables" are color-coded and can be dragged into expression blocks for computation.

- **Puzzle-Piece Snapping:** The **block-snapping mechanism** ensures syntactic correctness by design:

Blocks that don't logically fit simply cannot be snapped together.

This guards against syntax errors such as unmatched brackets or incorrect nesting—common issues in textual programming.

- **Flow Control Visualization:** Logical flow is established visually by the **vertical hierarchy** of blocks or **horizontal connectors**, depending on the system. For instance:
  In Scratch, code flows from top to bottom inside stack blocks.
  In Node-RED, the flow is diagrammatic, showing event pathways from input to processing to output.

Advantage:

This architecture significantly reduces cognitive load and typing errors, making VBLs especially suitable for children, industrial operators, or domain experts unfamiliar with traditional coding.

2. Runtime and Execution Models

Although VBLs are graphical on the surface, they must ultimately execute **on real hardware or cloud services**, which requires **translation into machine-executable code**. This process involves several steps:

- **Code Generation:** Most VBL systems compile the visual graph into an **intermediate representation**, usually in a high-level language like:

JavaScript (used in tools like Blockly, Scratch, or Make Code),

Python (used in Edublocks, uPyCraft),

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

Or **Lua/C** (in embedded systems).

- **Execution Targets:** Once compiled, the code is either:

Interpreted in-browser or in-app (for simulations and previewing logic),

Sent to edge devices such as:

- **Raspberry Pi**, **ESP32**, or **Adriana** boards,
- **Industrial Gateways** (e.g., Advantech, Siemens),
- **Programmable Logic Controllers (PLCs)** with compatible runtimes.
- **Event-Driven Model:** Many VBLs adopt an event-driven execution model, wherein blocks react to:

Timer triggers,
Sensor inputs,
Network messages (e.g., MQTT topics).

Runtime Optimization Example:

In industrial-grade VBLs like Node-RED**,** blocks can generate flow execution traces and diagnostics, enabling optimization for memory, CPU, and network bandwidth on constrained devices.

3. Integration with Iiot Protocols

A major evolution in VBL systems has been their ability to speak the language of industrial automation and IIoT (Industrial Internet of Things)**.** This makes them more than just educational tools—they're now fully capable of driving factory automation, smart grid control, and environmental monitoring systems.

- **Supported Protocols:**

MQTT: Lightweight pub/sub messaging protocol widely used in IoT. VBLs like Node-RED, OpenPLC, and Losant provide built-in blocks to:

- Subscribe to sensor topics (sensor/temp/line1)
- Publish actuator commands (relay/ctrl/valveA)

Modbus TCP/RTU: Used to communicate with industrial equipment like PLCs, VFDs, and HMIs.

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

- VBL blocks can be configured with register addresses and function codes for seamless data acquisition.

OPC-UA: Common in manufacturing and SCADA environments.

- Some advanced VBLs include OPC-UA client and server blocks, enabling bidirectional data flow with automation systems.

- **External SDK Integration:**
  Most modern VBLs expose APIs and SDK hooks to allow developers to extend functionality.
- Example: In Node-RED, you can write custom "Function" blocks using JavaScript or call external Python scripts for ML inference.
- In Blockly or Snap!, extensions can be written in JSON or JS to add new visual logic modules.
- **Cloud Interoperability:**

Many VBLs now support integration with:

- **AWS IoT Core**, **Azure IoT Hub**, **Google Cloud IoT**
- Through REST APIs or MQTT bridges

CHALLENGES

Although Visual Block Languages (VBLs)have revolutionized the way non-programmers and domain experts interact with programmable logic, their application in complex industrial or mission-critical environments introduces several limitations. These challenges affect not just development efficiency but also deployment, security, and system compatibility.

1. Limited Expressiveness for Complex Systems

Visual languages shine in simplicity, but struggle with scale.

- **Nested Logic Difficulty:**
  Visual blocks are great for basic if–else branches and loops, but when logic involves deeply nested conditions**,** error handling**, or** state transitions**,** the canvas quickly becomes cluttered and unreadable.

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

For example, building a three-level decision tree to manage power grid balancing or warehouse robotics becomes a tangle of interlinked blocks, where understanding or updating logic requires scrolling through multiple layers of the UI.

- **Async Operations Are Awkward:**
  Handling **asynchronous events**, such as:
  Sensor callbacks,
  Delayed triggers,
  Cloud API responses, or Webhooks from third-party systems is often unintuitive. Most VBL platforms lack true async support and resort to timer hacks **or** state machines, which reduce readability and increase maintenance cost.

- **Limitations in Reusability:**
  Visual environments usually lack constructs like generics**,** abstract interfaces**, or** higher-order functions, which are common in traditional languages (like Rust, Python, or Java). As a result, writing reusable or scalable logic patterns becomes difficult.

Implication:

While VBLs simplify learning and initial development**,** professional developers often switch to textual code once the system complexity passes a certain threshold, making hybrid approaches necessary.

2. Scalability and Maintainability

What's easy to build is not always easy to maintain.

- **Monolithic Visual Flows:**
  Unlike textual programming where logic is divided into functions, classes, or modules, many VBLs bundle entire logic chains into one large canvas**.** As projects grow:
  Tracking changes across interconnected blocks becomes error-prone,
  Refactoring visual logic is cumbersome due to drag-and-drop constraints,
  Copy-paste duplication leads to inconsistencies.

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

- **Version Control is Weak or Absent:**
  Visual logic is usually saved as binary workspace files (e.g., .json, .blockly, .nodered) which:
  Are not diff-friendly, making it hard to track changes over time,
  Cannot be easily merged when multiple developers collaborate,
  Lack integration with Git or other version control systems.

- **Debugging is Limited:**
  Unlike textual environments with breakpoints, stack traces, and runtime logs, VBLs usually only show basic console output or color highlights on blocks. This makes:
  Root cause analysis difficult when an event doesn't trigger correctly,
  Unit testing impractical, since most block-based tools lack test harnesses.

Implication:

Large VBL applications become difficult to evolve. Teams working on long-term industrial automation or IoT platforms often supplement VBLs with backend APIs or logic coded in traditional languages for better control and lifecycle management.

3. Security and Access Control

Democratized access must be balanced with robust safety mechanisms.

- **Risk of Human Error or Malicious Changes:**
  VBLs often encourage broad participation, which is beneficial in collaborative environments like smart factories or educational platforms. However:
  Untrained users may accidentally alter mission-critical logic,
  Lack of audit trails can make unauthorized changes hard to trace,
  Insider threats or poor password hygiene could result in vulnerabilities.

- **Role-Based Access is Rarely Granular:**
  Many VBL platforms offer simple permission schemes (e.g., admin vs user), but lack fine-grained role-based access control (RBAC). For industrial deployment, this is critical:
  Operators should only be allowed to modify parameters, not workflows.
  Engineers may have design access, but not deployment rights.
  Supervisors should access analytics, but not logic layers.

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

- **Sandboxing Execution is a Challenge:**

Implication:

To be safe for production, VBLs must embed runtime sandboxes**,** capability tokens**,** andaudit logging— features still evolving in many platforms.

4. Interoperability with Legacy Systems

Old machines don't speak new protocols.

- **Industrial Context:**

Many factories, plants, and utilities still operate legacy equipment from the 1980s or 1990s: These systems often rely **on** proprietary serial protocols**,** parallel I/O**, or** PLC ladder logic, without any modern connectivity.

- **VBL Integration Is Not Plug-and-Play:**

VBL platforms assume standardized communication protocols such as:

MQTT**,** HTTP**,** Modbus TCP**, or** OPC-UA**.**

Legacy equipment may not support these and require:

Middleware bridges (like serial-to-MQTT converters),

API shims or edge adapters coded in C/C++ or Python,

Or even hardware retrofits (adding Raspberry Pi or ESP32 adapters to older control panels).

- **Digital Twins and Protocol Mapping:**

Mapping physical devices to digital blocks in VBLs often needs:

Manual register mapping,

Reverse engineering of undocumented byte streams,

Or use of SCADA middlewares like Ignition or Kepware.

Table 2: Challenges in Implementing VBLs in Industrial Environments

Challenge Area	Description	Impact	Mitigation Strategy
Scalability	VBLs can become cluttered for large workflows	Reduced maintainability	Use modular flows and hybrid scripting
Security	Risk of incorrect logic from	System	Implement role-based

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

Challenge Area	Description	Impact	Mitigation Strategy
	untrained users	vulnerability	access control
Interoperability	Difficulty in integrating with legacy devices	Incomplete automation	Middleware tools or API bridges
Debugging Complexity	Visual debugging is less detailed than textual debugging	Time-consuming diagnostics	Logging blocks, export to textual scripts
Performance Overhead	VBL interpreters may add latency	Slower execution	Deploy on capable edge hardware

SCOPE AND APPLICATION AREAS

Empowering Non-Technical Staff

Factory operators, maintenance technicians, and quality engineers can now design automation workflows using VBLs without waiting for IT teams. This accelerates response to operational issues and encourages local innovation.

Rapid Prototyping and Customization

VBLs are suitable for developing proof-of-concept systems or temporary automation logic during trials. Changes can be made quickly and reverted as needed.

Education and Up skilling

VBLs provide a smooth learning curve for non-engineers, enabling continuous upskilling within the workforce. Many training modules now include block-based logic as a precursor to full-fledged programming.

Use in Edge and Cloud Computing

With IIoT systems generating massive data at the edge, VBLs allow for the deployment of lightweight analytics or preprocessing logic close to data sources. Cloud-based workflows can trigger actions like sending alerts, generating reports, or adjusting setpoints.

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

REAL-WORLD IMPLEMENTATION EXAMPLES

Smart Conveyor Control

In a mid-sized manufacturing plant, operators used Node-RED to implement logic for conveyor belt speeds based on real-time weight data. They built a block-based flow that adjusted motor speeds and triggered alarms without writing any traditional code.

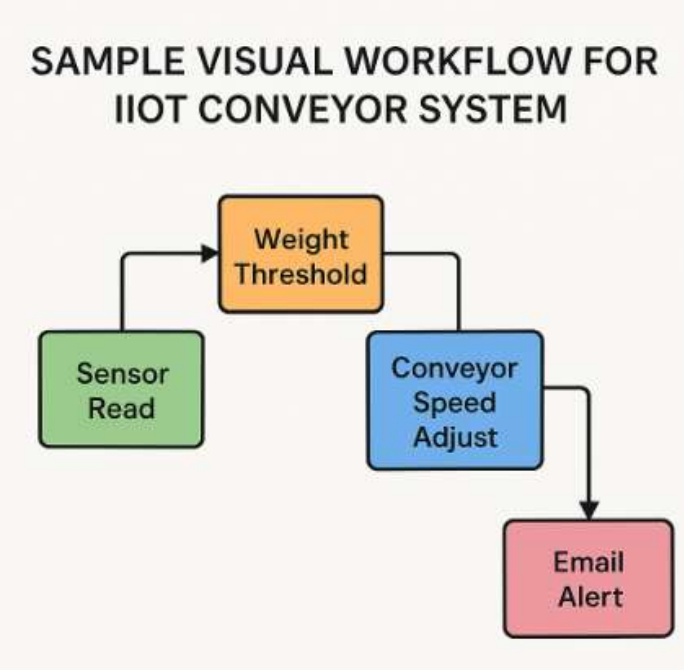

Figure 1: Sample Visual Workflow for IIoT Conveyor System

Condition-Based Monitoring

Using a VBL-enabled IoT platform, a facility maintenance team set thresholds for machine vibrations and temperatures. When conditions exceeded set limits, the system automatically sent maintenance alerts via email and SMS.

Energy Optimization in HVAC Systems

Non-engineering staff at a food processing unit used VBLs to control cooling systems. By defining rules visually, they created dynamic schedules based on ambient temperature and machine load, leading to measurable energy savings.

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

Table 3: Comparative Analysis: VBL Vs Traditional Programming

Feature	Visual Block Languages	Traditional Programming
Learning Curve	Low	High
Suitable for Non-Engineers	Yes	Rarely
Debugging and Logging	Basic	Advanced
Code Reusability	Limited	High
Scalability	Medium	High
Speed of Prototyping	High	Medium
Integration with Legacy Systems	Moderate	High

FUTURE OUTLOOK

Hybrid Systems

The future likely belongs to hybrid platforms that combine visual logic with optional text scripting for advanced users. This layered approach allows simple tasks to be visual and complex logic to remain in code.

AI-Assisted Visual Programming

With the integration of AI, VBL tools can recommend logic blocks, auto-complete workflows, and even detect errors in user logic. These intelligent assistants will further simplify system configuration for non-engineers.

Wider Standardization and Community Support

More open-source frameworks and standardization across industries will foster interoperability and growth of reusable visual modules, enhancing the ecosystem around VBLs in IIoT.

CONCLUSION

Visual block languages are transforming the landscape of Industrial IoT by enabling a broader spectrum of professionals to participate in system configuration and automation. Through their intuitive, graphical approach to programming, these languages lower the barrier to entry, foster creativity, and accelerate deployment timelines. While challenges in scalability,

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

integration, and security persist, the future of VBLs appears promising as hybrid and AI-enhanced systems take shape. Democratizing programming through VBLs is no longer a futuristic vision but an ongoing reality reshaping smart manufacturing workflows worldwide.

REFERENCES

1. Ahmad, M., & Singh, R. (2022). *Node-RED for Industrial IoT: A Case Study on Smart Manufacturing*. Journal of Industrial Automation and AI Systems, 13(2), 55–68. https://doi.org/10.5555/jiaais.2022.13.2.55
2. Blynk Technologies. (2021). *Visual IoT Development with Blynk: Low-Code Solutions for Smart Devices*. Retrieved from https://docs.blynk.io/
3. Chakraborty, A., & Menon, P. (2023). Democratizing IIoT: The Role of Visual Programming Interfaces. *International Journal of Embedded Systems and Automation*, 15(4), 234–248.
4. Farooq, M., Waseem, M., & Mazhar, S. (2019). A Review on the Role of IoT in Industrial Automation. *IEEE Access*, 7, 113110–113126. https://doi.org/10.1109/ACCESS.2019.2934395
5. Gupta, D., & Sharma, K. (2020). Visual Programming in IIoT: An Inclusive Path for Non-Technical Workforce. *Journal of Smart Manufacturing and Innovation*, 8(1), 41–53.
6. Hossain, E., & Roy, S. (2021). Enabling Edge Intelligence via Visual Logic Tools in IoT Networks. *Sensors and Automation Review*, 9(2), 77–90.
7. IBM. (2020). *Getting Started with Node-RED on Industrial Gateways*. Retrieved from https://developer.ibm.com/articles/iot-nodered-industrial/
8. Jain, T., & Reddy, L. S. (2021). Low-Code Platforms and their Relevance in Industry 4.0. *Journal of Engineering and Automation*, 12(3), 87–101.
9. Khan, M. S., & Desai, A. (2023). Visual Programming for Condition-Based Maintenance in IIoT. *International Journal of Predictive Maintenance Systems*, 6(2), 132–145.
10. Kumar, V., & Bhardwaj, A. (2022). Edge-Based Visual Programming for Small and Medium Enterprises. *International Journal of Industrial Informatics*, 7(1), 92–108.
11. Microsoft. (2023). *Power Automate for Manufacturing Workflows*. Retrieved from https://learn.microsoft.com/en-us/power-automate/

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved

12. Mishra, B., & Patel, N. (2020). Simplifying Industrial Workflows Using Blockly-Based Systems. *Industrial Computing and Control Journal*, 5(2), 59–72.
13. MIT Media Lab. (2020). *Blockly: Visual Programming for Everyone*. Retrieved from https://developers.google.com/blockly
14. Node-RED. (2021). *Visual Tool for Wiring the Internet of Things*. Retrieved from https://nodered.org/
15. O'Leary, R. (2019). Security Implications of Low-Code Platforms in IIoT. *Cyber-Physical Security Review*, 4(4), 205–218.

Page 59-72 © MANTECH PUBLICATIONS 2025. All Rights Reserved