|  |  |  |
| --- | --- | --- |
|  | **SMART PART VERIFCATION IN AUTOMOTIVE MANUFACTURING PLANT – BATCH 2** |  |
|  |  |  |
| Stage | Activity | Tool | Outcome |
| Stage I: Brainstorming | Group Discussion on requirements and solution model Challenges – Misidentification of parts, improper installations, manual verification delays  Opportunities – Real-time defect detection, predictive maintenance using ML models  Ideas – Extendable to full vehicle assembly audit, AI-based adaptive learning for defect types    FLASH CARD Type: Industry 4.0, Smart Manufacturing Domain: Automotive Quality Assurance & Process Optimization Stakeholders: Line supervisors, Plant QA teams, OEMs, Automation teams Technologies: PIR, IR, and Color sensors, MQTT, Cloud Computing, Machine Learning, Dashboards | A Smart Part Verification System for automotive manufacturing plants that can: (i) detect the presence of parts using PIR and IR sensors (ii) verify part type and color using a color sort sensor (iii) transmit sensor data to the cloud using MQTT (iv) analyze data with machine learning models to detect incorrect/missing parts (v) visualize insights and trends through dashboards for root cause analysis (vi) trigger real-time alerts for manual or robotic intervention to ensure minimal downtime and improved quality |
| Stage II: Idea Posting |  | In modern automotive manufacturing, ensuring the correct installation of parts is critical to maintaining product quality and minimizing rework. This project presents a smart part verification system using a combination of PIR (Passive Infrared), IR (Infrared), and Color Sort sensors to detect the presence, type, and color of components on the assembly line. The sensor data is transmitted to the cloud using the MQTT protocol, where machine learning models and analytics pipelines process it to identify installation defects such as missing parts, mismatches, or incorrect placements.  The system incorporates real-time dashboards that visualize defect trends and support root cause analysis. By generating instant alerts for human or robotic intervention, the solution enhances operational efficiency, reduces downtime, and minimizes production errors. This intelligent verification framework contributes significantly to automation and quality assurance in automotive assembly environments. |
| Stage III: Customer  Mapping | 1. For Assembly Line Operators  * What issues do you face with missing or incorrect part installations? * How often do you manually inspect part placements?  1. For Quality Assurance (QA) Inspectors  * How do you currently verify part type and color? * What accuracy level do you expect from an automated system?  1. For Maintenance Engineers  * How do you identify faulty sensors or system failures? * Would predictive maintenance alerts improve your workflow?  1. For Automation & Robotics Teams  * Do you need real-time alerts for robotic intervention? * How should sensor data integrate with your existing control systems? | Requirement Specification from Customer Mapping  **Assembly Line Operators** require a system that immediately detects missing/incorrect parts and reduces dependency on manual verification.  **Quality Assurance Inspectors** need highly accurate verification of part presence, type, and color for compliance and quality reporting.  **Maintenance Engineers** want the system to support predictive maintenance by identifying potential faults in sensors or communication.  **Automation Teams** need seamless MQTT-based data flow and real-time alerts for initiating robotic corrections without human delay. |
| Stage IV:  Idea Layout |  | This Smart Part Verification system for automotive manufacturing integrates hardware sensors with intelligent software to ensure accurate part installation and reduce rework. The system uses **PIR, IR, and color sensors** to detect part presence, verify correct seating, and confirm component color. These sensor inputs are processed locally using **edge computing**, enabling **real-time alerts** and operator feedback through a **simple dashboard**. This ensures quick error detection and immediate corrective actions on the production line.  On the data side, the system tracks **installation errors**, maintains **quality records** linked to each vehicle's VIN, and performs **trend analysis** to identify recurring issues. The implementation strategy emphasizes a **start-small approach**, incorporating **operator feedback** and minimizing disruption to existing processes. Looking ahead, the system can be extended into the **supply chain** for verifying supplier parts and implementing **predictive quality** measures to anticipate failures before they occur, further enhancing operational efficiency and reducing manufacturing defects. |
| Stage V: Reflection | A detailed checklist was used to assess the completeness, system functionality, deployment readiness, and user interaction capabilities of the Smart Part Verification System.   Key Evaluation Areas: **System Functionality**: Presence of PIR, IR, and Color sensors; reliable MQTT-based data transmission; effective ML integration for mismatch and defect detection; real-time dashboards; alert mechanisms. **Implementation Readiness**: Compatibility with existing assembly lines; adaptable ML model; basic fallback for cloud failure; stakeholder workflow integration.  **User Interaction**: Dashboard usability for QA teams; actionable mobile alerts; supervisor training. | • The ML system lacks **real-time adaptive learning**, limiting its ability to evolve with new defect patterns. • There's an **over-dependence on continuous cloud connectivity**, risking alert failures during outages. • The system lacks a **local fallback mechanism** for analytics and alerts when connectivity fails. • A **structured feedback loop** from line operators for improving system accuracy is missing. • **Scalability** remains a concern, especially when extending the solution plant-wide or integrating with complete vehicle audit systems. |
| Stage VI:  Design of Modules | Designing the Smart Part Verification System was approached in a modular manner to enable easier debugging, scalability, and hardware-software co-design. Each function—sensing, processing, communication, analytics, and response—was developed as a module.  **Component-Level Design:** • **Sensors**: PIR for movement, IR for proximity, Color sensor for verification • **Microcontroller**: Raspberry Pi Pico W • **Connectivity**: Wi-Fi via Pico W using MQTT protocol  **System-Level Design:** Each sensor module feeds into the Pico W. The Pico processes or forwards data to the cloud via MQTT. The cloud runs ML models, detects anomalies, and returns actionable commands (alerts, logs, interventions).  **Interface Definitions:** • **PIR/IR Sensors** → Digital GPIO inputs to Raspberry Pi Pico W • **Color Sensor** → I2C interface • **Wi-Fi Communication** → Pico W internal Wi-Fi with MQTT protocol over TCP/IP • **Cloud Server** → MQTT Broker (e.g., Mosquitto), connected to backend analytics & dashboards • **Actuation (Alerts)** → GPIO outputs for buzzer/LED fallback alerts  **Standards Used:** • **MQTT Protocol** for lightweight IoT communication • **JSON Payload Format** for structured sensor data • **Mermaid Diagramming Syntax** for visualization • **Wi-Fi IEEE 802.11 b/g/n** for wireless transmission • **I2C/GPIO** electrical interface standards for sensor connections | [BlockDiagram](https://www.mermaidchart.com/app/projects/82200603-389d-45a7-863d-239350f9a0c9/diagrams/f5327afb-e666-4c6d-9b2e-f22b6ce2664d/version/v0.1/edit) |
| Stage VII:  Resources  Identification | Raspberry Pi Pico W  PIR Sensor HC-SR501  IR Sensor  OLED  TCS3200 Color Sensor  Breadboard + Jumper Wires  Power Adapter + USB Cable  Buzzer + LEDs  Logic Level Converter  Miscellaneous Passive Components  Interfacing Software & Tools (WOKWI, IDE, etc.)  Total cost of the resources – ₹1,700 | **Hardware**   * Raspberry Pi Pico W * PIR Sensor * IR Sensor * Colour Sensor * Breadboard + Jumper Wires * Power Supply (Adapter + Cable) * Buzzer and LED Modules * Logic Level Converter * Passive Components   **Software**   * WOKWI Simulator * MQTT Protocol (Mosquitto or built-in) * ThingSpeak (Cloud IoT Dashboard) |
| Stage VIII:  Planning | | **Day** | **Stages Covered** | **Activities** | | --- | --- | --- | | **Day 1** | **I. Sensor Data Acquisition and Cloud Integration (Simulator)** | - Simulate PIR, IR, and Color sensor data using **Wokwi**- Write MicroPython code for Raspberry Pi Pico W- Publish sensor readings to **ThingSpeak** via MQTT- Configure ThingSpeak fields and live channel- Verify cloud dashboard receives updates in real time | | **Day 2** | **II. Hardware Implementation of Sensor Data Acquisition and Cloud Integration** | - Wire sensors to actual Raspberry Pi Pico W- Run tested code on real hardware- Ensure Wi-Fi connection and successful data transmission to ThingSpeak- Validate sensor readings (motion, distance, color) in cloud dashboard | | **Day 3** | **III, IV, V. Fetch Data, Failure Prediction & Predictive Maintenance** | - Fetch data from ThingSpeak using API (via Python or Excel)- Clean and preprocess data for training- Build and test ML model to classify defects- Based on predictions, design logic to schedule predictive maintenance (alerts, logs) | | **Day 4** | **VI. Data Analytics** | - Use ThingSpeak visualizations or export to analyze defect trends- Plot sensor reliability, failure counts, alert times, etc.- Compare simulation vs. real hardware trends- Summarize performance metrics and insights | | **Day 5** | **VII. Version Control & Final Integration** | - Set up and organize a GitHub repo with simulation, hardware, and ML files- Document code using README and flowcharts- Enable final **dashboard UI interface** for ThingSpeak with live updates from hardware- Integrate **LED/Buzzer feedback** mechanism on Pico W for real-time alerts (e.g., turn on buzzer when defect is detected)- Test full pipeline: sensor → cloud → decision → buzzer/dashboard- Final review and prepare team demo presentation | | **Risk**   1. Connectivity Failure – Interruption in Wi-Fi may lead to loss of real-time alerts and data transmission to ThingSpeak. 2. Sensor Malfunction – Hardware sensors like PIR, IR, or Colour sensors may degrade over time or give inaccurate readings due to dust, lighting, or vibration in manufacturing environments.   **Constraints**   1. Limited Processing Power – Raspberry Pi Pico W has limited resources, restricting the complexity of on-device machine learning or real-time analytics. 2. Real-Time Requirements – System must operate with minimal latency to ensure immediate alerts and effective part verification.   **Assumptions**   1. All sensors are pre-calibrated and installed in optimal conditions for accurate detection. 2. Continuous power and internet connectivity are available during the system's operation. 3. The system will be deployed in a controlled industrial environment with minimal interference or unexpected physical disruptions. |
| Stage IX:  Redesigning | * Added buzzer and LED alerts for local notification in case of part mismatch. * Mapped features directly to customer requirements (presence, type, color). * Included manual override button for worker feedback. * Improved ML logic for better mismatch detection. | * Ensured **offline alerting** through buzzer/LED. * System now aligns fully with **customer expectations**. * Enabled **worker feedback** to improve future accuracy. * Enhanced **fault detection** using refined ML pipeline. |
| Stage X:  Execution Framework | **Tools / Focused Modules:**   1. Administrative Dashboard (Power BI / Tableau) → Shows part presence, color mismatch alerts, and sensor activity in real-time. 2. Defect Analytics Module → Visualizes historical trends of part installation errors for root cause analysis. 3. Sensor Health Monitor → Tracks PIR, IR, and Color sensor status to ensure continuous operation. 4. Alert & Notification System → Pushes buzzer alerts (local) and mobile alerts (remote) for mismatched/missing parts. | Power BI/Tableau installed and integrated with MQTT/cloud database.  Dashboard must include part match/mismatch stats, sensor uptime, and fault logs.  Real-time local alerts via buzzer and remote alerts via mobile/ThingSpeak.  Historical analytics for supervisors to identify frequent defect zones or shifts.  Supports manual feedback entry from workers during misdetection. |
| Stage XI:  Micromodules | [**Mindmap**](https://lucid.app/lucidspark/77c5b256-1964-443b-a0b2-87fde13006dc/edit?viewport_loc=-1895%2C-181%2C2210%2C1171%2C0_0&invitationId=inv_bb3ac330-fb00-469f-be93-99790877ce22) |  |