Proposta Técnica: Sistema de Manutenção Preditiva

Enterprise Challenge - Fase 3

Integrantes:

Tayná Steves - rm562491
João Vittor Fontes - rm565999
Endew Alves dos Santos - rm563646
Carlos Eduardo de Souza - rm566487
Vinícius Divino dos Santos - rm566269

Instituição: FIAP
Curso: Inteligência Artificial
Data: 08 de maio de 2025

Índice

6. **Arquitetura do Sistema de Previsão de Falhas**
7. **Captura de Tela da Divisão de Tarefas**

Justificativa e Camada de Coleta

Justificativa do Problema

Falhas inesperadas em máquinas automotivas causam paradas significativas na produção, resultando em perdas financeiras e operacionais. Por exemplo, uma máquina que produz 1 motor por hora deixa de produzir 24 motores em 24 horas de inatividade, impactando a eficiência da linha de montagem. A solução proposta é uma plataforma de manutenção preditiva que utiliza dados de sensores IoT e Machine Learning para prever falhas, permitindo manutenções preventivas programadas em dias não produtivos (ex.: domingos). Isso reduz custos de reparos emergenciais e aumenta a confiabilidade da produção.

Objetivo do Projeto

Desenvolver um sistema que monitore máquinas em tempo real, detectando padrões de falhas e gerando alertas preditivos, objetivando reduzir as paradas não planejadas em 100%

Contribuição para a Empresa

O sistema otimizará a produção ao evitar interrupções, economizando recursos com manutenções preventivas e melhorando a competitividade da empresa ao garantir maior tempo de produção das linhas de montagem. A análise preditiva baseada em dados de sensores (ex.: temperatura, vibração) permite decisões estratégicas, como agendamento eficiente de reparos.

Camada de Coleta - Sensores e Coleta de Dados

Dispositivo de Coleta

Os dados serão coletados por um microcontrolador **ESP32**, um dispositivo IoT com conectividade Wi-Fi, escolhido por sua acessibilidade, suporte a múltiplos sensores e ampla adoção em projetos industriais. O ESP32 captura leituras de sensores analógicos e digitais, processando-as para envio ao sistema.

Processo de Coleta

Os sensores, conectados ao ESP32, capturam leituras a intervalos regulares (ex.: a cada 10 segundos para temperatura, 2 segundos para vibração, dependendo da criticidade). O ESP32 processa os dados e os envia via Wi-Fi usando o protocolo **MQTT** (Message Queuing Telemetry Transport) para um broker local, como o **Mosquitto**. O broker encaminha os dados a um script Python, que os insere no **Oracle Database** para armazenamento e análise. Esse processo garante coleta em tempo real, com modularidade para adicionar novos sensores sem alterar o sistema.

Confiabilidade dos Dados Coletados

A confiabilidade será garantida por:

- **Calibração inicial**: Sensores são ajustados antes da instalação para leituras precisas.
- **Validação no ESP32**: O microcontrolador verifica valores inconsistentes (ex.: temperatura > 100°C ou vibração fora da faixa 1-10 Hz) e descarta leituras inválidas antes do envio.
- **Verificação no sistema**: Um script Python compara novos dados com faixas aceitáveis (baseadas em históricos) e sinaliza anomalias para revisão.
- **Manutenção periódica**: Sensores serão recalibrados regularmente para evitar desvios.

Essas medidas asseguram que os dados sejam precisos e confiáveis, fundamentais para a análise preditiva do sistema.

Camada de Comunicação - Transmissão de Dados

Como os dados saem dos sensores e chegam até o sistema:

Os dados serão coletados por sensores conectados a um microcontrolador ESP32, que atuará como dispositivo IoT. O ESP32 processará as leituras dos sensores e as envia via Wi-Fi usando o protocolo MQTT (há uma explicação sobre o protocolo MQTT abaixo), escolhido por sua eficiência em redes IoT pois ele tem baixo consumo de banda e suporte a comunicação em tempo real. O ESP32 publicará os dados em um tópico específico em um broker MQTT local. Um script Python, configurado como assinante, monitora o tópico, recebe os dados em tempo real e os insere em um banco de dados Oracle, para armazenamento e posteriormente, para análise. Esse fluxo garante modularidade, pois cada parte do sistema funciona de forma independente, podendo ser trocada, atualizada ou ampliada e escalabilidade, pois ele pode crescer pois será possível adicionar mais sensores, dispositivos ou assinantes sem perda de desempenho.

Em relação ao serviço que irá gerenciar a troca de mensagens, Inicialmente usaremos o Mosquitto, um broker MQTT local, para gerenciar a troca de mensagens entre o ESP32 e o sistema (script Python). Ele receberá dados dos sensores e os distribuirá aos assinantes, ele será ideal para testes. Futuramente, iremos migrar para o serviço em nuvem AWS IoT Core, que oferece escalabilidade e um gerenciamento seguro de mensagens.

Protocolo MQTT:

A função do protocolo MQTT é possibilitar a comunicação eficiente entre dispositivos IoT e sistemas. é leve, eficiente e ideal para dispositivos IoT como o ESP32. Ele consome pouca energia e banda, suporta comunicação em tempo real, funciona bem em redes instáveis e é escalável para múltiplos sensores em uma fábrica.

Componentes principais do protocolo MQTT:

1. **Publicador**: Um dispositivo como um microcontrolador coleta dados e os envia para um tópico
2. **Tópico**: Um tópico é uma caixa de correio virtual onde vários sistemas têm acesso aos dados que estão ali. Ele é identificado por um nome, como por exemplo: “fabrica/linha1/temperatura”
3. **Broker**: Direciona as mensagens direcionadas aos programas inscritos no tópico.
4. **Assinante**: É um sistema ou programa que se inscreve em um tópico para receber os dados enviados ao broker.

Funcionamento:

1. O microcontrolador, conectado a sensores, lê dados, e usa o protocolo MQTT para publicá-los em um tópico.
2. O broker recebe a mensagem e verifica quais assinantes estão inscritos no tópico, e então encaminha os dados aos assinantes em tempo real.
3. O script python configurado como assinante recebe os dados e os processa.

Características Importantes:

1. Bidirecionamento: Além de enviar dados, o microcontrolador (no nosso caso o ESP32) pode receber comandos do broker. isso pode ser necessário por exemplo se precisarmos aumenta a frequência da leitura de dados
2. Escalabilidade: suporta milhares de dispositivos
3. Resiliência: Se a conexão falhar o ESP32 pode armazenas dados em uma memória temporária (buffer) e reenvia eles quando reconectado. Isso evita a perda de dados de sensores, garantindo que o sistema de manutenção preditiva receba todas as leituras para análise.

Segurança na comunicação dos dados

A segurança na comunicação dos dados será feita através de uma autenticação básica com usuário e senha no broker MQTT. Onde apenas dispositivos autorizados irão poder publicar dados em tópicos, e apenas assinantes autorizados poderão recebê-los.

Essa medida de segurança seria útil e necessária pois sem ela, seria possível um invasor enviar dados falsos, não permitindo o sistema de manutenção preditiva funcionar corretamente, além de evitar ataques que iriam sobrecarregar o broker com mensagens ou o vazamento de dados.

Camada de Processamento – Análise e Armazenamento

Armazenamento:

Os dados serão armazenados no Banco de Dados Oracle, dentre os motivos para escolhermos eles esta a sua confiabilidade, seu desempenho, sua escalabilidade e sua segurança.

Sua confiabilidade pois é amplamente usado em aplicações da indústria; seu desempenho pois oferece alta performance em consultas complexas, ideal para análises preditivas em tempo real que é o caso do nosso projeto; recursos avançados sua escalabilidade pois permite a migração para ambientes em nuvem, e sua segurança pois inclui recursos como autenticação e auditoria, protegendo assim dados sensíveis.

Os dados serão organizados em um banco de dados relacional, usando tabelas estruturadas. Cada leitura de sensorserá armazenada em uma tabela "sensores" com colunas: timestamp (data/hora da leitura), sensor_id (identificador do sensor), tipo (ex.: temperatura), valor (ex.: 80°C), unidade (em °C), e status (normal/anômalo). As tabelas seguirão normalização para evitar redundâncias, com índices para consultas rápidas, garantindo organização eficiente para análise preditiva.

Técnicas de IA que serão aplicadas

As técnicas aplicadas serão random forest, análise de séries temporais com ARIMA e detecção de anomalias com isolation forest. Essas técnicas são robustas, fáceis de implementar em protótipos e adequadas para dados de sensores IoT.

- Random Foreste pois é um algoritmo de aprendizado supervisionado que classifica padrões de falhas com alta precisão, usando bibliotecas como Scikit-learn. Ideal para correlacionar dados de sensores com eventos de falha.
- Análise de Séries Temporais com ARIMA: Para prever tendências em dados de sensores (ex.: aumento gradual de temperatura), usando a biblioteca Statsmodels. É eficaz para detectar comportamentos que precedem falhas.
- Detecção de anomalias com Isolation Forest: Identifica leituras anômalas (ex.: picos de corrente elétrica) em tempo real, também via Scikit-learn, permitindo alertas imediatos

Todos os modelos serão desenvolvidos em Python, utilizando as bibliotecas Scikit-learn, PyCaret e Pandas.

- **Scikit-learn**: Ideal para implementar modelos de Machine Learning como Random Forest e Isolation Forest. É bem documentada e perfeita para classificar padrões de falhas. Neste projeto essa biblioteca será usada para treinar modelos que irão prever falhas com base nos dados fornecidos pelos sensores.

- **PyCaret**: Simplifica a construção e comparação de modelos de Machine Learning com menos código. É útil para protótipos rápidos, permitindo testar vários algoritmos e selecionar o melhor automaticamente. Neste projeto ela será utilizada para agilizar o desenvolvimento e validação de modelos preditivos.
- **Pandas**: Essencial para manipulação e pré-processamento de dados. Será usada neste projeto para organizar dados de sensores recebidos via MQTT para análise.

Processamento

O processamento será feito localmente, em um computador executando scripts Python que utilizam bibliotecas como Scikit-learn e Pandas. Os dados de sensores, recebidos via MQTT e armazenados no Oracle Database, serão analisados localmente para treinar e executar modelos de IA. Essa abordagem reduz custos e é adequada para o protótipo (v1.0). Futuramente, o processamento irá migrar para uma instância do Amazon EC2 (AWS) na nuvem para maior escalabilidade.

Como o sistema será treinado

O sistema aprenderá a prever falhas nas máquinas por meio de aprendizado supervisionado, uma abordagem de Machine Learning onde um modelo é treinado com dados rotulados para fazer previsões.

1. Coleta e Preparação de Dados:

O sistema começará com um conjunto de dados históricos reais ou simulados, esses dados incluem entradas e saídas. As entradas (features) serão as leituras dos sensores coletadas via ESP32, já as saídas (rótulos) serão registros de eventos de falhas, indicando quando a máquina parou, e se possível também o tipo de falha (exemplo.: falha por superaquecimento).

Os dados são organizados em uma tabela no Oracle Database, com as colunas: timestamp, sensor_id, tipo, valor, e status (normal ou anômalo).

A biblioteca “Pandas” será usada para o pré-processamento e para a engenharia das features.

- pré-processametno: Irá remover valores nulos, corrigir outliers, e normalizar dados para escalas consistentes.
- Engenharia de features: Irá criar variáveis derivadas, como médias móveis de vibração, ou taxas de variação de temperatura, para capturar as tendências.

2. Treinamento do modelo

O sistema utiliza aprendizado supervisionado, onde o modelo aprende a associar padrões nos dados de sensores (features) com eventos de falha (rótulos). As técnicas de IA escolhidas,

como já mencionadas antes, incluem: Random Forest (via Scikit-learn), ARIMA (via Statsmodels) e Isolation Forest (via Scikit-learn)

3. Processo de Treinamento

O treinamento ocorre em um computador local, usando scripts Python. o processo inclui divisão dos dados, ajuste de modelo, e validação:

- Divisão de dados: O conjunto histórico é dividido em 80% para treinamento e 20% para teste, garantindo que o modelo generalize bem para novos dados.
- **Ajuste do modelo**: O Random Forest, será treinado para mapear combinações de features (ex.: vibração alta + corrente instável) a rótulos de falha.
- **Validação**: O modelo é testado com dados de teste para avaliar sua precisão (ex.: 92% de acertos em prever falhas). Métricas como acurácia, precisão e recall são analisadas.

4. Previsão de Falhas

Após o treinamento, o modelo é aplicado a novos dados de sensores em tempo real, recebidos via MQTT e armazenados no Oracle Database. O processo é:

- **Entrada de dados**: Um script Python lê as últimas leituras do banco.
- **Previsão**: O modelo Random Forest calcula a probabilidade de falha. O Isolation Forest verifica anomalias imediatas.
- **Saída**: Alertas são gerados e exibidos em um dashboard, informando a equipe de manutenção para agir preventivamente.

5. Melhoria Contínua

O sistema é projetado para evoluir:

- **Atualização do modelo**: Novos dados de sensores e falhas reais (quando disponíveis) são incorporados para retreinar o modelo, melhorando sua precisão.
- **Aprendizado contínuo**: Futuramente, na versão 2.0 (em AWS EC2), o sistema pode adotar técnicas de aprendizado online para se adaptar a novos padrões sem retreinamento completo.
- **Feedback**: Resultados de manutenções (ex.: "falha prevista confirmada") são usados para refinar os rótulos e features.

Camada de Aplicação – Visualização e Interface

Visualização e Interface:

A camada de aplicação é responsável pela interface com os usuários. Ela apresenta os dados processados de maneira clara e acessível por meio de dashboards desenvolvidos no Power BI. O objetivo é possibilitar que operadores e gestores visualizem o status atual das máquinas, identifiquem tendências, consultem históricos e tomem decisões com base em dados.

Tipos de Dashboards:

Dashboard Operacional: voltado para os operadores da linha de produção, apresenta os dados em tempo real, como temperatura, pressão, status de funcionamento e alertas de anomalias. É uma interface simples e visual, com indicadores de cores (verde, amarelo, vermelho), seguindo o padrão semafórico industrial para facilitar a leitura rápida.

Dashboard Gerencial: voltado para supervisores e gestores. Mostra análises históricas, KPIs de desempenho, quantidade de falhas evitadas, tempo de resposta a alertas, entre outros indicadores. Permite acompanhar a evolução da saúde dos equipamentos ao longo do tempo.

Atualização e Integração:

Os dashboards se atualizam automaticamente com base nos dados armazenados no banco Oracle. A atualização será feita a cada minuto no protótipo, mas no ambiente produtivo poderá ser contínua ou disparada por eventos críticos. A conexão entre Power BI e o banco é feita via consulta direta (ODBC). Futuramente, a aplicação pode ser migrada para a web ou para uma plataforma como Power BI Service, permitindo acesso remoto e em tempo real por qualquer dispositivo.

Experiência do Usuário (UX):

A interface será pensada para ser intuitiva, com ícones, gráficos de linha, barras e medidores visuais. As cores são usadas para indicar rapidamente o status das variáveis. Há destaque para alertas, com botões de confirmação de leitura, possibilidade de exportar relatórios e foco em acessibilidade, como contraste de cores e ícones descritivos.

Camada de Ação – Alertas e Respostas

A última camada do sistema HERMIA é responsável por reagir aos eventos identificados como anormais. Quando uma anomalia é detectada pelo modelo de IA, ela aciona o sistema de alertas, que notifica responsáveis através de e-mail ou WhatsApp. A meta é garantir que as informações cheguem de forma rápida e eficaz à equipe de manutenção.

Geração de Alertas:

Quando uma leitura ultrapassa o limite esperado, o sistema envia um alerta com os dados sobre a variável afetada, o valor atual, o horário da leitura, o grau de severidade (ex: leve, moderado, crítico) e ação recomendada.

Canal de Notificação:

O sistema usa APIs de e-mail e WhatsApp para envio dos alertas. Os alertas podem ser configurados para diferentes níveis de prioridade, com destinatários diferentes.

Recomendação de Ações:

Junto ao alerta, o sistema recomenda uma ou mais ações com base no tipo de falha:

- Reduzir a carga da máquina
- Agendar manutenção
- Parar o equipamento

Rastreabilidade:

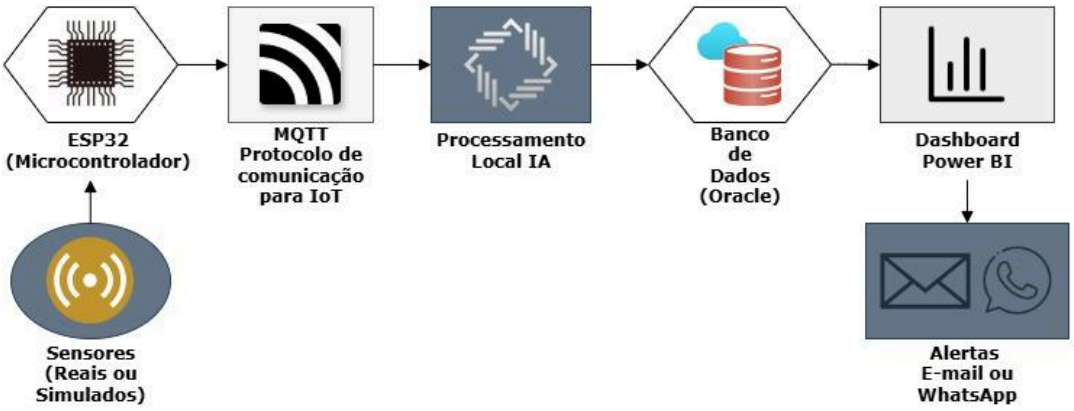
Toda a ação sugerida e executada é registrada no banco de dados, esses registros permitem analisar o histórico de respostas a falhas e aprimorar as recomendações futuras.

Melhorias Futuras:

- Integração com sistemas de automação industrial (ex: PLCs) para desligamento automático
- Sistema de feedback da equipe para avaliar a eficácia das recomendações

Com essa camada, o ciclo do sistema se fecha: da coleta de dados até a tomada de decisão baseada em IA, garantindo que o HERMIA não apenas detecte falhas, mas também contribua ativamente para evitá-las.

Arquitetura do Sistema de Previsão de Falhas



Captura de Tela da Divisão de Tarefas