SENTINEL

Progetto di Internet of Things

Simone Costanzo
Giuseppe Pitruzzella

Sommario

Descrizione del progetto

Sentinel propone una soluzione innovativa per affrontare le sfide legate alla manutenzione industriale in tunnel complessi. Partendo dal presupposto che la sicurezza e il benessere degli operatori siano prioritari, il progetto si concentra su tre aspetti principali:

- monitoraggio continuo della posizione e dello stato degli operatori;
- comunicazione efficiente;
- presa di decisioni intelligenti basate sull'analisi dei dati.

La soluzione proposta si basa sull'utilizzo di dispositivi indossabili alimentati a batteria, tecnologie di comunicazione a corto raggio (**BLE**) e a lungo raggio (**LoRa**), e un servizio decisionale basato sull'intelligenza artificiale (**AI**).

I dispositivi indossabili, simili a braccialetti, sono equipaggiati con sensori per monitorare parametri vitali come il battito cardiaco e la temperatura corporea degli operatori. In ottica di progetto, i valori estratti da questi sensori saranno emulati tramite estrazioni periodiche e randomiche dai possibili range.

Inoltre, i braccialetti possiedono una componente di Intelligenza Artificiale, in quanto riescono a prevedere anomalie nei valori registrati per mezzo di un meccanismo basato sul machine learning.

Il gateway funge da punto di raccolta dei dati provenienti dai dispositivi indossabili, inviandoli a un servizio amministrativo esterno tramite una connessione IP. Questo servizio analizza i dati ricevuti, li categorizza e li smista in base ai campi di questi e attiva messaggi speciali di avviso o supporto in tempo reale, migliorando la comunicazione e la sicurezza sul luogo di lavoro.

L'intero sistema è progettato per migliorare la sicurezza e ridurre il rischio di incidenti. Mediante l'utilizzo di tecnologie avanzate e l'analisi intelligente dei dati, **Sentinel** mira ad ottimizzare l'efficienza operativa e a proteggere la salute e il benessere degli operatori industriali.

Ruoli assegnati

Basso livello
Simone Costanzo

Frontend e Backend
Giuseppe Pitruzzella

Componenti

La seguente componentistica permette la concretizzazione del flusso di dati, il tutto permesso dai protocolli di comunicazione utilizzati.

Braccialetto

Il braccialetto, o un qualunque tipo di device portatile, è il cuore della sensoristica di Sentinel. È infatti responsabile della raccolta dei dati vitali dell'operatore all'interno del tunnel, spesso minacciato da condizioni avverse.

La scelta di questi parametri è cruciale per poter studiare il tasso di rischio di un eventuale problema cardiaco. Il braccialetto, per mezzo del machine learning, è infatti in grado di calcolare un valore che indica la probabilità di una fatalità per l'operatore, che verrà poi elaborata dalle anchor.

L'emulazione dei parametri è giustificata dall'effettiva presenza sul mercato dei sensori (compatibili con dispositivi **ESP32**) che li estrapolano.

Anchor

I dispositivi designati come “anchor” offrono diversi servizi. Vengono installati ogni 20 metri all’interno del tunnel e sono responsabili della raccolta dei parametri vitali a partire dai braccialetti; sono utili anche per l’invio in broadcast di questi ultimi e del conseguente indirizzamento verso i gateway esterni.

Gateway

I due gateway, posti alle estremità del tunnel, non sono altro che delle anchor speciali. A differenza di queste, infatti, possono collegarsi ad una Intranet aziendale, che permette l’invio dei dati al broker **MQTT**, tramite il meccanismo di pubblicazione offerto dal protocollo stesso.

Backend

Il server backend ha il compito di iscriversi ad un topic MQTT, per poi riceverne i dati ed elaborarli, controllare eventuali errori e smistarli opportunamente nel Database. Inoltre, aggiorna il frontend tramite una socket sempre attiva.

Frontend

La visualizzazione dei dati sull’interfaccia web è delegata al frontend. Verranno mostrati i dati relativi allo status e alle ultime posizioni rilevate degli operatori, dati delle ancore, ed eventuali notifiche di allerta, oppure di fatalità.

Connettività

I protocolli descritti in seguito garantiscono il funzionamento del flusso di dati rapido, efficace e sicuro.

LoRa

Il protocollo **LoRa** (Long Range) è progettato per la comunicazione a lungo raggio e a bassa potenza per dispositivi IoT. Sentinel sfrutta LoRa per via di una maggiore resistenza alle interferenze e una maggiore copertura rispetto ad altre tecnologie wireless. In questo caso, LoRa utilizza una struttura a topologia lineare, pur mantenendo l'invio di messaggi in broadcast. Ciò è gestito tramite un meccanismo di flooding controllato (scarto il pacchetto se già ricevuto), per cui non satura l'intera banda.

Bluetooth Low Energy

Bluetooth Low Energy (BLE) è una tecnologia wireless progettata per consentire la comunicazione a basso consumo energetico tra dispositivi con bassa potenza di elaborazione e una lunga durata della batteria, come dispositivi IoT, dispositivi indossabili e sensori.

Sentinel sfrutta BLE designando i braccialetti come BLE Server, mentre le anchor saranno inizializzate come BLE Client. La comunicazione avviene tramite scan periodico gestito dalle Anchor, che legge e scrive le strutture dati messe a disposizione dal protocollo per poter garantire un flusso bidirezionale. La comunicazione è principalmente Server -> Client, a meno di un ACK scritto da quest'ultimo nel caso degli Alert.

Wi-Fi

Il protocollo WiFi permette ai dispositivi di comunicare senza fili tramite reti locali (WLAN). I dispositivi cercano reti, si associano con un access point (AP), stabiliscono una connessione, trasmettono dati utilizzando pacchetti e gestiscono la rete per evitare congestioni e garantire sicurezza.

Il protocollo Wi-Fi è impiegato dai Gateway per potersi connettere alla Intranet aziendale ed inviare i pacchetti al Broker MQTT. Inoltre, viene inizializzato un server Wi-Fi anche nelle Anchor, sebbene sia utile soltanto in fase di setup. Questo permette di fornire agli operatori una pagina Captive Portal, così da impostare l'identificativo della Anchor stessa. Appena questa sarà inizializzata, il server Wi-Fi non sarà più utilizzato.

MQTT

MQTT (Message Queuing Telemetry Transport) è un protocollo di messaggistica leggero progettato per dispositivi con limitate risorse di rete e alimentazione, tipicamente utilizzato in soluzioni IoT. Il suo funzionamento si basa su un modello publish-subscribe:

- **Publisher**: I dispositivi che inviano dati (publisher) inviano messaggi a un server MQTT chiamato broker. Questi messaggi sono etichettati con un "topic", una stringa che funge da chiave di indirizzamento.

- **Broker**: Il broker riceve i messaggi dai publisher e li instrada verso i sottoscrittori appropriati in base ai topic a cui sono interessati.

- **Subscriber**: I dispositivi che ricevono dati (subscriber) si connettono al broker e sottoscrivono specifici topic. Quando un messaggio associato a un topic sottoscritto viene ricevuto dal broker, lo inoltra al subscriber corrispondente.

I due Gateway di Sentinel, così come il server Backend, fungono sia da Publisher che da Subscriber. Questo garantisce un flusso bidirezionale di dati, in particolare:

- I gateway ascoltano sul topic ***/sentinel/messages*** e pubblicano sul topic ***/sentinel/ack***.
- Le anchor, al contrario, ascoltano su ***/sentinel/ack*** e pubblicano su ***/sentinel/messages***.

Il broker utilizzato è **EMQX**, una piattaforma open source affidabile.

Implementazione

Verranno qui affrontate le questioni tecniche dei singoli componenti del progetto, con un grado di dettaglio superiore ai paragrafi precedenti.

Anchor

Ogni anchor comincerà il proprio ciclo di vita inizializzando le strutture necessarie per permettere il funzionamento dei protocolli.

Viene eseguito il setup per il protocollo **LoRa**, che permette la comunicazione broadcast dei valori vitali a tutti gli altri dispositivi in ascolto sulla stessa frequenza.

Infine, vengono inizializzate le strutture necessarie al funzionamento della connessione **Bluetooth Low Energy (BLE)**.

Le anchor necessitano di un ulteriore setup prima di poter cominciare a raccogliere dati. In particolare, sarà un operatore, in fase di installazione, a impostare il progressivo della anchor per mezzo di un **Captive Portal** messo a disposizione da quest'ultima se non inizializzata. Il portale permetterà l'inserimento dell'identificativo della anchor, che verrà salvato in memoria permanente tramite la libreria **Preferences**.

Dopo aver completato la configurazione, la anchor verrà riavviata, facendo così partire la seguente routine di operazioni:

- Gestisce i pacchetti LoRa provenienti da altre anchor; i pacchetti vengono rimbalzati in broadcast se sono nuovi, altrimenti scartati; se il pacchetto è di tipo **ACK**, allora scatterà un meccanismo per conservare l indirizzo del destinatario, che riceverà l ACK al ciclo successivo di scan nel caso in cui verrà rilevato nelle vicinanze.
- Esegue la scan degli operatori vicini; per ognuno di essi (identificato dal nome operator"), avverrà una connessione BLE, una lettura delle caratteristiche contenenti i dati vitali e altre informazioni di controllo e una disconnessione. Dopo aver letto i dati, la anchor invierà un messaggio di Log in broadcast.

Se la caratteristica relativa alle anomalie risulta posta a 1”, viene inviata una richiesta di soccorso in broadcast.

Se la caratteristica relativa alla morte dell operatore risulta posta a 1”, viene inviato un messaggio apposito al gateway.

Se l operatore è uno dei destinatari di ACK, la caratteristica di Alert viene posta a 2”, verrà quindi letta dall operatore che riceverà quindi una conferma di ricezione e la reimposterà a 0”, come in origine.

- Invia periodicamente un messaggio di Ping verso i gateway, permettendo alla anchor di rimanere attiva in backend.

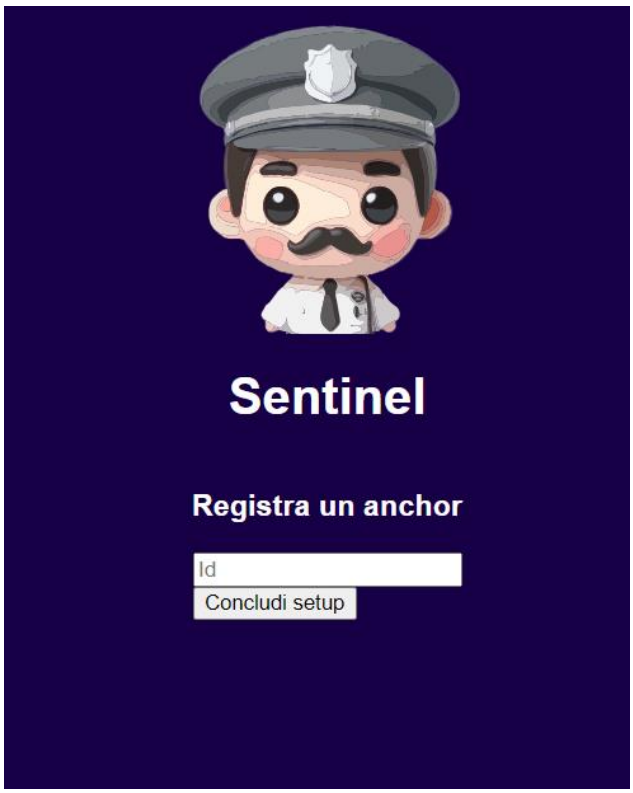

Figura: Captive Portal

Braccialetto

I dispositivi alimentati a batteria fungono da braccialetti indossabili per gli operatori, trasmettendo continuamente dati relativi alla salute e messaggi speciali. I dispositivi utilizzano **BLE** per la comunicazione a corto raggio e **LoRa** per quella a lungo raggio.

Il codice fornito per i dispositivi alimentati a batteria gestisce varie funzionalità, tra cui la comunicazione LoRa e BLE, la scansione degli operatori e la gestione dei messaggi speciali. Garantisce la trasmissione efficiente dei dati relativi alla salute e dei messaggi speciali.
Il codice del braccialetto BLE simula i dati generati dai dispositivi indossabili, trasmettendo valori casuali di BPM e temperatura a intervalli regolari. Questa simulazione è essenziale per testare e convalidare la funzionalità complessiva del sistema. Il codice del braccialetto BLE genera valori casuali di BPM e temperatura, trasmettendo questi dati utilizzando la tecnologia BLE. Questa simulazione aiuta a replicare scenari del mondo reale per scopi di testing e sviluppo.

All'accensione di una nuova ancora, l'operatore (i.e. installatore) dovrà collegarsi al Wi-Fi interno (i.e. operator). A quel punto verrà presentato un captive portal in grado di eseguire il setup della nuova ancora. Rispetto la configurazione del gateway, le operazioni da eseguire affinché questo possa essere inizializzato correttamente sono le seguenti: scaricare VSCode e conseguentemente l'estensione PlatformIO, scaricare lo script relativo al gateway ed aprire quest'ultimo con ciò che è stato installato nel primo punto (i.e. VSCode), modificare i dati relativi al Wi-Fi con i propri dati (e.g. Wi-Fi aziendale). Infine, non rimane che accendere ed indossare il braccialetto, il quale effettua una connessione con le ancore ed invia - autonomamente o manualmente - richieste alle ancore più vicine contenenti i propri parametri vitali. Quest'ultime si passeranno tra loro il messaggio in questione, senza mai perdere o ripetere quest'ultimo. Una volta arrivato al Gateway, questo viene spedito al Server. Insieme ai parametri vitali, chiaramente, viene inoltrata la posizione dell'operazione da cui è stata generata la richiesta; quest'ultima tale da

Gateway

Il gateway svolge un ruolo centrale nel ricevere dati dai dispositivi alimentati a batteria e nel trasferirli a un broker MQTT esterno su una rete IP. Serve da interfaccia tra i dispositivi di monitoraggio locali e
l infrastruttura di rete più ampia. Il codice del gateway coinvolge la comunicazione LoRa per ricevere messaggi dai braccialetti e la

comunicazione MQTT per inoltrare i dati a un broker esterno. Assicura un flusso senza intoppi delle informazioni tra i dispositivi di monitoraggio locali e il servizio amministrativo esterno. Rispetto l'utilizzo di MQTT, vengono utilizzate di ***/sentinel/messages*** in invio e ***/sentinel/acks*** in ascolto.

L'unico setup necessario per il gateway riguarderà l'SSID e la password della intranet aziendale.

Backend

Il codice del backend è sviluppato utilizzando Flask, un framework leggero per applicazioni web in Python.

Flask è un micro-framework web leggero per Python che consente di creare applicazioni web complesse attraverso uno sviluppo semplice, non a caso il principio su cui si è basato il suo ideatore - ovvero, Armin Ronacher - per lo sviluppo del framework stesso è "less is more". Flask, infatti, offre esclusivamente le funzionalità essenziali per la creazione di un'applicazione web, escludendo funzionalità che possono risultare avanzate e complesse.

Flask supporta il pattern architetturale Model-View-Controller (MVC) ed offre la possibilità di gestire il routing per le richieste HTTP, i cookie, le sessioni e l'utilizzo di template. Inoltre, attraverso Flask possono essere utilizzate estensioni dello stesso utili all'aggiunta di particolari feature per la nostra applicazione web. Alcune di esse sono strettamente relative all'implementazione di autenticazione dell'utenza, gestione di e-mail ed utilizzo di database. Quest'ultimo - ovvero, Flask SQL-Alchemy - è presente ed utilizzato all'interno del progetto e viene discusso all'interno dell'apposito paragrafo.

Questo backend gestisce la persistenza dei dati e comunica con il frontend React per fornire informazioni in tempo reale provenienti da un sistema IoT. Il backend inizia importando le li- brerie necessarie, tra cui Flask stesso per la creazione dell applicazione web, SQLAlchemy per interagire con il database, Flask-MQTT per gestire la comunicazione MQTT, e altre librerie di utilità. L uso di CORS è abilitato per consentire le

richieste da origini diverse. Flask è inizializzato, e vengono configurati parametri come l URL del database da utilizzare. SQLAlchemy è configurato per interagire con il database, che viene inizializzato e pronto per l uso. Una funzione di utilità mqtt_create_log viene definita per semplificare la creazione di nuovi log all interno del database in base ai dati ricevuti tramite MQTT. La funzione prende i dati come parametro e li utilizza per creare e aggiungere un nuovo log attraverso SQLAlchemy. Il backend si connette a un broker MQTT tramite Flask-MQTT e quando si verifica una connessione riuscita, si sottoscrive al topic /sentinel/messages . I messaggi ricevuti su questo topic vengono gestiti dalla funzione *handle_mqtt_message*, che decodifica il payload e utilizza la funzione create log per inserire i dati nel database. Sono definite tre classi di modelli SQLAlchemy: User e Log. Ognuna di queste rappresenta una tabella nel database, con attributi specifici e relazioni definite tra di loro.

Al tempo stesso viene utilizzato Flask-SocketIO, implementato per l'utilizzo di SocketIO in Flask e successivamente - nel Frontend - in React. Rispetto SocketIO, vengono definiti diversi *endpoint* per gestire le richieste. Quest'ultime sono get_logs, get_notify e get_anchors, rispettivamente per richiamare nuovi logs, notifiche ed ancore. Quest'ultime vengono richiamate ogni volta che vi è un nuovo messaggio MQTT. Ad ogni route, segue l'emit di un nuovo evento, sul quale sarà in ascolto il client.

Il Backend è configurato per eseguirsi sulla porta 4000 e per accettare connessioni da qualsiasi indirizzo IP (0.0.0.0). Quando il file viene eseguito attraverso Docker, l app Flask viene avviata. In sintesi, questo Backend Flask è progettato per gestire la persistenza dei dati provenienti da un sistema IoT, consentendo al Frontend React di visualizzare informazioni in tempo reale sui dipendenti ed i loro log. La comunicazione MQTT permette la trasmissione efficiente dei dati da dispositivi al backend, mentre l'utilizzo di SocketIO permette l'inoltro efficiente dei dati al client.

Frontend

La parte Frontend relativa al progetto si basa su una applicazione NextJS, in cui vengono utilizzate le dipendenze SocketIO-Client ed Axios.

Il progetto qui presentato è un applicazione web sviluppata in React e Type- Script, focalizzata sulla gestione e la visualizzazione di dati provenienti da un sistema IoT.

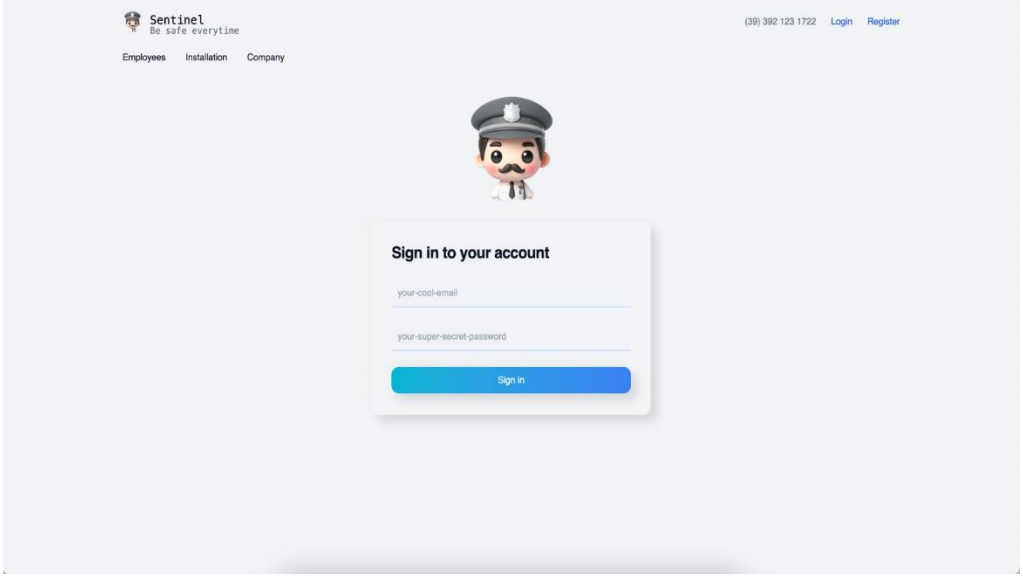

Figura: Interfaccia di login

React è una libreria JavaScript open-source sviluppata da Facebook, ampiamente utilizzata per la costruzione di interfacce utente dinamiche e reattive nelle applicazioni web. Fondata su un paradigma di programmazione dichiarativo, React semplifica la creazione di componenti riutilizzabili che gestiscono in modo efficiente lo stato e il rendering delle interfacce.

Nello specifico, l'applicazione prevede di *renderizzare* un componente *index.tsx* all'interno di un componente SocketProvider, responsabile

dell'inizializzazione di SocketIO verso il Backend. Successivamente, dalla schermata di login può essere effettuato l'accesso o registrazione - raggiungendo la relativa pagina in grado di *renderizzare* il componente *register.tsx* - per l'utente (o admin), per cui viene effettuata una POST rispettivamente alle route ***api/login*** o ***api/register***. Alle route

strettamente legate a quest'ultime operazioni, vengono descritte anche ***/api/@me*** e ***/logout***, relative a verificare che un utente si sia *loggato* e ad effettuare il *logout* di questo.

Effettuato il login, viene ***renderizzato*** il componente *Dashboard*, il quale prevede al suo interno tre ulteriori componenti: **Log**, **Anchor** e **Notify**, rispettivamente per ritornare una tabella da popolare con i rispettivi dati. In particolare modo, i dati vengono ricevuti attraverso l'ascolto dello specifico evento SocketIO nel Backend.

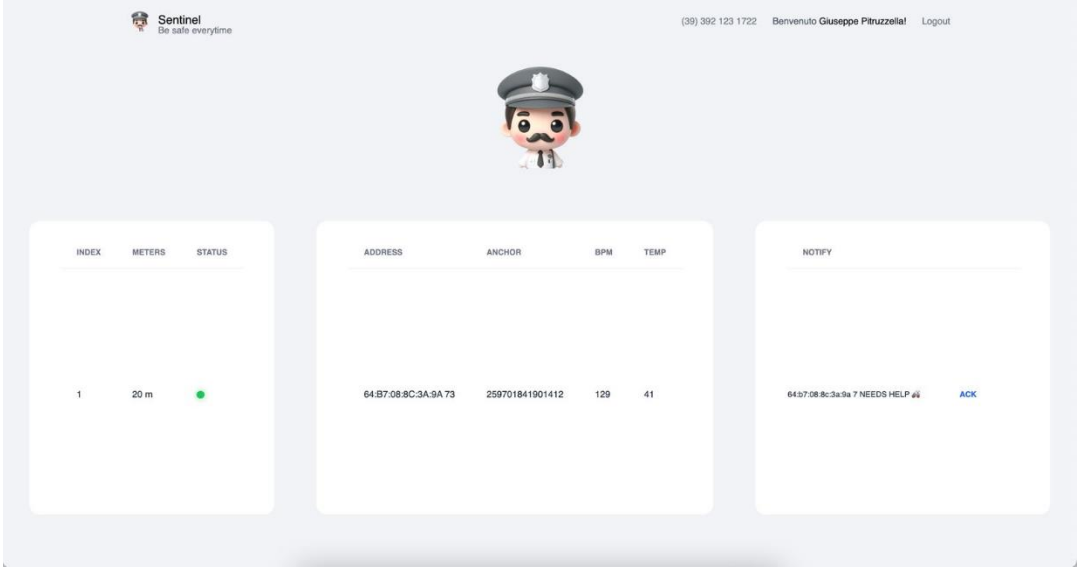

Figura: Interfaccia utente con dati e notifiche

Conclusioni

Sentinel rappresenta una soluzione innovativa e promettente per migliorare la sicurezza e l'efficienza nelle operazioni di manutenzione industriale nei tunnel complessi. Attraverso l'utilizzo di braccialetti portatili, tecnologie di comunicazione avanzate e un servizio decisionale basato sull'intelligenza artificiale, siamo stati in grado di implementare un sistema completo per il monitoraggio degli operatori e la gestione delle emergenze.

L'integrazione delle tecnologie BLE e LoRa ha permesso una comunicazione affidabile e a lungo raggio, consentendo il monitoraggio in tempo reale degli operatori anche in ambienti sotterranei o ad alta densità strutturale.

Tuttavia, ci sono ancora alcune aree che richiedono ulteriori miglioramenti e sviluppi. Ad esempio, potrebbe essere necessario raffinare gli algoritmi di intelligenza artificiale per una maggiore precisione nelle analisi dei dati.

In conclusione, il progetto ha rappresentato un passo significativo verso l'implementazione di soluzioni intelligenti per la manutenzione industriale, con il potenziale di migliorare notevolmente la sicurezza e l'efficienza delle operazioni in tunnel complessi. Siamo fiduciosi che con ulteriori sviluppi e iterazioni, questa soluzione possa diventare un elemento chiave nelle pratiche di manutenzione industriali del futuro.