FACULDADE DE TECNOLOGIA DE TAUBATÉ

GILMAR APARECIDO SANTOS COSTA

JÚLIO CÉSAR VALIM

MARCO AURÉLIO ALVES

RAFAEL PIRES TOSETTO

RODRIGO CARDOSO MUNIZ

SISTEMA DE TELEMETRIA PARA MONITORAMENTO E SEGURANÇA EM EQUIPAMENTOS DE MOVIMENTAÇÃO

TAUBATÉ

2025

FACULDADE DE TECNOLOGIA DE TAUBATÉ

GILMAR APARECIDO SANTOS COSTA

JÚLIO CÉSAR VALIM

MARCO AURÉLIO ALVES

RAFAEL PIRES TOSETTO

RODRIGO CARDOSO MUNIZ

SISTEMA DE TELEMETRIA PARA MONITORAMENTO E SEGURANÇA EM EQUIPAMENTOS DE MOVIMENTAÇÃO

Trabalho de Graduação apresentado à Coordenação do Curso Superior de Tecnologia em Eletrônica Automotiva do Centro Estadual de Educação Tecnológica Paula Souza para a obtenção da aprovação do Projeto de Trabalho de Graduação.

Orientadora: Prof.ª Me. Tamy Fernandes Pereira

Orientador: Prof.º Cristovão Guimarães Miranda

TAUBATÉ

2025

GILMAR APARECIDO SANTOS COSTA

JÚLIO CÉSAR VALIM

MARCO AURÉLIO ALVES

RAFAEL PIRES TOSETTO

RODRIGO CARDOSO MUNIZ

SISTEMA DE TELEMETRIA PARA MONITORAMENTO E SEGURANÇA EM EQUIPAMENTOS DE MOVIMENTAÇÃO

Trabalho de Graduação apresentado a Faculdade de Tecnologia de Taubaté, como parte das exigências para a aprovação do Projeto de Trabalho de Graduação.

Orientadora: Prof.ª Me. Tamy Fernandes Pereira

Orientador: Prof.º Cristovão Guimarães Miranda

Taubaté, 02 de Dezembro de 2025.

BANCA EXAMINADORA

_____

Prof. Clayton Koba

FATEC Taubaté

_____

Prof. Flávio Groh

FATEC Taubaté

_____

Prof. Cristóvão Guimarães Miranda

FATEC Taubaté

Dedico este trabalho ao Prof. MSc. Claudio Alberto Langui pelo apoio, incentivo e empenho incansável em busca da excelência de seus alunos. Sempre buscando oportunidades para demonstrar que a especialização não se constrói apenas com os livros, mas também pela paixão de colocá-los em prática

AGRADECIMENTOS

A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho. Aos nossos pais, esposas e companheiras que proporcionaram toda condição necessária para a conclusão desta faculdade, e amigos, pela amizade.
Aos professores envolvidos, pelo inestimável apoio na orientação deste trabalho.
Aos professores pelos ensinamentos e dedicação ímpar em nos ensinar, aos colegas de classe, com quem nesses anos de estudo tive a felicidade de conviver.

Não se deve ir atrás de objetivos fáceis, é preciso buscar o que só pode ser alcançado por meio dos maiores esforços.

.

(Albert Einstein)

RESUMO

O aumento da procura por soluções sustentáveis e seguras na indústria está impulsionando a utilização da telemetria em equipamentos de movimentação. Este estudo apresenta um sistema de telemetria fundamentado no ESP32, incorporado à plataforma Arduino IoT Cloud, visando o acompanhamento de empilhadeiras. O sistema reúne informações como temperatura do motor, inclinação, nível de bateria, presença e velocidade, permitindo medidas preventivas, diminuição de falhas e aumento da eficiência energética. Além de aprimorar as operações, sua contribuição para a sustentabilidade é notável, pois diminui paradas não planejadas, consumo de recursos e impacto ambiental. O estudo comprova que a proposta é uma solução tecnológica inteligente e acessível para o setor logístico.

Palavras-chave: Sistemas Embarcados, Indústria 4.0, Sustentabilidade, Gestão de Frota.

ABSTRACT

The growing demand for sustainable and safe solutions in industry is driving the use of telemetry in handling equipment. This study presents a telemetry system based on ESP32, integrated into the Arduino IoT Cloud platform, for monitoring forklifts. The system gathers information such as motor temperature, inclination, battery level, presence, and speed, enabling preventive measures, reducing failures, and increasing energy efficiency. In addition to improving operations, its contribution to sustainability is notable, as it reduces unplanned downtime, resource consumption, and environmental impact. The study proves that the proposal is a smart and affordable technological solution for the logistics sector.

Keywords: Embedded Systems, Industry 4.0, Sustainability, Fleet Management.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ADC	Conversor Analógico-Digital
AD	Conversor Digital-Analógico
API	Interface de Programação de Aplicações
CAN	*Controller* Area *Network*
C++	Linguagem de Programação
DAC	Digital-*to-Analog* Converter
DISPATCH	Despachar
ERP	Sistema de Planejamento de Recursos
ESP32	*Espressif Systems 32 bits*
GPIOs	General-*Purpose Input-Output*
GPRS	Serviço Geral de Pacotes por Rádio
GPS	Sistema de Posicionamento Global
GSM	Sistema Global para Comunicações Móveis
HTTP	Protocolo de Transferência de Hipertexto
IA	Inteligência Artificial
I2C	*Inter-Integrated Circuit*
INT	Telemetria de Rede em Banda
IoT	Internet das Coisas
IHM	Interface Homem Maquina
LoRa	Comunicação de Longo Alcance
LTE	Evolução de Longo Prazo (Tecnologia de transmissão de dados móveis)
LAN	Rede Local
MQTT	Protocolo de Transporte de Telemetria por Fila de Mensagens
ODS	Objetivos de Desenvolvimento Sustentável
ONU	Organização das Nações Unidas
OBD	Diagnóstico de Bordo
P4	Programação de Processadores de Pacotes Independentes de Protocolo
PWM	*Pulse Width Modulation*
RFID	Identificação por Rádio Frequência

SCADA	Supervisão, Controle e Aquisição de Dados
SDN	Objetivando da Telemetria *In-Band*
CSV	Comma Seoarated Values
TCP/IP	Protocolo de Controle de Transmissão / Protocolo da Internet
UART	Receptor-Transmissor Assíncrono Universal

SUMÁRIO

1 INTRODUÇÃO

A crescente complexidade das operações industriais, aliada à demanda por segurança e sustentabilidade, impulsiona o uso de tecnologias de monitoramento avançado. A telemetria, nesse contexto, surge uma solução estratégica para o controle em tempo real de equipamentos de movimentação, como empilhadeiras. Sua capacidade de coletar, transmitir e analisar dados operacionais permite não apenas antecipar falhas e reduzir paradas não planejadas, mas também garantir a integridade dos equipamentos e a segurança dos operadores.

Ao integrar sensores e sistemas inteligentes, a telemetria viabiliza práticas de manutenção preditiva e gestão energética mais eficiente. Tais ações, além de elevarem a produtividade, alinham-se aos princípios da sustentabilidade, ao contribuírem para a redução do consumo de insumos e da geração de resíduos industriais.

Dessa forma, a aplicação da telemetria configura-se como um pilar tecnológico indispensável à modernização da indústria, promovendo operações mais seguras, econômicas e ambientalmente responsáveis.

Como um sistema de telemetria pode auxiliar as funções logísticas de forma que os equipamentos de movimentação de materiais se tornem mais sustentáveis?

1.1 Objetivo

Desenvolver um sistema de telemetria para empilhadeiras, visando aprimorar a segurança, eficiência e sustentabilidade das operações logísticas.

Objetivos Específicos

- Identificar variáveis operacionais relevantes (temperatura, velocidade, inclinação, presença e bateria).
- Utilizar sensores e microcontroladores de baixo custo para coleta e transmissão dos dados.
- Avaliar a viabilidade de integração com plataformas em nuvem (IoT).

1.2 Justificativa

A justificativa para o desenvolvimento deste projeto está ancorada na necessidade de soluções tecnológicas que respondam simultaneamente aos desafios de produtividade, segurança operacional e sustentabilidade ambiental no setor industrial. A telemetria, enquanto tecnologia de monitoramento remoto, tem se consolidado como um recurso fundamental para a gestão de equipamentos críticos, como empilhadeiras, ao permitir a coleta e análise contínua de dados operacionais.

A proposta deste trabalho está alinhada aos pilares da Indústria 4.0, especialmente pela adoção de dispositivos conectados (IoT) Internet das Coisas e pela ênfase na automação inteligente. Além disso, o projeto contribui para o cumprimento das metas dos Objetivos de Desenvolvimento Sustentável (ODS) 8, 9 e 12, sobretudo aqueles voltados à modernização da indústria, uso responsável de recursos e melhoria das condições de trabalho.

Ao utilizar plataformas de hardware e software acessíveis, a solução apresentada se torna especialmente relevante para pequenas e médias empresas, promovendo a democratização do acesso à tecnologia e incentivando práticas sustentáveis no ambiente fabril.

1.3 Aspecto Metodológico

Esta subseção descreve o método de pesquisa que será utilizado para o desenvolvimento deste trabalho, que pode ser classificado como de natureza aplicada, com objetivo exploratório e uma abordagem quantitativa. Para o alcance dos objetivos propostos, serão utilizados procedimentos que incluirão revisão teórica e modelagem e simulação para a coleta de dados.

1.4 Aporte Teórico

Para avaliar a relevância do tema, no dia 24/03/2025 foram pesquisados os artigos publicados na base de dados Google Acadêmico. A busca foi realizada no idioma Português, considerando os termos leitura do resumo e aderência ao tema,

exclusivamente no título dos artigos indexados entre os anos 2019 a 2025. Esta busca obteve 119 resultados, sendo classificados por ordem de relevância, e escolhidas as 50 pesquisas mais pertinentes para este estudo após a leitura dos respectivos resumos.

1 FUNDAMENTAÇÃO TEÓRICA

Esta seção irá descrever os aspectos relacionados à telemetria, abordando sua origem, funcionamento e evolução com ênfase na aplicação em equipamentos automotores especialmente em empilhadeiras, destacando seus benefícios para a operação, segurança e gestão eficiente dos processos logísticos.

2.1 Telemetria

A telemetria veicular é uma tecnologia que permite o monitoramento remoto de veículos por meio da captação e transmissão de dados em tempo real, como velocidade, localização, consumo de combustível e condições mecânicas. Essa prática tem sido amplamente utilizada na gestão de frotas, com o objetivo de reduzir custos, aumentar a segurança, prevenir falhas e melhorar a eficiência operacional. Apesar dos inúmeros benefícios, a telemetria apresenta limitações, como o alto custo de implantação, a dependência de conectividade estável, questões relacionadas à privacidade e possível resistência dos motoristas ao monitoramento contínuo. Em paralelo, a telemetria também tem sido aplicada em redes computacionais, onde permite o acompanhamento preciso do tráfego e do desempenho de dispositivos de rede, especialmente por meio das filas associadas às portas dos switches. A técnica de In-band Network Telemetry (INT), por exemplo, permite a coleta de métricas detalhadas com alta frequência, proporcionando uma visão precisa do estado da rede (BRAGATTO et al., 2024). Essa abordagem é fortalecida pelo uso de tecnologias como a linguagem P4 e arquiteturas de redes definidas por software (SDN), permitindo medir latência, ocupação de filas e outras variáveis críticas (JOB et al., 2023). Além disso, com o avanço da Ethernet Industrial e da integração com sistemas inteligentes de controle e automação, a telemetria passou a ser uma ferramenta estratégica para transformar dados em informações úteis e apoiar a tomada de decisões ágeis e com menor margem de erro (GOMES, 2022). Isso se torna essencial diante das exigências

do mercado atual, caracterizado por rápidas mudanças e alta competitividade. A instrumentação de redes com foco em telemetria in-band, apesar de demandar maior capacidade de armazenamento e processamento, possibilita a utilização de técnicas de inteligência artificial para análise de dados em tempo real, promovendo segurança, qualidade e lucratividade. Dessa forma, a telemetria veicular, quando aplicada corretamente, contribui significativamente para a redução de falhas operacionais e para a adaptação eficiente às demandas do mercado moderno.

2.1.1 Evolução da Telemetria

A telemetria é uma tecnologia destinada à medição e transmissão remota de dados, essencial para o monitoramento de equipamentos e veículos. Com origem na palavra grega que combina “tele” (longe) e “meter” (medir), foi inicialmente aplicada em 1845 na Rússia, para comunicações militares, e posteriormente expandiu-se para outras áreas, como meteorologia e sismologia, antes de se popularizar na década de 1990, com destaque para sua utilização na Fórmula 1, revolucionando o monitoramento veicular em tempo real (MIOTTO, 2023). Evoluindo de medições simples para sistemas inteligentes, a telemetria atualmente utiliza sensores, microcontroladores e Internet das Coisas (IoT), permitindo uma gestão segura e eficaz de equipamentos industriais e sistemas energéticos residenciais. No ambiente industrial, destaca-se por detectar falhas precocemente, otimizar recursos e aumentar a competitividade das empresas (GUILMO, 2021). Suas funcionalidades incluem geração de relatórios, análise preditiva e detecção de falhas em tempo real, sendo fundamentais para a segurança dos operadores, prevenção de paradas e otimização de recursos (FACCIONI FILHO, 2016). A telemetria também potencializa a manutenção preventiva, reduzindo custos de reparos emergenciais e evitando falhas operacionais, especialmente com o uso de tecnologias como IoT e protocolos como MQTT, que permitem monitoramento contínuo e preciso (OLIVEIRA, 2023). No entanto, sua implementação apresenta desafios, como a integração com sistemas legados e a necessidade de garantir segurança da informação, protegendo os dados sensíveis de acessos não autorizados e ataques cibernéticos (REIS et al., 2019). A aplicação conjunta de telemetria e inteligência artificial tem contribuído para mitigar riscos operacionais em equipamentos de movimentação, antecipando falhas e emitindo alertas preventivos, o que fortalece a segurança e melhora a eficiência dos

processos (MENEGHINI, 2018). Casos reais demonstram que empresas que adotaram a telemetria reduziram custos de manutenção, aumentaram a produtividade e aprimoraram a segurança operacional, evidenciando seu impacto positivo na gestão industrial (MACHADO, 2017). Assim, a telemetria se consolida como uma ferramenta estratégica para a inovação, eficiência e sustentabilidade nos mais diversos setores industriais.

2.1.2 Arquitetura de Sistemas de Telemetria

A arquitetura de sistemas de telemetria exerce papel fundamental no monitoramento e na segurança de equipamentos de movimentação, ao possibilitar a coleta, transmissão e análise de dados em tempo real. Essa arquitetura é responsável por integrar sensores, dispositivos de controle e servidores de processamento, além de definir as camadas de hardware e software necessárias para o funcionamento eficaz do sistema, oferecendo uma visão completa do estado operacional dos equipamentos (MARTINS, 2020). Tecnologias como sensores sem fio, redes IoT, protocolos MQTT e plataformas em nuvem são comumente empregadas nesse tipo de arquitetura, permitindo o armazenamento e análise eficiente dos dados coletados (CARÁ, 2021). Os benefícios operacionais são significativos: monitoramento remoto, aumento da disponibilidade dos equipamentos, redução de custos com manutenção corretiva e maior eficiência dos processos produtivos por meio da identificação de padrões anômalos (REIS et al., 2013). Estudos de caso evidenciam a eficácia desses sistemas, mostrando redução de acidentes e melhoria na gestão da manutenção, além da possibilidade de análises históricas que contribuem para a longevidade e desempenho dos equipamentos (ANDRADE, 2022). Assim, a arquitetura de sistemas de telemetria consolida-se como elemento estratégico na modernização e na segurança operacional da indústria.

2.2 Equipamentos Automotores

Os equipamentos automotores exercem um papel estratégico na indústria de movimentação de cargas, contribuindo diretamente para a eficiência e agilidade no transporte de materiais em diversos setores. Equipamentos como empilhadeiras e guindastes são essenciais para as operações logísticas e produtivas, o que torna

fundamental a adoção de sistemas de segurança e monitoramento que evitem acidentes e prejuízos (SILVEIRA, 2021). Contudo, desafios persistem, como a ausência de visibilidade em tempo real, dificultando a previsão de falhas mecânicas e comprometendo a segurança operacional. Além disso, a interoperabilidade entre sensores e plataformas de telemetria nem sempre é eficiente, exigindo avanços em segurança cibernética, infraestrutura digital e capacitação profissional para garantir a integridade dos operadores (SILVA; AMICI, 2022). A integração da telemetria com sistemas de segurança, como alarmes sonoros, alertas visuais e bloqueios automáticos, permite respostas imediatas a situações de risco, reduzindo danos materiais e lesões (OLIVEIRA, 2025). Nesse contexto, a telemetria se consolida como ferramenta indispensável para a gestão inteligente e segura dos equipamentos automotores no setor industrial.

2.3 Telemetria em Equipamentos Automotores

Sua aplicação tem sido decisiva na modernização do setor automotivo, oferecendo soluções inovadoras para otimização do desempenho e sustentabilidade (FUCK, 2024). No contexto industrial, a telemetria é essencial para a segurança e eficiência dos equipamentos, possibilitando análise preditiva dos dados operacionais por meio de sensores inteligentes, que detectam falhas precocemente, diminuem riscos de acidentes e viabilizam a manutenção preditiva, evitando paradas não programadas. Entre as tecnologias empregadas destacam-se GPS, sensores de temperatura e acelerômetros, que permitem rastrear a localização e condições dos equipamentos, tornando a gestão das frotas mais abrangente e detalhada (BOFF, 2017). Contudo, desafios persistem, como a compatibilidade entre diferentes marcas e modelos de sistemas de telemetria e a necessidade de padronização para garantir interoperabilidade, coleta e análise eficaz dos dados. Além disso, a segurança cibernética é imprescindível para proteger informações sensíveis transmitidas pelos equipamentos (SILVA; AMICI, 2022). A implementação desses sistemas gera impactos positivos na produtividade das empresas do setor ao permitir o acompanhamento remoto das atividades da frota e identificar oportunidades de otimização, tornando-se uma ferramenta essencial para impulsionar o crescimento organizacional (MIOTTO, 2023). As perspectivas futuras para o desenvolvimento da telemetria incluem a integração com inteligência artificial e machine learning para

análise preditiva, possibilitando antecipar falhas com base em padrões identificados e otimizar rotinas operacionais, aprimorando ainda mais o desempenho dos equipamentos (SANTOS, 2018).

2.3.1 Aplicações em Logística e Indústria

A telemetria depende da integração de tecnologias como sensores embarcados, GPS, redes sem fio e softwares especializados de análise, os quais possibilitam o monitoramento preciso e em tempo real dos equipamentos, apoiando decisões estratégicas nas operações industriais (SANTOS, 2018). Em veículos elétricos, a telemetria contribui diretamente para a segurança ao permitir o monitoramento de variáveis críticas como sobrecargas e superaquecimento, possibilitando a manutenção antecipada e promovendo maior vida útil dos componentes, além de reduzir comportamentos de risco e acidentes, por meio da análise de dados históricos e padrões de condução (PINTO, 2017). Os avanços recentes na área têm sido impulsionados pela integração com inteligência artificial e automação industrial, proporcionando sistemas mais sofisticados, eficientes e seguros para o setor logístico e industrial (FACCIONI FILHO, 2016). As perspectivas futuras indicam um cenário de maior eficiência com a convergência da telemetria, Internet das Coisas (IoT) e IA, viabilizando gestão integrada da frota em tempo real, análises preditivas de falhas, otimização dos processos operacionais e práticas sustentáveis, como o controle de emissões e consumo de combustível. Soluções como o sistema DISPATCH demonstram os avanços nessa direção, promovendo automação inteligente e elevação da produtividade (ANDRADE, 2022).

2.4 Telemetria em Empilhadeiras

A aplicação da telemetria em empilhadeiras tem se mostrado uma solução eficaz para o aprimoramento do controle operacional e da gestão de frotas, por meio da coleta de dados em tempo real sobre consumo de energia, utilização dos equipamentos e desgaste de componentes (LINO, 2019). Sensores embarcados monitoram parâmetros como temperatura do motor, velocidade, vibração e carga transportada, permitindo a manutenção preditiva, a redução de custos operacionais e o aumento da vida útil dos equipamentos (OLIVEIRA, 2023). Além disso, o sistema

analisa estilos de condução, detecta comportamentos de risco, como excesso de velocidade ou impactos, e emite alertas automáticos que contribuem para a segurança dos operadores. Os testes realizados validaram a eficácia da tecnologia, consolidando sua importância para operações logísticas inteligentes e sustentáveis (LINO, 2019). A integração da telemetria com sistemas empresariais, embora desafiadora, permite o alinhamento das informações coletadas com processos de gestão mais amplos, otimizando recursos e garantindo maior confiabilidade nas operações (OLIVEIRA, 2023). A padronização das práticas de telemetria em empilhadeiras permite uma análise mais precisa do desempenho dos equipamentos, facilitando a adoção de medidas preventivas e melhorando a produtividade nas operações de movimentação de cargas (FUCK, 2024). Estudos de caso demonstram que empresas que implementaram esses sistemas obtiveram redução de custos, melhoria nas condições de trabalho e aumento da eficiência na gestão da frota, evidenciando o potencial transformador da telemetria no setor logístico (REIS et al., 2019).

2.5 Telemetria e Sustentabilidade (ODS)

A telemetria exerce um papel fundamental no monitoramento remoto e em tempo real de equipamentos, sendo uma aliada estratégica para a eficiência operacional e a sustentabilidade, em consonância com os Objetivos de Desenvolvimento Sustentável (ODS). Por meio da coleta e análise de dados, é possível otimizar o consumo de recursos, prever falhas, reduzir custos operacionais e prolongar a vida útil dos equipamentos, além de melhorar a gestão de frotas e reduzir impactos ambientais (LINO, 2019). Com testes e simulações, confirma-se a eficácia desses sistemas para ajustes estratégicos voltados à sustentabilidade e ao desempenho técnico (PINTO, 2017). Outro aspecto relevante é a relação entre a telemetria e a segurança dos trabalhadores, visto que o monitoramento contínuo permite identificar situações de risco antes que se concretizem, prevenindo acidentes e promovendo ambientes de trabalho mais seguros (FUCK, 2024). Casos de sucesso relatados por empresas do setor de movimentação de cargas comprovam os impactos positivos da telemetria na redução de custos, aumento da produtividade e elevação da segurança laboral, servindo de modelo para organizações que almejam excelência operacional e sustentabilidade (FACCIONI FILHO, 2016).

3 METODOLOGIA

O desenvolvimento deste projeto de TG foi estruturado em cinco etapas principais, com o objetivo de garantir um processo lógico, seguro e tecnicamente consistente. Cada fase foi cuidadosamente planejada para atender aos requisitos funcionais e normativos do projeto, com base em normas técnicas como a NBR 14153 e a NBR ISO/IEC 20922.

A metodologia adotada envolve desde a aquisição de dados por sensores, passando pelo processamento das informações, comunicação remota e interação com o operador, até a validação final do sistema. Essa abordagem modular permite melhor controle sobre cada aspecto do desenvolvimento, assegurando a confiabilidade dos dados, a usabilidade da interface e a segurança da operação.

A seguir, são descritas detalhadamente as cinco etapas que compõem o processo de desenvolvimento do sistema:

Desenvolvimento em 5 etapas

Na **Etapa 1** a Aquisição de Dados é a primeira etapa do desenvolvimento foi dedicada à preparação e configuração dos sensores que capturam os dados essenciais do equipamento.

- **Sensor Hall** para contagem de rotações e cálculo de velocidade.
- **MPU6050** para detecção de inclinação.
- **Tensão da Bateria** divisor resistivo.
- **LM35** para medir a temperatura.
- **Sensor RCWL-0516** para detectar pessoas próximas.
- **Sistema de Arquivos** Cartão SD para registrar atividade.

Para garantir precisão, todos os sensores foram calibrados de acordo com os parâmetros da **norma NBR 14153**, que define critérios de segurança para sistemas de parada de emergência em máquinas e equipamentos. Além disso, foram realizados testes preliminares para verificar a consistência dos dados e a estabilidade da leitura.

O circuito foi equipado com proteções contra curto-circuito, além da aplicação de filtros RC nas entradas analógicas — esses filtros atuam como uma primeira barreira contra ruídos elétricos, assegurando que os dados recebidos sejam limpos e confiáveis.

Na **Etapa 2** com os dados sendo adquiridos corretamente, passou-se ao desenvolvimento dos algoritmos responsáveis pelo processamento dessas informações, sendo que um dos principais cálculos realizados é o da velocidade, obtida por meio da seguinte equação:

Velocidade: O firmware utiliza uma janela de contagem, acumulando os pulsos do sensor Hall e calculando a velocidade média nesse intervalo.

1. Frequência de pulsos de medida:

$$f_p = 0{,}884Hz$$

2. Conversão para rotações por segundo (4 pulsos por volta):

$$f_{rev} = \frac{f_p}{PULSOS\_POR\_VOLTA} = \frac{0{,}884}{4} \approx 0{,}221\ rev/s$$

3. Circunferência da roda (R = 0,2 m):

$$C = 2\pi R = 2\pi * 0{,}2 \approx 1{,}2566\ m$$

4. Velocidade em metros por segundo:

$$v_m = f_{rev} * C = 0{,}221 * 1{,}2566 \approx 0{,}2777\ m/s$$

5. Conversão para km/h:

$$v_{km/h} = v_{m/s} * 3{,}6 = 0{,}2777 * 3{,}6 \approx 1\ km/h$$

Além disso, o sistema aplica um tempo limite para zerar a velocidade em caso de ausência de pulsos e utiliza um filtro de média exponencial (EWMA) para suavizar variações rápidas. Esse método garante maior estabilidade e confiabilidade na leitura em comparação com o cálculo direto por período.

A diferença entre o valor injetado (0,884 Hz) e o mostrado (1 km/h) deve-se aos arredondamentos matemáticos na conversão e à precisão dos cálculos com variáveis float. O sistema está funcionando conforme programado, convertendo frequência de pulsos em velocidade linear baseada no raio da roda.

Inclinação: Processamento de dados do MPU6050 usando funções trigonométricas (arctangente) para determinar ângulos nos eixos X e Y configurado no endereço I2C 0x69.

Princípio de Cálculo
O cálculo da inclinação utiliza as componentes do vetor aceleração para determinar os ângulos de inclinação nos eixos X e Y através das seguintes fórmulas:

InclinaçãoX = atan2(ax, sqrt(ay² + az²)) × 180/π
InclinaçãoY = atan2(ay, sqrt(ax² + az²)) × 180/π

Tensão da Bateria: A tensão da bateria é medida através do pino analógico conectado ao divisor resistivo, que adequa a tensão de entrada ao limite de 3,3 V do ADC do ESP32. A leitura passa por uma média aparada para reduzir ruídos e por um filtro de suavização (EWMA) para maior estabilidade.

Fórmula de Cálculo

$$Tensão\ (V) = (\frac{leitura_{analógica}}{4095} * 3{,}3) * 6{,}06$$

Implementação

A função lerTensaoBateria() realiza a leitura analógica com média aparada de 32 amostras, converte para tensão (0–3,3 V) e aplica o fator do divisor resistivo (6,06), retornando o valor em volts suavizado por filtro EWMA.

Parâmetros Utilizados

- leitura_analógica: Valor lido pelo ADC de 12 bits (0–4095)
- 3,3 V: Tensão de referência nominal do ESP32
- 20v / 3,3v = 6,06: Fator do divisor de tensão (R1/(R1+R2)) definido no circuito.

Temperatura Motor: A temperatura do motor é medida através do sensor LM35 conectado ao pino analógico GPIO34. Esse sensor fornece uma saída linear de aproximadamente 10 mV/°C, porém no ESP32 a leitura passa por ajustes de calibração para compensar não linearidades do ADC.
Fórmula de Cálculo

$$Temperatura\ (°C) = (\frac{leitura\ analogica}{4095} * 1100) * 0{,}1 * T - GAIN + T\_OFFS$$

Implementação

A função lerTemperatura() realiza a leitura do ADC com média aparada de múltiplas amostras, converte o valor em milivolts (faixa 0–1100 mV) e aplica a sensibilidade do LM35 (10 mV/°C). Em seguida, utiliza fatores de calibração (T_GAIN, T_OFFS) e um filtro de média exponencial (EWMA) para obter uma leitura mais estável e precisa.

RCWL-0516

Utiliza um sinal de micro-ondas na frequência de ~3,18 GHz (banda ISM). Quando o sinal emitido reflete em um objeto em movimento, ocorre uma mudança na frequência (efeito Doppler), que é detectada pelo circuito integrado RCWL-9196. Diferentemente de sensores PIR, detecta movimento através de materiais não metálicos (como vidro ou plástico) e não é sensível a variações de temperatura.

Vantagens

- Detecta movimento através de obstáculos não metálicos
- Não requer aquecimento inicial (diferente de PIR)
- Insensível a corpos estáticos (evita falsos positivos)
- Cobertura omnidirecional (útil para aplicações de 360°)

Limitações

- Pode ser ativado por movimento de fluidos (ex.: água em tubulações)
- Sensível a metais em movimento (ex.: ventoinhas, portas metálicas)
- Menor precisão em ambientes com múltiplos reflexos de micro-ondas

Fundamento Teórico

Baseia-se no efeito Doppler para detecção de movimento:

$$f' = f\frac{c \pm v}{c \mp v}$$

Onde f' é a frequência refletida, f a frequência emitida, c a velocidade da luz e v a velocidade do objeto.

Sistema de Arquivos

- Arquivos de log nomeados por data (log_AAAAMMDD.csv)
- Cabeçalho CSV com todas as variáveis monitoradas
- Registro periódico com timestamp preciso do RTC
- Otimização para evitar abrir/fechar arquivo frequentemente

Figura 1: Dados CSV SDCARD

Data	Hora	Operador	TempLM3	TempRTC	Bateria	Inclinacao	Velocidade	Presenca	Checklist	
29/08/2025	01:13	ID 179	47.4	26.2	0.48	66.7	7.1	0	1,1,1,1,1,0,1,0,1,	
29/08/2025	01:14		34.1	26.2	0.25	72.4	7.1	0		
29/08/2025	01:14		37.7	26.2	0.35	68.0	7.1	0		
29/08/2025	01:14		37.0	26.2	0.27	86.5	7.1	0		
29/08/2025	01:14		33.4	26.2	0.33	50.6	7.0	0		
29/08/2025	01:14		33.4	26.2	0.40	48.5	7.1	0		
29/08/2025	01:14		33.8	26.5	0.36	61.1	7.1	0		
29/08/2025	01:14		37.6	26.2	0.34	60.0	7.1	0		
29/08/2025	01:14		34.4	26.5	0.34	58.2	7.1	0		
29/08/2025	01:15		37.8	26.5	0.31	63.7	7.1	0		
29/08/2025	01:15		33.0	26.5	0.25	60.8	7.1	0		
29/08/2025	01:15		37.9	26.5	0.31	58.5	7.0	0		
29/08/2025	01:15		34.3	26.5	0.43	66.9	7.1	0		
29/08/2025	01:15		38.5	26.5	0.53	61.3	7.1	0		
29/08/2025	01:15		36.9	26.5	1.64	60.9	7.1	0		

Fonte: Elaborado pelo próprio Autor (2025).

A figura 1 apresenta um trecho do arquivo CSV gerado pelo sistema de telemetria e armazenado no cartão SD.

Cada linha corresponde a um registro de amostra coletada em tempo real.

As colunas organizam os parâmetros monitorados:

- Data e Hora: momento da coleta.
- Operador: identificação do usuário autenticado.
- TempLM35 / TempRTC: leituras de temperatura (sensor LM35 e RTC DS3231).
- Bateria: tensão da bateria medida e ajustada pelo divisor resistivo.
- Inclinacao: ângulo em graus obtido pelo sensor de inclinação.
- Velocide: velocidade calculada pelo sensor Hall.
- Presença: status do sensor de presença.

- Checklist: sequência de confirmações realizadas pelo operador.

Gráfico

Figura 2: Gráfico Bateria

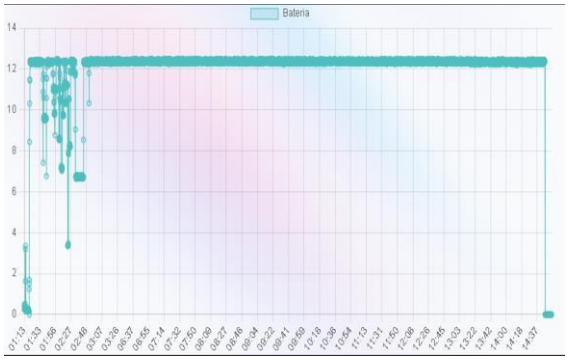


Figura 3: Gráfico Inclinação

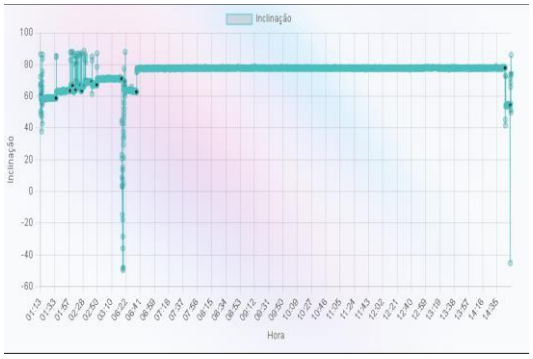


Figura 4: Gráfico Presença

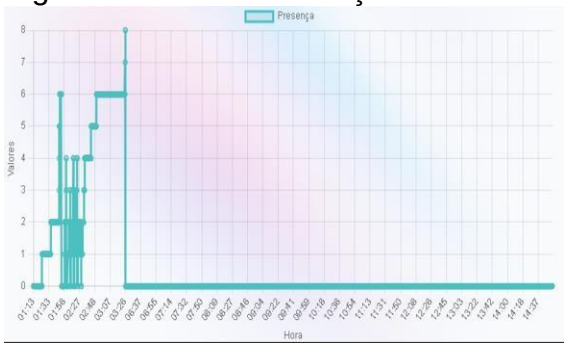


Figura 5: Gráfico TempRTC

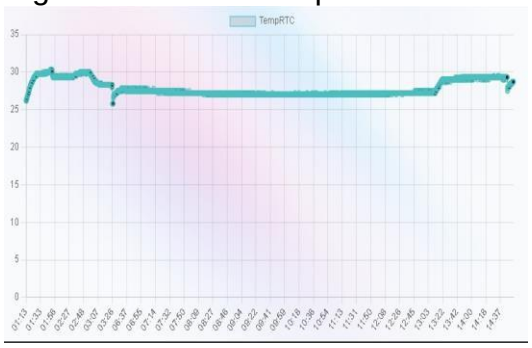


Figura 6: Gráfico Temperatura Motor

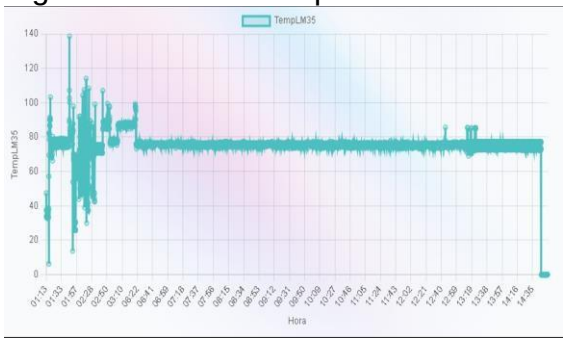
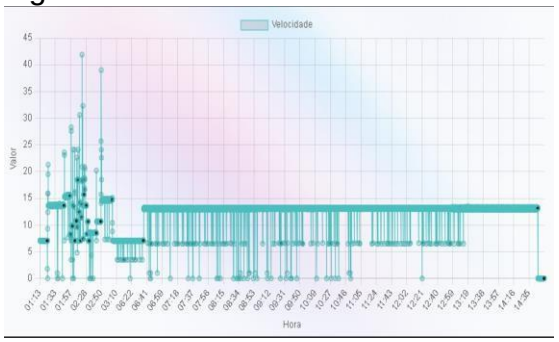


Figura 7: Gráfico Velocidade



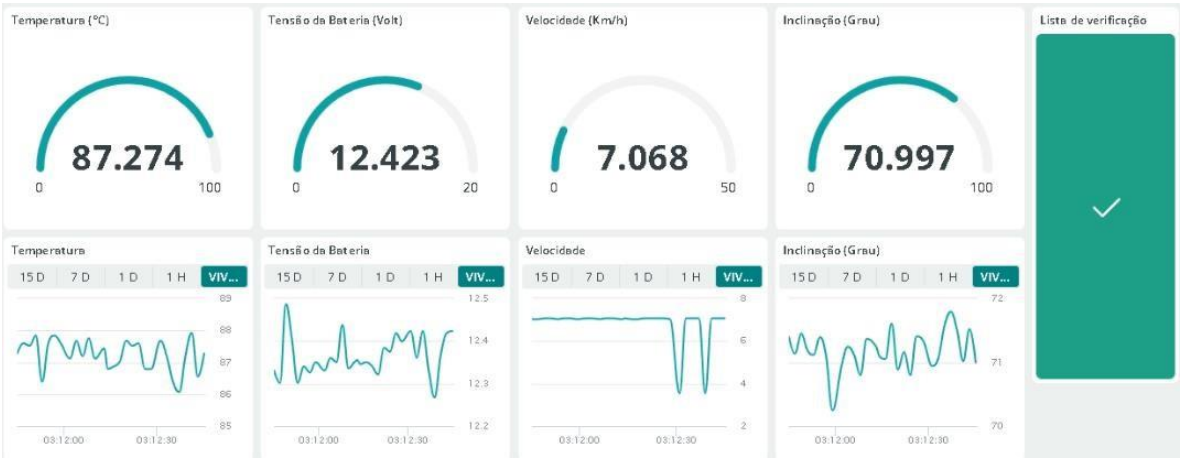
Fonte: Elaborado pelo próprio Autor no chat2csv.com (2025).

As figuras 2 a 7 apresentam os gráficos gerados a partir dos dados armazenados em planilhas CSV no cartão de memória do sistema de telemetria. Cada gráfico corresponde a uma variável monitorada:

Esses gráficos permitem visualizar de forma clara o comportamento temporal das variáveis críticas, possibilitando identificar tendências, picos ou desvios em relação aos limites estabelecidos. Dessa forma, o sistema garante não apenas o monitoramento em tempo real, mas também a análise histórica para manutenção preventiva e segurança operacional.

Na **Etapa 3** é feito há Comunicação para que o sistema possa enviar e monitorar dados remotamente, foi adotado o **protocolo MQTT**, conforme definido na **norma NBR ISO/IEC 20922**. Esse protocolo é leve, eficiente e ideal para aplicações embarcadas e IoT.

Figura 8: Painel Arduino IoT Cloud



Fonte: https://app.arduino.cc/ (2025).

A figura 8 apresenta o painel da Arduino IoT Cloud, no qual são exibidos em tempo real os dados coletados pelo hardware ESP32 do sistema de telemetria.

- Temperatura (°C): leitura do motor, exibida em mostrador analógico e gráfico temporal.
- Tensão da Bateria (V): monitoramento da alimentação elétrica, com registro histórico.
- Velocidade (km/h): valor calculado pelo sensor Hall, mostrado em tempo real.
- Inclinação (grau): ângulo medido pelo sensor de inclinação.
- Lista de verificação: indicador de conclusão do checklist do operador.

Figura 9: Variáveis do sistema

Cloud Variables ADD

	Name ↓	Last Value	Last Update
☐	**checklistCompleto** `bool checklistCompleto;`	true	17 Sep 2025 03:10:08
☐	**inclinacao** `CloudAngle inclinacao;`	65.7	18 Sep 2025 01:50:33
☐	**temperatura** `CloudTemperatureSensor temperatura;`	86.111	18 Sep 2025 01:50:33
☐	**tensaoBateria** `CloudElectricPotential tensaoBateria;`	13.92	18 Sep 2025 01:50:33
☐	**velocidade** `CloudVelocity velocidade;`	15.161	18 Sep 2025 01:50:33

Fonte: https://app.arduino.cc/ (2025).

A figura 9 mostra a tela de variáveis da Arduino IoT Cloud, que são os parâmetros enviados e monitorados pelo ESP32 no sistema de telemetria.

- checklistCompleto (bool): variável booleana que indica se o checklist de segurança foi concluído pelo operador.

- inclinacao (CloudAngle): armazena o valor do ângulo de inclinação da empilhadeira, obtido pelo sensor MPU6050.

- temperatura (CloudTemperatureSensor): representa a temperatura do motor, medida e processada pelo sistema.

- tensaoBateria (CloudElectricPotential): registra a tensão da bateria após leitura e cálculo pelo divisor resistivo.

- velocidade (CloudVelocity): indica a velocidade atual, calculada a partir dos pulsos do sensor Hall.

Figura 10: Configuração da rede wi-fi

Fonte: https://app.arduino.cc/ (2025).

A figura 10 apresenta a tela de configuração de rede da Arduino IoT Cloud para o dispositivo DOIT ESP32 DEVKIT V1 utilizado no sistema de telemetria da empilhadeira.

- Nome do Wi-Fi (SSID): rede sem fio à qual o ESP32 será conectado.
- Senha: chave de acesso à rede Wi-Fi.
- Chave secreta: credencial gerada pela plataforma para autenticar o dispositivo na nuvem.

Na **Etapa 4** à Interface Homem-Maquina (IHM); a interação entre o operador e o sistema foi cuidadosamente planejada com base na mesma **NBR 14153**, aplicando suas diretrizes de segurança ao design da interface. A tela TFT (128X160) _ST7735.

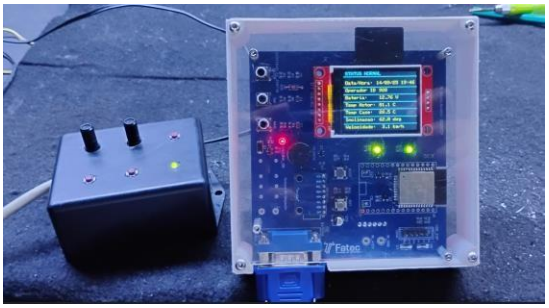
Exibe:

- Um checklist sequencial com 20 perguntas, que o operador deve responder antes de iniciar a operação.
- As leituras dos sensores de forma clara e contínua.
- Alertas visuais (mensagens na tela) e auditivos (via buzzer), ativados automaticamente em caso de falha.

A IHM foi desenhada para ser objetiva e fácil de usar, mesmo em condições de trabalho intensas, reduzindo a possibilidade de erro humano.

Na **Etapa 5** é feito a validação, o sistema foi submetido a uma fase intensiva de testes em um ambiente controlado durante 24 horas contínuas. Durante esse período, foram simuladas diversas situações de falha e operação normal para verificar a robustez e a reatividade do sistema.

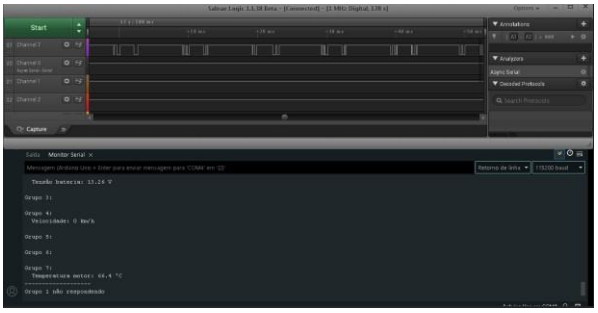
Figura 11: Sistema de monitoramento



Fonte: Elaborado pelo próprio Autor (2025).

A figura 11 demonstra o sistema de monitoramento em tempo real em bancada, onde os valores de sensores de tensão, temperatura, inclinação, presença e velocidade são simulados e exibidos no display TFT, validando o funcionamento do projeto de telemetria.

Figura 12 : Sinais digitais capturados



Fonte: Elaborado pelo próprio Autor (2025).

A figura 12: tela do notebook está dividida em duas partes principais:

Parte superior (software Logic 1.1.30):

Mostra os sinais digitais capturados pelo analisador lógico, conectados ao pino 7 (K-Line) da interface OBD2. É possível observar os pacotes de dados referentes à comunicação entre a ECU e o Arduino UNO, exibidos como formas de onda no tempo.

Parte inferior (IDE Arduino – Monitor Serial):

Mostra os dados já decodificados pelo Arduino UNO e enviados via porta serial.

Entre os parâmetros recebidos da ECU estão:

- Tensão da bateria = 13,26 V
- Velocidade = 0 km/h
- Temperatura do motor = 66,4 °C

Figura 13: Tela dados ECU

Fonte: Elaborado pelo próprio Autor (2025).

A figura 13 mostra a tela TFT do sistema de telemetria em modo Status Normal.

Dados em verde (ECU): são informações recebidas do módulo eletrônico do veículo via protocolo OBD2 e interpretadas pelo ESP32.

Nesse caso:

- ECU Ubat: tensão da bateria do veículo (12,65 V).
- ECU Vel: velocidade lida pelo módulo (26 km/h).
- ECU Temp: temperatura do motor (92,0 °C).

Dados em amarelo (Sensores locais): são os valores lidos diretamente pelos sensores conectados ao ESP32:

- Bateria: monitoramento de tensão (0,00 V, sem leitura válida).
- Temp LM35: temperatura medida pelo sensor LM35 (10,2 °C).
- Temp RTC: temperatura ambiente (30,5 °C).
- Impacto: aceleração medida pelo acelerômetro (0,26 g).
- Velocidade: leitura via sensor Hall ou equivalente (0,0 km/h).

O display diferencia as leituras próprias dos sensores (amarelo) das leituras do veículo via ECU/OBD2 (verde), permitindo comparar dados locais com os oficiais do módulo eletrônico.

Essa tela foi utilizada apenas para demonstrar a viabilidade de obtenção de dados diretamente da máquina, utilizando o protocolo correto com o ESP32. Esse recurso comprova que é possível expandir o projeto de telemetria para além dos sensores locais, integrando dados oficiais da ECU.

No software atual, está implementada apenas a função de leitura dos sensores conectados ao ESP32 (temperatura, tensão, inclinação, presença, velocidade etc.). Entretanto, o hardware já dispõe de entrada dedicada para a rede CAN/OBD2, o que possibilita, em versões futuras, o desenvolvimento de um software específico para determinado tipo de máquina, permitindo a aquisição direta dos dados pela rede CAN e ampliando a confiabilidade e o escopo do sistema de telemetria.

Figura 14: Cabo adaptador OBD2 para DB15

Fonte: Elaborado pelo próprio Autor (2025).

A figura 14 mostra um cabo adaptador OBD2 para DB15, utilizado para comunicação entre a ECU e o sistema de telemetria com ESP32.

Na placa interna é possível identificar:

O conversor L9637D, responsável por traduzir os sinais da linha K do OBD2 para nível lógico TTL 3,3 V, compatível com o ESP32.

Um filtro de alimentação estabilizado, implementado com o TIP122, garantindo a regulação da tensão proveniente da tomada OBD2, de forma a alimentar corretamente o ESP32.

Em resumo: este cabo atua como interface física e elétrica entre a rede OBD2 do veículo e o ESP32, permitindo que os sinais sejam convertidos e tratados de forma segura no sistema de telemetria.

Figura 15: Leitura do sensor de velocidade

Fonte: Elaborado pelo próprio Autor (2025).

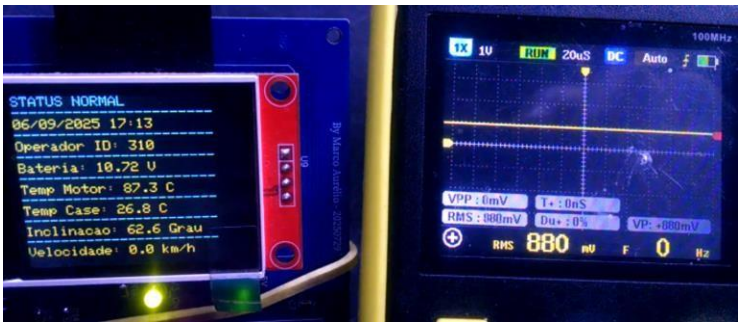
A figura 15 mostra a validação da leitura do sensor de velocidade. Quando são injetados pulsos de 0,884 Hz, o sistema calcula aproximadamente 1 km/h.

Figura 16: Leitura de tensão da bateria

Fonte: Elaborado pelo próprio Autor (2025).

A figura 16 apresenta a validação prática da leitura de tensão da bateria realizada pelo sistema. À esquerda, observa-se o display TFT conectado ao ESP32, exibindo o valor de 13,8 V, e à direita, o osciloscópio registra o mesmo valor diretamente nos terminais da bateria.

Figura 17: Leitura de temperatura



Fonte: Elaborado pelo próprio Autor (2025).

A figura 17 apresenta a validação prática da leitura de temperatura obtida no pino analógico GPIO34 do ESP32. À direita, o osciloscópio registra uma tensão contínua de aproximadamente 880 mV RMS aplicada ao pino. À esquerda, o display TFT mostra a variável Temp Motor com valor de 87,3 °C.

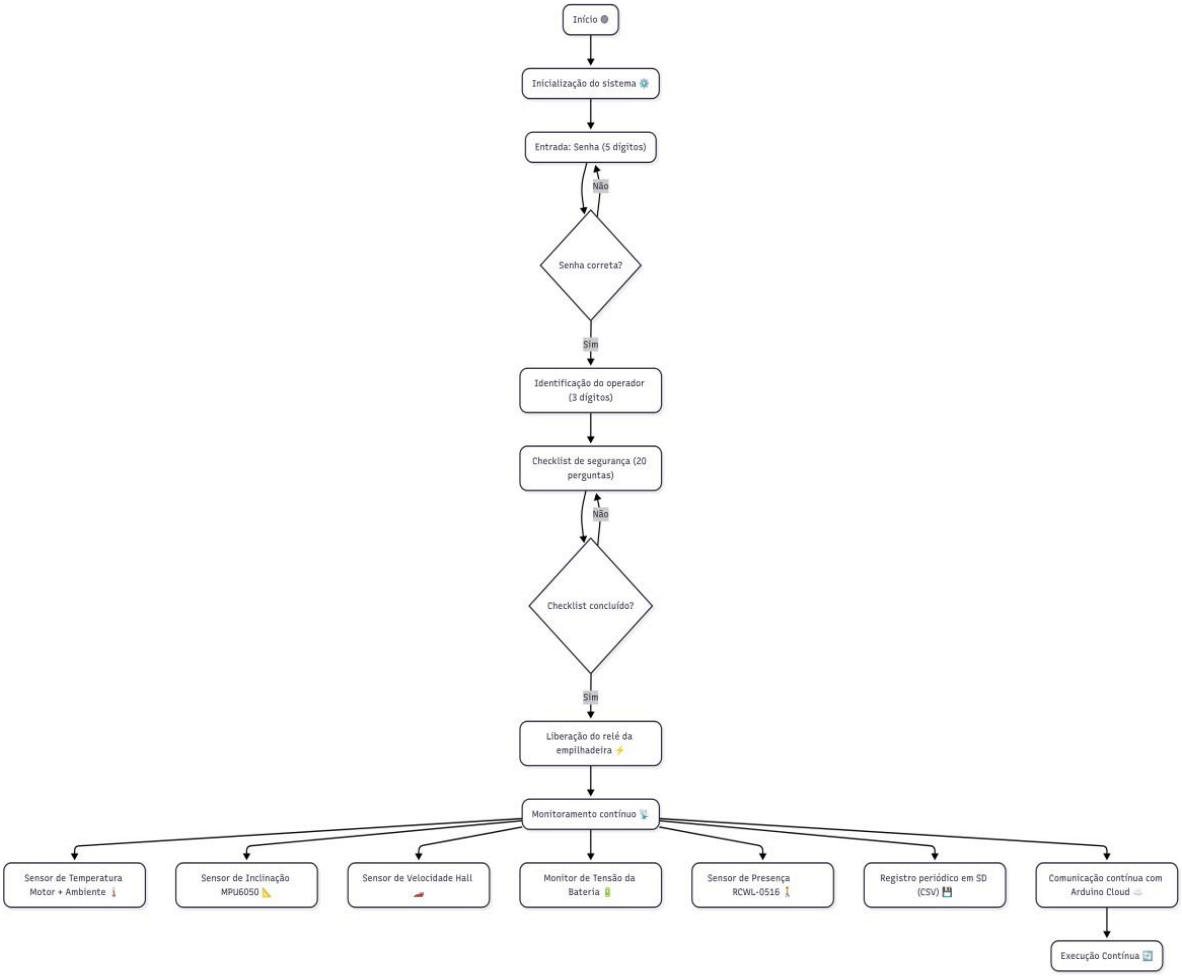
Figura 18: Telas do sistema de telemetria

Fonte: Elaborado pelo próprio Autor (2025).

A figura 18 apresenta, de forma sequencial da esquerda para a direita, todas as telas implementadas no sistema de telemetria desenvolvido. Nelas é possível observar: a tela inicial de autenticação por senha, a identificação do operador, o checklist de segurança interativo, a tela de status normal com exibição contínua das variáveis monitoradas (tensão da bateria, temperatura, inclinação, ID Operador, Data/Hora, Temperatura da case e velocidade), além das telas de alerta, que são acionadas automaticamente quando algum parâmetro ultrapassa os limites estabelecidos.

Essa disposição demonstra a estrutura lógica do fluxo de operação do sistema, evidenciando desde a inicialização e validação do operador até a fase de monitoramento e segurança ativa do equipamento.

Figura 19: Fluxograma



Fonte: Elaborado pelo próprio Autor no mermaidchart (2025).

4 RESULTADOS

O sistema demonstrou robustez na operação contínua, com:

- Tempo de resposta adequado para interface do usuário
- Precisão satisfatória nas medições dos sensores
- Confiabilidade na detecção de condições críticas
- Estabilidade na conexão com nuvem mesmo com intermitência
- Capacidade de operação offline com registro local de dados

A interface desenvolvida na plataforma Arduino IOT Cloud, demonstrou ser eficiente, oferecendo visualização remota e acompanhamento em tempo real dos dados obtidos.

Ao comparar o sistema proposto com tecnologias já existentes no mercado,

nota-se as soluções de redução de custo e recursos de conectividades robustas e flexíveis. Embora sistemas já evolucionados e vendidos possuam interfaces mais sofisticadas e com ERPs industriais, a proposta desenvolvida por nós se destaca pela facilidade de customização, acessibilidade e foco no atendimento de pequenas e médias operações de logística. Além da viabilidade de implementar rotinas de manutenção preditiva, por meio de verificação dos dados coletados. É possível prever desgastes dos equipamentos, falhas, reduzir paradas não programadas e melhorar o planejamento em relação à manutenção. O que contribui de forma direta com a diminuição de custos e maior efetividade do equipamento.

Além disso, vale ressaltar a pauta da sustentabilidade, visto que permite o consumo de recursos desnecessários e ainda, visa contribuir para o ciclo de vida dos equipamentos, o que leva consequentemente a redução de impactos ambientais.

Portanto, os resultados obtidos até o momento comprovam que o sistema de telemetria atinge os objetivos propostos em relação às questões técnicas, econômicas e operacionais. Na indústria este sistema pode trazer melhorias e transformar o gerenciamento de equipamentos de movimentação, tornando-as mais eficientes, com mais segurança e sustentável.

4.1. Desempenho do Sistema Desenvolvido

O sistema de telemetria desenvolvido demonstrou um desempenho satisfatório, atendendo aos requisitos propostos, que é o monitoramento de empilhadeiras em tempo real. Durante os testes simulados, os sensores MPU6050, temperatura LM35 e o sensor Hall 3144E — apresentaram respostas satisfatórias e rápidas na obtenção de dados operacionais, como temperatura, aceleração, inclinação.

A plataforma ESP32, utilizada como núcleo do sistema, mostrou-se eficiente ao integrar os sensores e ao transmitir os dados por via conexão Wi-Fi direcionada a nuvem, por meio do serviço *Arduino IoT Cloud*.

4.2. Comparação com Tecnologias Existentes

Comparado com às tecnologias comerciais atualmente disponíveis no mercado, o sistema desenvolvido apresenta vantagens e algumas limitações. As principais vantagens estão associadas a redução de custo de desenvolvimento, à

flexibilidade para personalização e à facilidade de manutenção, pois se baseia em plataformas de hardware livre, como o ESP 32, e em softwares de código aberto.

Soluções comerciais normalmente oferecem integração mais robusta com sistemas de gestão empresarial (ERP), além de suporte técnico especializado e interfaces gráficas mais sofisticadas, capazes de gerar relatórios avançados e análises preditivas baseadas em IA. No entanto, essas soluções têm custos significativamente elevados, tanto em relação à aquisição de uma interface quanto na manutenção desta.

Por outro lado, o sistema desenvolvido se destaca pela simplicidade na instalação, pela escalabilidade e pela facilidade de adaptação a diferentes tipos de equipamentos. Ele é ideal para pequenas e médias empresas que buscam alternativas viáveis para monitorar, sem a necessidade de altos investimentos iniciais.

Em termos funcionais, o sistema desenvolvido oferece os principais recursos esperados de uma solução de telemetria moderna: coleta de dados operacionais em tempo real, geração de alertas, acompanhamento remoto e histórico de dados. Embora não atinja o mesmo nível de automação e integração dos sistemas de grandes fabricantes, cumpre com eficiência as necessidades básicas de segurança, monitoramento a gestão de equipamentos de movimentação; contribuindo em relação às questões sustentáveis.

4.3. Módulo de Conectividade Sem Fio: ESP32

Na figura 20 o módulo de conectividade sem fio utilizado neste projeto é o ESP32, um microcontrolador altamente integrado desenvolvido pela *Espressif Systems*. Esse chip é amplamente reconhecido por seu desempenho elevado aliado à eficiência energética, sendo ideal para aplicações embarcadas que exigem conectividade Wi-Fi e Bluetooth de forma simultânea (PINO, 2018). O ESP32 possui arquitetura de dois núcleos de processamento (dual-core), suporte a frequências de até 240 MHz, memória RAM dinâmica (SRAM), além de interfaces como ADC, DAC, SPI, I2C, UART, PWM, CAN e GPIOs, proporcionando ampla flexibilidade para desenvolver sistemas.

Uma das características que o diferencia de modelos anteriores, como o ESP8266, é a possibilidade de integração com sensores complementares. Entre eles destaca-se a Unidade de Medida Inercial (IMU), frequentemente acoplada ao módulo

por meio de sensores, tais como o MPU6050 ou o BNO055, permitindo o monitoramento de aceleração, rotação e orientação espacial. Essa funcionalidade é extremamente útil para aplicações que envolvem detecção de movimento, como controle gestual, estabilização de dispositivos, telemetria veicular, segurança e automação residencial (STEWART, 2019).

Além disso, o ESP32 possui recursos avançados de economia de energia, com modos como *deep sleep*, *light sleep* e *modem sleep*, o que o torna ideal para aplicações alimentadas por bateria, como dispositivos portáteis e soluções IoT remotas. Sua uniformidade com ferramentas de monitoramento em nuvem, como *Blynk, ThingsBoard, Arduino IoT Cloud* e servidores MQTT, facilita a agregação de sistemas inteligentes, dos quais tem recursos que alertam em tempo real por meio de controle remoto por aplicativos móveis.

A escolha do ESP32 para este projeto se deu por sua aplicabilidade em gerenciar múltiplas tarefas simultaneamente, pela alta demanda em documentos disponíveis e por sua relação custo-benefício, sendo amplamente adotado no meio acadêmico e em aplicações industriais.

Figura 20: Módulo ESP32 DevKit V1 (30 pinos)

Fonte: Elaborado pelo próprio (2025).

4.3.1. Sensor Inercial MPU6050

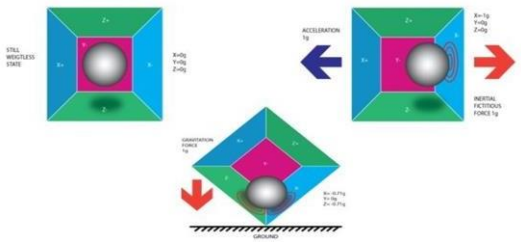
Na figura 21 o sensor MPU6050 é um módulo abundantemente utilizado em sistemas embarcados para analisar movimentos e orientação. Refere-se a um dispositivo do tipo IMU (*Inertial Measurement Unit*) que combina um acelerômetro de três eixos com um giroscópio de três eixos em um único chip. Esse sensor tem a capacidade na detecção de movimentos lineares (aceleração) e rotações angulares, sendo essencial para aplicações que requerem monitoramento de inclinação, vibração, impactos ou navegação inercial.

O MPU6050 se comunica com microcontroladores, como o ESP32, através do protocolo I2C, permitindo uma integração rápida com sistemas embarcados, tensão de operação 3-5v, conversor AD 16 bits. A alta sensibilidade do acelerômetro (±2g a ±16g) e do giroscópio (±250°/s a ±2000°/s) possibilita uma análise precisa de movimentos, o que o torna ideal para projetos como drones, robótica móvel, wearables e sistemas de telemetria veicular.

Aceleração, propriamente dita, refere-se a taxa de variação da velocidade de um corpo em seu próprio quadro de repouso instantâneo, o que é diferente da aceleração por coordenadas, sendo esta aceleração em um sistema de coordenadas fixo. Por exemplo, um acelerômetro em repouso na superfície da Terra medirá uma aceleração devida à gravidade da Terra, diretamente para cima (por definição) de g ≈ 9,81 m/s2. Em contraste, acelerômetros em queda livre (caindo em direção ao centro da Terra a uma taxa de cerca de 9,81 m/s2) medem zero.

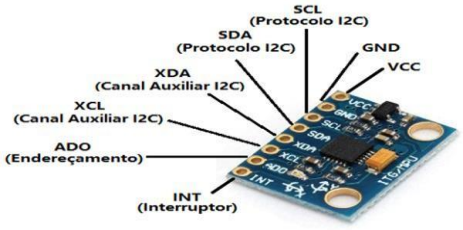
Um acelerômetro funciona segundo o princípio do efeito piezo elétrico. Imagine uma caixa cuboide com uma pequena bola dentro dela, como na foto abaixo. As paredes desta caixa são feitas de cristais piezo elétricos. Sempre que você inclina a caixa, a bola é forçada a se mover na direção da inclinação devido à gravidade. A parede com a qual a bola colide cria pequenas correntes piezo elétricas. Existem três pares de paredes opostas em um cuboide. Cada par corresponde a um eixo no espaço 3D: eixos X, Y e Z. Dependendo da corrente produzida pelas paredes piezo elétricas, podemos determinar a direção de inclinação e sua magnitude (ELETROGATE, 2022).

Figura 21: Sensor Inercial MPU6050 - Princípio do efeito piezoeléctricos



Fonte: *Eletrogate* (2020).

Figura 22: Sensor Inercial MPU6050 – Pinagem



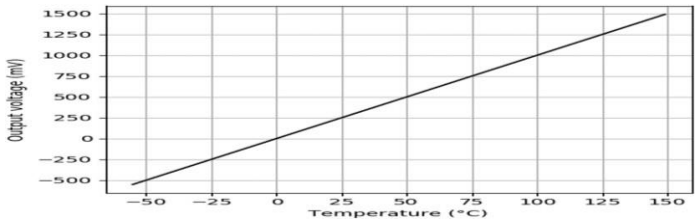
Fonte: *Eletrogate* (2020).

4.3.2. Sensor de Temperatura: LM35

Na figura 24 o LM35 trata-se de um sensor de temperatura analógico de precisão, muito utilizado em projetos de monitoramento térmico. Ele fornece uma saída linear proporcional à temperatura ambiente, com uma escala de 10 mV/°C. Por exemplo, uma leitura de 250 mV corresponde a 25 °C. O principal diferencial refere-se a facilidade de uso, não sendo necessário qualquer tipo de calibração externa.

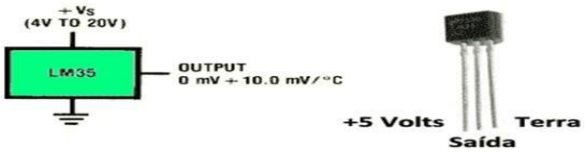
O sensor pode ser alimentado com tensões entre 4V e 30V, operando internamente na faixa de temperatura de -55°C a +150°C. No contexto deste projeto, o LM35 serve para a monitoração da temperatura de componentes críticos relacionados ao sistema, assim como o motor ou unidade hidráulica, sendo conectado ao conversor analógico-digital do ESP32 para obtenção de dados.

Figura 23: Gráfico de Saída do Sensor de Temperatura LM35



Fonte: alldatasheet.com (2025).

Figura 24: Sensor de Temperatura LM35



Fonte: Elaborado pelo próprio Autor (2025).

4.3.3. Sensor de Velocidade / Rotação: Sensor Hall 3144E

Na figura 25 o Sensor Hall 3144E é um sensor digital baseado no efeito Hall, utilizado para detectar a presença de campos magnéticos. Esse sensor fornece uma saída digital que muda de estado quando um ímã se aproxima ou se afasta, sendo ideal para aplicações como medição de rotação de eixos, para controlar a velocidade e para detectar a posição.

No presente projeto, o sensor Hall é aplicado para medir a rotação de eixos mecânicos por meio da contagem de pulsos gerados pela passagem de um ímã fixado ao eixo, permitindo assim o cálculo da velocidade em km/h ou do número de rotações por minuto (RPM). O sinal digital pode ser facilmente lido por uma entrada GPIO do ESP32 com interrupções para maior precisão.

Figura 25: Sensor Hall 3144E

Fonte: Elaborado pelo próprio Autor (2025).

4.3.4. Sensor de Presença RCWL-0516

Na figura 26 o RCWL-0516 é um sensor de movimento por radar de micro-ondas. Diferente dos sensores PIR (que detectam radiação infravermelha do corpo humano), ele emite ondas eletromagnéticas de 3,18 GHz e mede a reflexão dessas ondas nos objetos próximos.

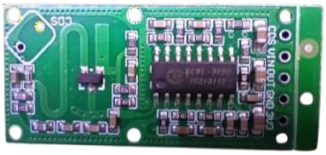
Funcionamento: O módulo gera ondas de micro-ondas contínuas.

Quando um objeto em movimento (pessoa, veículo, etc.) altera o padrão de reflexão, ocorre o chamado efeito Doppler. O circuito interno detecta essa variação e ativa a saída digital (HIGH). O sinal fica ativo por alguns segundos (tempo configurável com resistor ou capacitor).

Características principais:

- Alcance: 5 a 7 metros.
- Ângulo de detecção: ~360° (pode detectar até através de vidro ou paredes finas).
- Saída: digital (HIGH/LOW).
- Tensão de operação: 4–28 V DC.

Figura 26: O sensor de movimento RCWL-0516



Fonte: Elaborado pelo próprio Autor (2025).

4.3.5. Relógio RTC-DS3231

Na figura 27 o RTC DS3231 é um módulo de relógio em tempo real (Real Time Clock) que mantém informações de data, hora e temperatura com alta precisão.

Ele fornece ao ESP32 a hora e data corretas, mesmo quando o microcontrolador é desligado. Também pode medir a temperatura interna, usada para compensação e para aplicações de monitoramento.

Oscilador interno compensado por temperatura (TCXO) - garante precisão muito maior que RTCs comuns, sem necessidade de cristal externo. SQW/32K - pinos opcionais que podem gerar sinal quadrado ou clock de 32 kHz (não usados aqui).

Figura 27: Relógio RTC DS3231

Fonte: Elaborado pelo próprio Autor (2025).

4.3.6. Cartão de Memória

Na figura 28 o Cartão SD é responsável pelo armazenamento de dados locais em formato CSV.

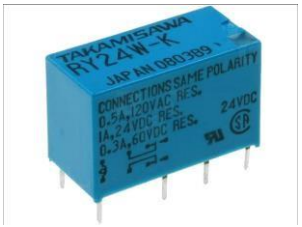
Figura 28: Cartão SD 4Mb formatação FAT32

Fonte: Elaborado pelo próprio Autor (2025).

4.3.7. Relé

A figura 29 Módulo Relé é responsável pelo controle de ativação/desativação do equipamento

Figura 29: O relé (RY-12W-K)



Fonte: th.bing.com (2025).

4.3.8. Buzzer

A figura 30 mostra Buzzer Alertas sonoros para notificações críticas

Figura 30: Buzzer (HYDZ)

Fonte: Elaborado pelo próprio Autor (2025).

4.3.9. Display TFT

Na figura 31 esse módulo combina display gráfico colorido (ST7735) e cartão SD, ambos comunicando via SPI com o ESP32. O display mostra informações em tempo real (temperatura, velocidade, status). O SD armazena dados históricos (ex.: leituras de sensores).

Figura 31: Display ST7735

Fonte: Elaborado pelo próprio (2025).

4.4. Estrutura de Hardware

1. OBD2 K-LINE (DB15 - interface automotiva)

É a conexão padrão de diagnóstico OBD2 (K-Line) usada em veículos.

Permite ler sensores como:

- Temp. (temperatura)
- Pres. (pressão)
- Veloc. (velocidade)
- +B (alimentação do veículo)
- K-Line_TX / K-Line_RX: canais de comunicação serial com o barramento OBD2.

2. Proteções adicionais

D9 e D10 (SMAJ3.3): protegem as linhas de comunicação contra surtos de tensão.

Esse circuito:

- Usa +12 V do veículo como fonte.
- Permite comunicação entre ESP32 - OBD2 (K-Line).
- Faz a leitura de parâmetros automotivos (temperatura, inclinação, velocidade etc.).
- Inclui proteção contra surtos e adequação de nível de tensão para não danificar o microcontrolador.

Em outras palavras: é o elo entre o veículo (via OBD2) e o sistema de telemetria no ESP32.

4.4.1. Esquema Elétrico: ESP32

- ESP32-DEVKITC V1 (módulo com USB e fonte integrado) é uma placa de desenvolvimento.

Já vem com:

- Chip ESP32-WROOM-32 montado.
- Conversor USB–Serial: permite gravar e comunicar com o PC via cabo USB.
- Regulador de tensão: aceita 5 V da USB ou fonte externa e converte para 3,3 V.
- Circuito de reset/boot automático: não precisa apertar botões ao programar.
- Uso: ideal para testes, protótipos e desenvolvimento rápido, basta ligar no USB e começar a programar.

- ESP32-WROOM-32 (somente o chip/módulo)

- É o módulo “puro”, contendo apenas o ESP32 com memória flash.
- Precisa de circuitos externos para funcionar:
- Regulador de 3,3 V.
- Conversor USB–Serial para gravação.
- Circuito de reset/boot (com transistores).
- Uso: indicado para produtos finais, quando se quer integrar o ESP32 diretamente em um projeto, otimizando custo e espaço.

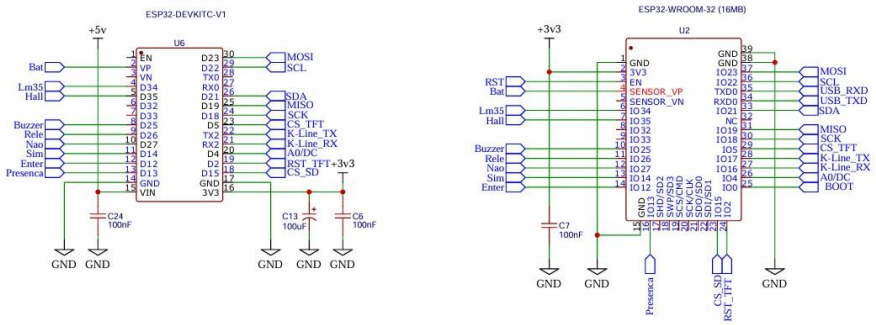
Resumo da diferença:

- O ESP32-DEVKITC é uma placa completa, pronta para desenvolvimento, com USB e fonte inclusos.
- O ESP32-WROOM-32 é apenas o módulo WiFi/Bluetooth + MCU, exigindo que o projetista adicione fonte, USB e circuitos auxiliares para funcionar.

Em outras palavras:

- DEVKITC = fácil de usar (plug and play).
- WROOM = flexível e econômico para projetos finais.

Figura 32: Esquema Elétrico: Módulo ESP32-C3



Fonte: Elaborado pelo próprio Autor **(**2025).

4.4.2. Esquema Elétrico: Display ST7735

Na figura 33 esse esquema mostra um display TFT 1.8" (128×160 pixels, controlador ST7735, interface SPI) com suporte adicional para cartão SD.

Funcionamento do display (ST7735):

O controlador ST7735 recebe comandos e dados via SPI.

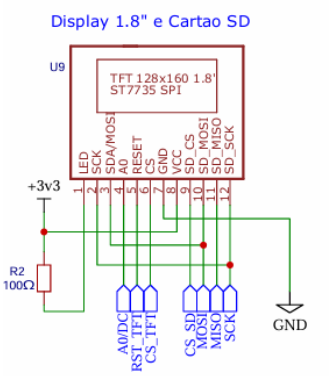
Os sinais principais:

- MOSI / SDA (Serial Data): envia os dados de imagem (pixels).
- SCK (Serial Clock): fornece o clock para sincronizar a transmissão SPI.
- CS_TFT (Chip Select): habilita o display para comunicação.
- RESET: reinicializa o display.
- A0/DC (Data/Command): define se o que está sendo enviado é um comando ou dado gráfico.
- Alimentado com 3,3 V, ele aciona a matriz TFT de 128×160 pixels coloridos.

Funcionamento do Cartão SD (no mesmo módulo):

Também usa a interface SPI (MOSI, MISO, SCK), mas com um pino CS_SD exclusivo para selecionar o cartão. Permite salvar/ler arquivos (ex.: logs de telemetria em CSV). Resistor R2 (100 Ω): Limita a corrente para o pino de LED do backlight, protegendo e controlando a intensidade da iluminação.

Figura 33: Esquema Elétrico: Display ST7735 128 x 160 1.8”

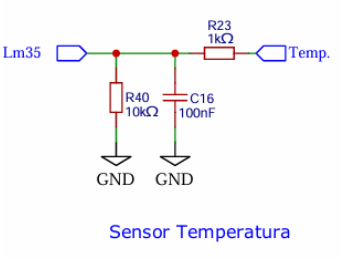


Fonte: Elaborado pelo próprio Autor (2025).

4.4.3. Esquema Elétrico: Sensor de Temperatura

A figura 34 mostra o circuito do LM35, sensor analógico de temperatura no qual fornece uma saída linear proporcional à temperatura ambiente (10 mV/°C). É conectado a uma entrada analógica do ESP32, permitindo o monitoramento térmico do motor ou de componentes da empilhadeira.

Figura 34: Esquema Elétrico: Sensor de Temperatura LM35

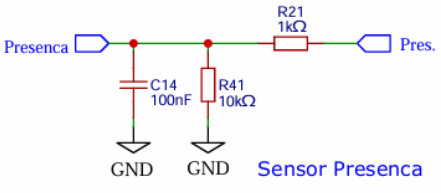


Fonte: Elaborado pelo próprio Autor (2025).

4.4.4. Esquema Elétrico: Sensor de Presença

A figura 35 mostra o circuito do RCWL-0516 que funciona como um “radar em miniatura” para detectar movimento, sendo mais sensível e de maior alcance que sensores PIR comuns.

Figura 35: Esquema Elétrico: Sensor de Presença

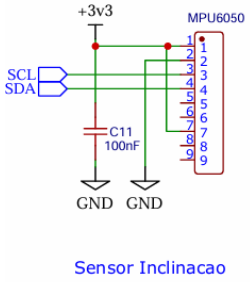


Fonte: Elaborado pelo próprio Autor (2025).

4.4.5. Esquema Elétrico: Sensor de Inclinação

A figura 36 mostra o circuito do MPU6050 que combina um acelerômetro de três eixos e um giroscópio, possibilitando a medição de aceleração, inclinação e rotação. A comunicação é feita via I2C com o ESP32. Esse sensor é essencial para detectar impactos, inclinação da empilhadeira e análise de segurança operacional.

Figura 36: Esquema Elétrico: Sensor de Movimento MPU6050

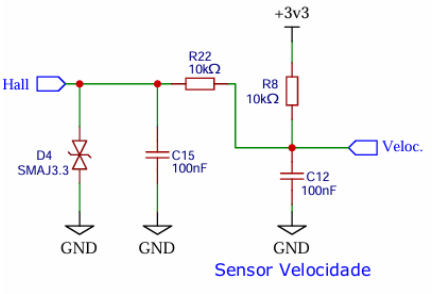


Fonte: Elaborado pelo próprio Autor (2025).

4.4.6. Esquema Elétrico: Sensor Magnético Hall

A figura 37 mostra o circuito do sensor Hall 3144E que detecta campos magnéticos e usa-se para monitorar velocidade ou posição de rodas/eixos. Ao detectar um ímã passando em sua frente, ele gera um sinal digital.

Figura 37: Esquema Elétrico: Sensor Magnético Hall (3144E)

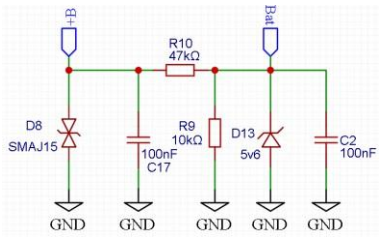


Fonte: Elaborado pelo próprio Autor (2025).

4.4.7. Esquema Elétrico: Divisor de Tensão Bateria

A figura 38 mostra o divisor resistivo (R10 e R9) reduz a tensão da bateria para um valor proporcional, dentro da faixa de leitura do ADC do ESP32. Exemplo: se a bateria for 12 V, o divisor ajusta para algo próximo de 3 V. Diodo zener D13 (5V6) garante que a tensão na entrada nunca ultrapasse ~ 5,6 V, protegendo o pino analógico do ESP32. Capacitores (C17 e C2, ambos 100 nF) fazem o filtro (desacoplamento), eliminando ruídos e estabilizando a leitura analógica. Diodo D8 (SMAJ15) Atua como proteção contra surtos (transientes de tensão), evitando danos ao circuito em caso de picos vindos da bateria.

Figura 38: Esquema Elétrico: Divisor de Tensão da Bateria



Fonte: Elaborado pelo próprio Autor (2025).

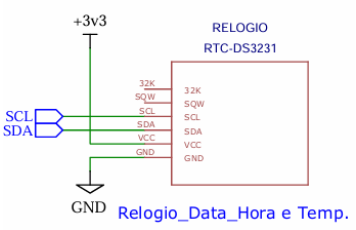
4.4.8. Esquema Elétrico: Relógio

A figura 39 mostra o circuito do DS3231, ele é responsável por fornecer data, hora exata e temperatura ao ESP32 via I²C, garantindo que o sistema de telemetria registre os eventos corretamente no tempo, mesmo em reinicializações ou quedas de energia.

Funcionamento no esquema mostrado

- Alimentação (VCC = 3,3 V e GND): mantém o circuito ativo.
- Comunicação I²C (SCL e SDA): permite que o ESP32 troque dados com o DS3231.
- SCL = linha de clock.
- SDA = linha de dados.

Figura 39: Esquema Elétrico: Relógio (DS3231)

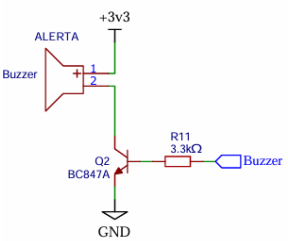


Fonte: Elaborado pelo próprio Autor (2025).

4.4.9. Esquema Elétrico: Buzzer

A figura 40 mostra o circuito do buzzer ativado por um transistor NPN BC847, configurados como chave. Ele fornece alertas sonoros em situações críticas, como alta temperatura ou falha no sistema. O resistor serve como polarização do circuito.

Figura 40: Esquema Elétrico: Buzzer de Alerta

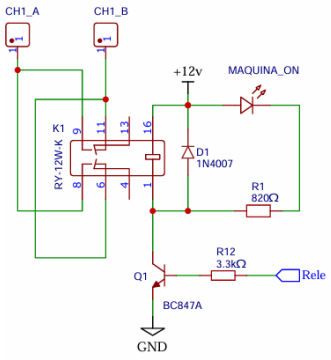


Fonte: Elaborado pelo próprio Autor (2025).

4.4.10. Esquema Elétrico: Relé RY-12-K

A figura 41 mostra o circuito do relé controlado pelo ESP32 através de um transistor, o relé é aplicado para atuar cargas maiores, como chave geral do sistema. Opera com tensão de controle de 12V e permite o isolamento galvânico entre o circuito lógico e a carga.

Figura 41: Esquema Elétrico: Relê RY-K (K1)



Fonte: Elaborado pelo próprio Autor (2025).

4.4.11. Esquema Elétrico: Reguladores de Tensão

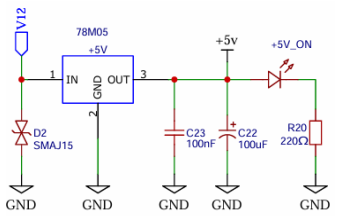
Regulador de tensão +5 V

A figura 42 apresenta o circuito regulador de tensão responsável pela alimentação do ESP32 e demais módulos. O sistema recebe +12 V na entrada, que é protegido por um diodo de supressão de transientes (SMAJ15) contra picos e surtos de tensão. O regulador 78M05 converte essa tensão para +5 V estáveis.

Na saída, capacitores de desacoplamento e filtragem (C23 = 100 nF e C22 = 100 µF) garantem a redução de ruídos e a estabilidade da linha de alimentação. Um LED indicador em série com resistor (R20 = 220 Ω) sinaliza o estado de operação da fonte (+5V_ON).

Esse arranjo assegura uma alimentação estável e protegida, fundamental para o correto funcionamento do ESP32 e sensores do sistema de telemetria.

Figura 42: Esquema Elétrico: Regulador de Tensão 78M05.



Fonte: Elaborado pelo próprio Autor (2025).

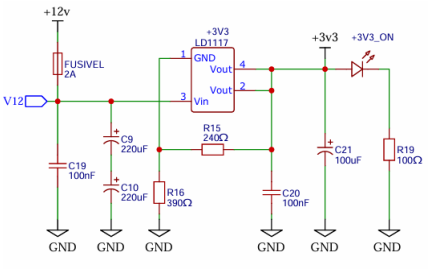
A figura 43 mostra o circuito regulador de tensão responsável por fornecer +3,3 V para a alimentação dos sensores e da tela TFT do sistema. O circuito recebe +12 V na entrada, protegido por um fusível de 2 A e estabilizado por capacitores de filtro (C9, C10, C19).

O regulador LD1117 reduz a tensão para +3,3 V, enquanto os resistores (R15 e R16) ajustam a operação interna do regulador. Capacitores adicionais (C20, C21) garantem a filtragem fina e a estabilidade da saída.

Um LED em série com resistor (R19) atua como indicador de funcionamento (+3V3_ON), confirmando a presença da tensão regulada.

Esse arranjo assegura uma fonte estável e confiável, essencial para o correto funcionamento de componentes sensíveis como sensores analógicos e a interface gráfica do display TFT.

Figura 43: Esquema Elétrico: Regulador de Tensão LD1117.

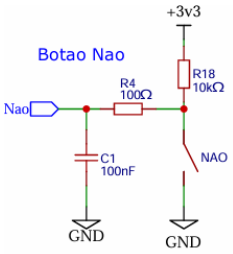


Fonte: elaborado pelo próprio Autor (2025).

4.4.12 Esquema Elétrico: Botões

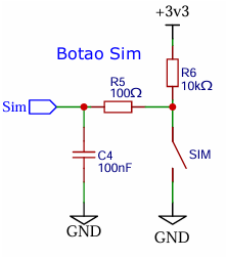
Nas figuras 44,45 e 46 os botões são usados para entrada do operador, como Sim, Não, Enter. Estão conectados com resistores *pull-up*, seguidos dos capacitores de 100nF (C1, C4 e C5) garantindo leitura estável pelo ESP32.

Figura 44: Esquema Elétrico: Botão Não



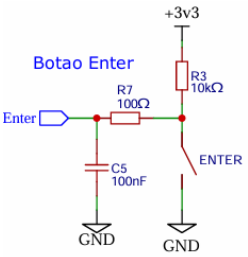
Fonte: Elaborado pelo próprio Autor (2025).

Figura 45: Esquema Elétrico: Botão Sim



Fonte: Elaborado pelo próprio Autor (2025).

Figura 46: Esquema Elétrico: Botão Enter



Fonte: Elaborado pelo próprio Autor (2025).

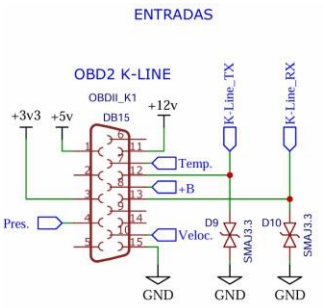
A figura 47 apresenta o esquema elétrico do módulo de interconexão baseado em um conector DB25, responsável pela entrada e distribuição dos sinais no sistema de telemetria. À esquerda observa-se a entrada de alimentação em +12 V, que é regulada e distribuída internamente para fornecer saídas de +3,3 V e +5 V utilizadas pelos sensores externos.

À direita, está representada a interface OBD2 K-Line, com suporte ao protocolo CAM, permitindo futura implementação da leitura direta da ECU automotiva. Além disso, são mostradas as conexões dedicadas aos sensores de temperatura, presença e velocidade, bem como linhas de transmissão e recepção (K-Line_TX e K-Line_RX).

Esse arranjo garante a modularidade e expansibilidade do sistema, permitindo

tanto o monitoramento de sensores dedicados quanto a integração futura com a rede de diagnóstico automotivo (OBD2).

Figura 47 – Esquema elétrico do conector DB25 e interfaces do sistema



Fonte: Elaborado pelo próprio Autor (2025).

4.5. Estrutura de Software

O firmware implementa uma máquina de estados complexa com as seguintes fases sequenciais:

4.5.1. Inicialização e Configuração

- Inicialização de periféricos (display, sensores, SD card)
- Conexão não-bloqueante com Arduino IoT Cloud
- Carregamento da senha da EEPROM
- Configuração de interrupções para sensor de velocidade

4.5.2. Autenticação do Operador

- Sistema de senha de 5 dígitos com entrada por botões
- Validação contra senha armazenada na EEPROM
- Identificação por ID numérico de 3 dígitos

4.5.3. Checklist de Segurança

- Sequência de 20 perguntas abrangentes sobre condições do equipamento
- Interface intuitiva com botões SIM/NÃO/ENTER
- Armazenamento das respostas para registro e análise
- Liberação do relay apenas após conclusão satisfatória

4.5.4. Operação Contínua

- Monitoramento em tempo real dos sensores:
- Temperatura do motor (LM35)
- Inclinação do equipamento (MPU6050)
- Tensão da bateria (divisor resistivo)
- Velocidade de deslocamento (sensor Hall)
- Detecção de presença (RCWL-0516)
- Atualização periódica para Arduino Cloud
- Registro em arquivo CSV no cartão SD
- Alertas visuais e sonoros para condições críticas

4.5.5. Ajuste de Data e Hora

- Método de Acesso

O ajuste de data e hora é realizado através da combinação simultânea dos botões SIM e NÃO por 3 segundos, que aciona a função ajustarDataHora().

4.5.6. Interface de Ajuste

O sistema apresenta uma interface sequencial para ajuste de:

Ano (formato completo 20XX)

Mês (1-12)

Dia (1-31, com verificação de validade)

Hora (0-23)

Minuto (0-59)

4.5.7. Implementação

A função ajustarDataHora() guia o usuário através das etapas de ajuste utilizando os botões SIM (incrementar), NÃO (decrementar) e ENTER (confirmar). Os valores são gravados no RTC DS3231 através das funções rtc.setYear(), rtc.setMonth(), rtc.setDate(), rtc.setHour() e rtc.setMinute().

4.6. Gestão de Energia e Conexão

O sistema implementa estratégia de reconexão não-bloqueante para Wi-Fi e Arduino IoT Cloud, garantindo operação contínua mesmo com flutuações de conexão. A verificação periódica evita travamentos do sistema.

4.6.1. Interface do Usuário

- Telas contextuais adaptadas a cada fase de operação
- Feedback visual imediato para ações do usuário
- Alertas coloridos conforme criticidade (vermelho, amarelo, verde)
- Sistema de navegação intuitivo com três botões

4.6.2. Gestão de Segurança

- Bloqueio do equipamento via relay até conclusão do checklist
- Monitoramento contínuo durante operação
- Parâmetros configuráveis de limites críticos (temperatura, inclinação, velocidade)
- Detecção de pessoas próximas com alerta específico

4.6.3. Protocolo de Comunicação

- Arduino IoT Cloud: Atualização periódica das variáveis definidas no Thing
- Formato CSV: Estrutura padronizada para análise posterior dos dados
- Protocolo I2C: Comunicação com MPU6050 e DS3231
- SPI: Comunicação com display e cartão SD

5. Custo do Projeto:

ID	Nome	Designador	Pegada	Quantidade	Valor
9	240R	R15	R0603	1	R$ 0,76
10	390R	R16	R0603	1	R$ 0,76
1	10k	R8,9,21,22,23,3,6,18	R0603	8	R$ 6,08
2	47k	R10	R0603	1	R$ 0,76
3	820R	R1	R0603	1	R$ 0,76
4	100R	R2, R19	R0603	2	R$ 1,52
5	220R	R20	R0603	1	R$ 0,76
6	100R	R4, R5, R7	R0603	3	R$ 2,28
7	3.3k	R11, R12	R0603	2	R$ 1,52
8	100nF	C2,12,14,15,16,17,19,20,23,1,4,5,6,7,11,24	C0603	16	R$ 10,40
9	100uF	C21, C22, C13	CAP-SM D_BD5.0-L5.3-W5.3-RD	3	R$ 32,52
10	220uF	C9	CAP-SM D_BD5.0-L5.3-W5.3-RD	1	R$ 10,84
11	1N4007	D1	SM A_L4.3-W2.6-LS5.2-RD	1	R$ 0,54
12	5v6	D13	DO-35_BD2.0-L4.2-P8.20-D0.5-RD	1	R$ 1,90
13	SMAJ15	D2, D8	SM A_L4.3-W2.6-LS5.1-BI	2	R$ 4,00
14	SMAJ3.3	D3, D4, D5, D9, D10	SM A_L4.3-W2.6-LS5.1-BI	5	R$ 10,00
15	BC847A	Q1,Q2	SOT-23-3_L2.9-W1.6-P1.90-LS2.8-BR	2	R$ 5,10
16	LD1117	+3V3	LD1117AG-33-AA3-A-R-SOT-223	1	R$ 14,25
17	78M05	+5V	TO-252-2L78M05	1	R$ 19,00
18	2A	FUSIVEL	FUSÍVEL-SMD_L6.1-W2.7	1	R$ 10,00
19	RY-12W-K	K1	RELÉ-TH_RY-XXW-K	1	R$ 27,10
20	DB15	OBDII_K	Plug Femea DB15	1	R$ 8,00
21	Buzzer	ALERTA	BUZ-TH_BD12.0-P7.60-D0.6-FD	1	R$ 5,23
22	Push-Buttom	ENTER, NAO, SIM	SWICHE PULSADOR 6X6X4M M	5	R$ 12,20
23	ESP32-WROOM-32 (16MB)	U2	WIFIM-SMD_ESP32-WROOM-32-N4	1	R$ 45,90
24	TFT (128X160) _ST7735	U9	TFT1,8_SPI_128X160ST7735S_M Y	1	R$ 50,46
25	Led 3MM	+3V3_ON,+5V_ON,M AQUINA_ON	LED-TH_BD3.8-P2.54_L240CTZ0G3MM	3	R$ 1,50
26	MPU6050	MPU6050	HDR-TH_9P-P2.54-H-F-W10.0-N	1	R$ 40,41
27	RTC-DS3231	RELOGIO	DS3231 OK	1	R$ 33,92
28	RCWL-0516	Sensor Presença		1	R$ 9,35
29	LM 35	Sensor Temperatura		1	R$ 23,00
30	Hall 3144E	Sensor Velocidade		1	R$ 19,90
31	Placa Circuito Impresso	PCB	JLCPCB	5	R$ 131,44
TOTAL:	**R$ 542,16**				

Fonte: Elaborado pelo próprio Autor no easyeda.com

6 CONCLUSÕES

O hardware desenvolvido alcançou plenamente os objetivos propostos pelo trabalho, proporcionando uma solução completa de telemetria e segurança para empilhadeiras. A integração entre componentes físicos, software embarcado e plataforma cloud resultou em um sistema confiável que atende aos requisitos de segurança operacional e monitoramento contínuo.

O sistema implementa todas as funcionalidades especificadas: autenticação de operadores, checklist preventivo, monitoramento em tempo real, registro de dados, alertas de segurança e integração com plataforma IoT. A arquitetura escolhida permite expansões futuras, como adição de novos sensores ou integração com sistemas de gestão de frota.

A solução desenvolvida representa um avanço significativo na segurança de equipamentos de movimentação de carga, potencialmente reduzindo acidentes e melhorando a eficiência operacional através do monitoramento sistemático de condições críticas do equipamento e do operador.

Mesmo que na execução dos sistemas de telemetria exija investimentos em infraestrutura tecnológica e na capacitação de operadores, os benefícios operacionais, econômicos e ambientais se mostram superiores aos desafios que teremos na implantação do sistema. A evolução constante da tecnologia, com sensores mais avançados e sistemas inteligentes, tem tornado esse processo cada vez mais acessível e eficiente. Dessa forma, conclui-se que a telemetria representa uma ferramenta imprescindível para organizações que buscam inovação, segurança e competitividade.

As perspectivas futuras indicam uma evolução contínua dessa tecnologia, principalmente com a incorporação de recursos como inteligência artificial (IA). Tais avanços tendem a ampliar ainda mais as possibilidades de análise preditiva e de automação, fortalecendo a gestão de equipamentos e contribuindo para operações industriais mais seguras, eficientes e sustentáveis.

7 PROJETOS FUTUROS

Este trabalho abre caminho para diversas melhorias e expansões tecnológicas, visto que podem ser implementadas em outras máquinas e equipamentos. O sistema de telemetria é eficiente e aplicável a diferentes ambientes industriais. Sendo assim há inúmeras possibilidades de projetos para o futuro, tais como:

Integração com Inteligência Artificial (IA): Desenvolver algoritmos de aprendizado para que as máquinas possam realizar análises preditivas, precisas e eficientes, permitindo antecipar falhas, otimizar rotas e melhorar o desempenho operacional dos equipamentos e de forma sustentável.

Implementação de Machine Learning: na obtenção de dados de forma contínua, é possível a criação de modelos que aprendam o comportamento padrão dos equipamentos, detectando automaticamente anomalias ou padrões de uso que possam indicar falhas iminentes.

Expansão para Outras Máquinas Industriais: Adaptar o sistema não apenas para empilhadeiras, mas também para outros equipamentos de movimentação, como guindastes, pontes rolantes, tratores, colheitadeiras e caminhões industriais.

Integração com Sistemas ERP e SCM: Incorporar o sistema de telemetria às plataformas de gestão empresarial (ERP) e gestão da cadeia de suprimentos (SCM), proporcionando uma visão mais ampla e integrada da operação.

Geolocalização Avançada: Implementar monitoramento por GPS para rastreamento em tempo real dos equipamentos, controle de acesso por áreas e segurança contra roubos ou uso indevido.

Dashboards Interativos e Relatórios Inteligentes: Desenvolver plataformas gráficas mais sofisticadas, com dashboards que permitam personalização dos dados visualizados, geração de relatórios automáticos e análises em tempo real.

Implementação de Sensores Adicionais: Ampliar a quantidade e variedade de sensores, incluindo sensores de pressão hidráulica, detecção de colisões, controle de carga transportada e monitoramento de consumo energético.

Uso de Redes LoRaWAN ou 5G: Explorar redes de comunicação de longo alcance (LoRaWAN) ou 5G para permitir operação em ambientes industriais mais amplos e com grande volume de dados, garantindo comunicação estável e de baixa latência.

Soluções Voltadas à Sustentabilidade: Incorporar funcionalidades

específicas para monitorar o consumo de energia, emissões de CO_2 e promover práticas sustentáveis dentro do sistema industrial, de suas operações.

Essas possibilidades reforçam que o projeto desenvolvido possui grande potencial de evolução, podendo ser continuamente aprimorado para atender às perspectivas do mercado e acompanhar as tendências tecnológicas do mercado.

8 REFERÊNCIAS

Usando o Display TFT 1.8" ST7735 com NodeMCU ou Arduino. 2018. Disponível em: https://portalvidadesilicio.com.br/controlando-display-tft-st7735-nodemcu/. Acesso em: 06 setembro 2025.

ADLER, D. **MPU6050 Motion Sensor Guide**. Instructables, 2020. Disponível em: https://www.instructables.com/MPU6050-Tutorial/. Acesso em: 18 maio 2025.

ADVANCED MONOLITHIC SYSTEMS. **K1, U1, U7, SW1, SW2 E SW3**. AMS1117 Datasheet. AMS, 2015.

ALLEGRO MICROSYSTEMS. **A3144 Hall Effect Sensor IC Datasheet**. Worcester, MA, 2021. Disponível em: <https://www.allegromicro.com/en/products/sense/hall-effect-sensors>. Acesso em: 18 maio 2025.

ALTRONICS. **Sensor Hall 3144E com Ímã**. 2025. Disponível em: <https://th.bing.com/th/id/OIP.zi0FIEybPB57EjCnuEYSzAAAAA?rs=1&pid=ImgDetMain>. Acesso em: 20 maio 2025.

ALTRONICS. **Sensor Hall 3144E com Ímã**. 2025. Disponível em: <https://altronics.cl/image/cache/catalog/productos/electronica/sensores/efecto_hall_ky003/sensor-hall-ky003-1-500x500.png>. Acesso em: 20 maio 2025.

ANDRADE, Danilo Cordeiro. **Supervisório de frotas DISPATCH:** eletrônica embarcada aplicada na otimização de processos de mineração. 2022. Disponível em: <https://repositorio.ufc.br/handle/riufc/72315>. Acesso em: 18 maio. 2025.

BOFF, Willian Malacarne. **Sistema de alarme conectado via redes móveis para monitoramento e segurança de veículos automotivos através de aplicativo de telefone celular**. 2017. Disponível em: <https://lume.ufrgs.br/handle/10183/172714>. Acesso em: 18 maio 2025.

BRAGATTO, Mateus. **Telemetria em Switches Programáveis com Múltiplas Filas usando Roteamento Multicaminhos na Origem**. 2024. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/29814>. Acesso em: 18 mai. 2025.

CARÁ, Cesar Augusto Führ**. Aplicação de protocolo IOT para monitoramento veicular**. 2021. Disponível em: <http://repositorio.jesuita.org.br/bitstream/handle/UNISINOS/11678/C%C3%A9sar+Augusto+F%C3%BChr+Car%C3%A1.pdf?sequence=1>. Acesso em: 18 maio 2025.

DIODES INCORPORATED. (**U8 – 3144E).** 3144 Hall-Effect Sensor Datasheet. Diodes Incorporated, 2014.

ELETROGATE. **Módulo ESP32 DevKit V1 (30 pinos).** 2018. Disponível em: <https://blog.eletrogate.com/wp-content/uploads/2018/11/ESP32-DEVKIT-Gustavo.jpg>. Acesso em: 20 maio 2025.

ELETROGATE. **Botões boot e reset**. 2018. Disponível em: <https://blog.eletrogate.com/wp-content/uploads/2018/11/ESP32-DTR-RQS.jpg>. Acesso em: 20 maio 2025.

ELETROGATE. **Sensor Inercial MPU6050**. Princípio do efeito piezo elétrico. Crédito: Maker Pro. 2020. Disponível em: <Fonte: https://blog.eletrogate.com/wp-content/uploads/2020/08/como-funciona-acelerometro.jpg>. Acesso em: 20 maio 2025.

ESPRESSIF SYSTEMS. **Módulo ESP32-C3 (U6).** ESP32-C3-DevKitC-02 Datasheet. Espressif Systems, 2022.

ESPRESSIF SYSTEMS. **ESP32-WROOM-32 Datasheet**. Disponível em: <https://www.espressif.com/en/products/socs/esp32/resources>. Acesso em: 18 maio 2025.

FACCIONI FILHO, Mauro. **Internet das coisas**. Disponível em: <https://www.researchgate.net/profile/Mauro-Fazion-Filho/publication/319881659_Internet_das_Coisas_Internet_of_Things/links/59c038d5458515e9cfd54ff9/Internet-das-Coisas-Internet-of-Things.pdf>. Acesso em: 18 maio 2025.

FUCK, Guilherme**. Proposta de estruturação do processo de montagem de dispositivos eletrônicos de telemetria automotiva e portuária**. 2024. Disponível em: <https://repositorio.ufsc.br/handle/123456789/255718>. Acesso em: 18 maio 2025.

GOMES, Mariana. **Monografia de Graduação em Engenharia de Controle e Automação**. 2022. Disponível em: <https://monografias.ufop.br/bitstream/35400000/4331/2/MONOGRAFIA_Telemetria Veicular.pdf>. Acesso em: 20 maio 2025.

GUILMO, João Antonio. **Sistema semi-intrusivo de monitoramento de energia elétrica residencial.** 2021. Disponível em: <http://riut.utfpr.edu.br/jspui/handle/1/36449>. Acesso em: 20 maio 2025.

INVENSENSE. **MPU6050(H1**). MPU-6050 Product Specification. InvenSense Inc., 2013.

JOB, Débora. **Telemetria Adaptativa Usando Aprendizado por Reforço Profundo em Redes Definidas por Software**. 2023. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/24540>. Acesso em: 20 maio 2025.

LINO, Matheus Silva de. **Sistema de telemetria e rastreamento de veículos**. 2019. Disponível em: <http://repositorio.utfpr.edu.br/jspui/handle/1/15788>. Acesso em: 18 maio 2025.

MACHADO, Cesar Abascal. **Sistema de telemetria para veículos automotores**. 2017. Disponível em: <https://repositorio.ufsm.br/handle/1/25254>. Acesso em: 18 maio 2025.

MAKERGUIDES. **Gráfico Voltage versus temperatura**. 2020. Disponível em: <https://www.makerguides.com/wp-content/uploads/2020/10/LM35-output-voltage-versus-temperature.png>. Acesso em: 20 maio 2025.

MARTINS, Leandro Ramos. **Construção de um protótipo de barco robô de baixo custo para análise de qualidade de água em rios e reservatórios**. 2020. Disponível em: <http://repositorio.ufpa.br/handle/2011/12745>. Acesso em: 18 maio 2025.

MEDEIROS, Cleiton Baptista; PONGELUPPE FILHO, Eduardo; PAIVA, José Mauro de. **Gestão energética:** estudos e alternativas para o uso eficiente e sustentável do combustível no transporte rodoviário. 2024. Disponível em: <https://repositorio.itl.org.br/jspui/handle/123456789/774>. Acesso em: 18 maio 2025.

MENEGHINI, Samuel. **Sistema de advertência na operação de tratores**. 2023. Disponível em: <https://repositorio.ucs.br/xmlui/handle/11338/12043>. Acesso em: 18 maio 2025.

MIOTTO, João Vitor Araújo. **Desenvolvimento de sistema de telemetria para protótipo de carro elétrico aplicado a competições de eficiência energética**. 2023. Disponível em: <https://repositorio.ufsc.br/handle/123456789/248465>. Acesso em: 19 maio 2025.

NEXPERIA. (**U2)**. BC847 Series Datasheet. Nexperia, 2020.

NOVA ELETRÔNICA. **Sensor de Temperatura LM35**. 2025.Disponível em: <https://th.bing.com/th/id/R.5b3d0cc6fe643bc342470a35f6117897?rik=FPbpUlUNohVT7w&riu=http%3a%2f%2fblog.novaeletronica.com.br%2fimg%2fsensor-lm35.gif&ehk=DxmWQVOis5tx87inDECNq7pb7zFuKdfOVw0kyrgir%2bQ%3d&risl=&pid=ImgRaw&r=0>. Acesso em: 20 maio 2025.

OLIVEIRA, André Luís Araújo. **Prototipagem para monitoramento e análise de sistema de fluxo de ar em galerias de minas subterrâneas com o uso de sistemas embarcados, módulos e sensores**. Disponível em: <https://repositorio.ufc.br/handle/riufc/80241>. Acesso em: 20 maio 2025.

OLIVEIRA, Matheus Silva de. S**istema de monitoramento e gerenciamento para microrredes residenciais dotadas de geração solar fotovoltaica**. 2023. Disponível em: <https://repositorio.ufpe.br/handle/123456789/50259>. Acesso em: 19 maio 2025.

OLIVEIRA, Rubens Moreno Alves. **Sistema embarcado para monitoramento automotivo em tempo real**. 2022. Disponível em: <https://www.eng-

automacao.araxa.cefetmg.br/wp-content/uploads/sites/152/2022/08/TCC2-Rubens-Moreno-de-Oliveira.pdf>. Acesso em: 18 maio 2025.

PINO, J. **Internet das Coisas com ESP32:** Programação prática com Arduino IDE. São Paulo: Novatec, 2018.

PINTO, Hyoran Spessatto. **Desenvolvimento de dispositivo de medição e aquisição de dados para veículo elétrico**. 2017. Disponível em: <http://repositorio.utfpr.edu.br/jspui/handle/1/15059>. Acesso em: 18 maio 2025.

POZZO, Luciano Augusto. **Big data e suas vantagens no ramo de monitoramento de veículos**. 2017. Disponível em: <http://riut.utfpr.edu.br/jspui/handle/1/19417>. Acesso em: 19 maio 2025.

REIS, Ícaro Uzeda; COSTA, João Marcelo; SANTANA, Juliana Maria; SAPUCAIA, Rodrigo Travasso. **Otimização do plano de manutenção de uma petrolífera visando a melhoria da produtividade e confiabilidade do sistema**. 2019. Disponível em: <http://repositoriosenaiba.fieb.org.br/bitstream/fieb/1732/1/Theoprax_%C3%8DCARO%20UZEDA%20REIS%20et%20al.pdf>. Acesso em: 18 maio 2025.

SANTOS, Mauricio Ribeiro. **Programação embarcada para rastreamento de veículos de transporte terrestre**. Revista Eletrônica e-Fatec, 2018. Disponível em: <https://pesquisafatec.com.br/ojs/index.php/efatec/article/view/130>. Acesso em: 20 maio 2025.

SILVA, Fernando Paim da. **Sistema para gerenciamento de frotas automotivas baseado em IOT**. 2021. Disponível em: <https://repositorio.ufsm.br/handle/1/21341>. Acesso em: 18 maio 2025.

SILVA, Luís Fernando Ferreira da; AMICI, Thiago Tadeu. **Sistema remoto de monitoramento de veículos auto guiados**. 2022. Disponível em: <https://revistabrmecatronica.sp.senai.br/ojs/index.php/revistabrmecatronica/article/view/166>. Acesso em: 20 maio 2025.

SILVEIRA, Saulo Lima da. **Sistema de sensores para monitoramento de peso em veículos de carga**. 2021. Disponível em: <http://riut.utfpr.edu.br/jspui/handle/1/29953>. Acesso em: 19 maio 2025.

STEWART, J. **Smart Home Automation with ESP32:** A practical guide to designing IoT solutions. New York: Maker Media, 2019.

TEXAS INSTRUMENTS. **LM35 Precision Centigrade Temperature Sensors**. Dallas, 2022. Disponível em: <https://www.ti.com/product/LM35>. Acesso em: 18 maio 2025.

ANEXO

```
/*
  Sketch gerado pelo Arduino IoT Cloud Thing "Untitled"
  https://create.arduino.cc/cloud/things/252b03c0-1674-4144-a69a-6962a560fdee

  Descrição das variáveis da nuvem IoT do Arduino:
  - As variáveis abaixo são automaticamente atualizadas quando alteradas no Thing
*/

/*
 * SISTEMA DE TELEMETRIA PARA EMPILHADEIRAS COM ARDUINO CLOUD
 * ----------------------------------------------------
 *
 * DESCRIÇÃO:
 * Sistema completo de telemetria e segurança para empilhadeiras com:
 * - Autenticação por senha de 5 dígitos
 * - Checklist operacional interativo
 * - Monitoramento de sensores (temperatura, inclinação, tensão da bateria,
presença e velocidade)
 * - Comunicação via Arduino Cloud
 * - Interface com display ST7735
 * - Controle de relé para liberação do equipamento
 * - Armazenagem de dados local, cartão sd
 *
 * AUTOR: Marco Aurélio Alves
 * DATA: 14/06/2025
 * VERSÃO: 1.3
 *
 * HARDWARE:
 * - ESP32 DOIT DEVKIT V1
 * - Display ST7735
 * - Sensor MPU6050 (Acelerômetro/Giroscópio)
 * - Sensor LM35 (Temperatura)
 * - Sensor 3144 Hall (Velocidade)
 * - Sensor RCWL-0516 (Presença)
 * - Sensor e Relógio RTC-DS3231
 * - Cartão de memória
 * - Divisor de tensão para medição da bateria
 * - Buzzer para alertas sonoros
 * - Relé para controle do sistema
 * - 3 botões para interface (SIM, NÃO, ENTER)
 */

// ======================== BIBLIOTECAS ========================
#include "thingProperties.h"       // Gerada automaticamente pelo Arduino IoT Cloud
#include <Wire.h>                  // Para comunicação I2C
#include <Adafruit_MPU6050.h>      // Para sensor MPU6050
#include <DS3231.h>                // Relógio
#include <Adafruit_Sensor.h>       // Base para sensores Adafruit
```

```
#include <Adafruit_GFX.h>          // Gráficos para displays
#include <Adafruit_ST7735.h>       // Biblioteca do display ST7735
#include <SPI.h>                   // Comunicação SPI
#include <SD.h>                    // Comunicação Cartão SD
#include <EEPROM.h>                // Para armazenamento da senha
#include <WiFi.h>
#include <WiFiClientSecure.h>

// ================= CONFIGURAÇÕES GERAIS ================
#define BATERIA_MIN 10.0           // Tensão mínima da bateria (em volts)
#define BATERIA_MAX 14.2           // Tensão máxima da bateria (em volts)
#define TEMP_MAX 95.0              // Temperatura máxima permitida (°C)
#define INCLINACAO_MAX 85.0        // Inclinação máxima permitida (graus)
#define INCLINACAO_MIN 50.0        // Inclinação minima permitida (graus)
#define VELOC_MAX 20.0             // Velocidade máxima permitida (km/h)
#define HISTORICO_SIZE 20          // Tamanho do histórico de dados
#define TAMANHO_SENHA 5            // Tamanho da senha (5 dígitos)
#define RAIO_RODA 0.2              // Raio da roda em metros
#define PULSOS_POR_VOLTA 4         // Pulsos por volta completa do sensor Hall

// Pinos para display ST7735
#define TFT_CS     5               // Pino Chip Select do display
#define TFT_RST    2               // Pino Reset do display
#define TFT_DC     4               // Pino Data/Command do display
#define TFT_SCLK   18              // Pino Clock SPI do display
#define TFT_MOSI   23              // Pino MOSI SPI do display

// Pinos para SD Card
#define SD_CS      15              // Pino Chip Select do cartão SD
#define SD_MISO    19              // Pino MISO do cartão SD

// Pinos para DS3231
#define PIN_SDA    21              // Pino SDA para I2C (RTC)
#define PIN_SCL    22              // Pino SCL para I2C (RTC)

// ================= DEFINIÇÃO DOS PINOS ================
const int pinoLM35 = 34;           // Pino do sensor de temperatura LM35
const int pinoHall = 35;           // Pino do sensor de velocidade Hall 3144
const int pinoBateria = 36;        // Pino para medição da tensão da bateria
const int pinoBuzzer = 25;         // Pino do buzzer para alertas sonoros
const int pinoRele = 26;           // Pino do relé para liberação do sistema
const int botaoSim = 14;           // Pino do botão SIM (com pull-up interno)
const int botaoNao = 27;           // Pino do botão NÃO (com pull-up interno)
const int botaoEnter = 12;         // Pino do botão ENTER (com pull-up interno)
const int pinoPresenca = 13;       // Pino sensor de presença RCWL-0516

// ================= CONFIGURAÇÕES ADC ================
constexpr float VDIV_BATT = 6.06f;        // Fator de divisão de tensão da bateria
constexpr float ALPHA_VBAT = 0.20f;       // Fator de suavização para bateria (0-1)
```

```
constexpr float ALPHA_TEMP = 0.15f;       // Fator de suavização para temperatura (0-1)
constexpr float T_GAIN = 1.000f;          // Ganho para calibração do LM35 (ajuste se necessário)
constexpr float T_OFFS = 0.0f;            // Offset para calibração do LM35 (ajuste se necessário)

// Protótipos de funções
float adcReadTrimmedMean(int pin, uint8_t N = 32, uint8_t K = 4);

// ================ CONFIGURAÇÃO DA SENHA ================
int senhaDigitada[TAMANHO_SENHA] = {0};  // Array para armazenar a senha digitada pelo usuário
int senhaCorreta[TAMANHO_SENHA] = {1, 2, 3, 4, 5};  // Senha padrão do sistema
int digitoAtual = 0;              // Índice do dígito atual sendo digitado

// ================ GERA ÍCONE DE ALERTA ================
int cx = 80;                      // Posição X do centro do triângulo de alerta
int cy = 45;                      // Posição Y do centro do triângulo de alerta
int halfBase = 10;                // Metade da base do triângulo de alerta
int triHeight = 10;               // Altura do triângulo de alerta

// ==================== OBJETOS ====================
Adafruit_ST7735 display = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST); // Objeto do display ST7735
DS3231 rtc;                       // Objeto do relógio RTC DS3231
Adafruit_MPU6050 mpu;             // Objeto do sensor MPU6050 (acelerômetro/giroscópio)
File logFile;                     // Objeto para manipulação de arquivos no cartão SD

//================ VARIÁVEIS GLOBAIS ================
bool sistemaAtivo = false;        // Controla se o sistema está ativo ou não
int perguntaAtual = 0;            // Índice da pergunta atual do checklist
bool senhaValidada = false;       // Indica se a senha foi validada com sucesso
float velocidadeAtual = 0;        // Velocidade atual em km/h
String operadorAtual = "OPERADOR"; // Nome ou identificação do operador atual
bool respostasChecklist[20];      // Array para armazenar as respostas do checklist (20 perguntas)
bool century = false;             // Flag para século no RTC (não utilizado no DS3231)
bool h12Flag;                     // Flag para formato de 12 horas no RTC
bool pmFlag;                      // Flag para período PM no RTC
String nomeArquivoLog = "";       // Nome do arquivo de log atual
unsigned long tempoUltimoLog = 0;  // Timestamp do último registro no log
const unsigned long intervaloLog = 5000; // Intervalo entre registros no log (5 segundos)
float tempRTC;                    // Temperatura lida do RTC
bool checklistImpresso = false;   // Indica se o checklist já foi impresso no log
bool operadorImpresso = false;    // Indica se o operador já foi impresso no log
int operadorID = 0;               // ID numérico do operador
bool modoAjusteHorario = false;   // Controla o modo de ajuste de data/hora
```

```
int etapaAjuste = 0;              // Etapa atual do ajuste de data/hora
bool pessoaDetectada = false;     // Estado atual de detecção de pessoa
int contadorPresencas = 0;        // Contador de detecções de presença

// ---- Contagem de pulsos do Hall (mais estável) ----
volatile uint32_t hallPulseCount = 0;  // total acumulado de pulsos
volatile uint32_t lastPulseMicros = 0;  // timestamp do último pulso (µs)

// Janelas/tempos (ajuste fino se quiser)
constexpr uint32_t HALL_GUARD_US     = 500;    // 0,5 ms: ignorar glitches muito
próximos
constexpr uint32_t VEL_WINDOW_US     = 400000; // 400 ms: janela de contagem
constexpr uint32_t STOP_TIMEOUT_US  = 800000; // 0,8 s sem pulso =>
velocidade = 0
constexpr float    ALPHA_VEL         = 0.25f;  // filtro EWMA (25% amostra nova)

// ---- Valores locais para EXIBIÇÃO (não são Cloud) ----
volatile float ui_vbat = 0.0f;
volatile float ui_tlm35 = 0.0f;
volatile float ui_incl  = 0.0f;
volatile float ui_vel   = 0.0f;

// ---- Estado do MPU ----
bool mpuOK = false;

// ========== CONFIGURAÇÕES DE DISPLAY SEM CINTILAÇÃO ==========
static bool statusLayoutPronto = false;

// Calcular a coluna dos valores dinamicamente.
static int X_LABEL = 4;
static int X_VAL   = 64;  // valor padrão; será recalculado no desenho

// Linhas fixas (y)
static const int Y_TIT  = 0;
static const int Y_HORA  = 20;
static const int Y_OPID  = 36;
static const int Y_BATT  = 52;
static const int Y_TLM35 = 68;
static const int Y_TRTC  = 84;
static const int Y_INCL  = 100;
static const int Y_VEL   = 116;

// Caches de texto (evita redesenho se igual)
static char cacheHora[24]  = "";
static char cacheOpId[24]  = "";
static char cacheBatt[24]  = "";
static char cacheTLM35[24] = "";
static char cacheTRTC[24]  = "";
static char cacheIncl[24]  = "";
static char cacheVel[24]   = "";
```

```
// thresholds/histéreses para não "piscar"
static const float H_BATT = 0.02f; // 20 mV
static const float H_TEMP = 0.10f; // 0.1 °C
static const float H_INCL = 0.20f; // 0.2 °
static const float H_VEL  = 0.20f; // 0.20 km/h

// Agendadores por campo
static uint32_t tHora = 0, tBatt = 0, tTemp = 0, tIncl = 0, tVel = 0;

// Protótipos das funções
float lerTemperatura();
float lerInclinacao();
float obterVelocidade();
float lerTensaoBateria();
void mostrarAlerta(String tipoAlerta, float valor);
static void desenharLayoutStatus();
void mostrarDebugSensores();
void registrarNoSD();
void IRAM_ATTR detectarPulsoHall();
void inicializarSensorHall();
void verificarCombinacaoBotoes();
void ajustarDataHora();
void setupADC();

// "últimos valores" usados pela histerese (mantidos fora da função e resetados no
layout)
static float lastBatt = NAN, lastLM = NAN, lastRTC = NAN, lastInc = NAN, lastVel =
NAN;

// Função segura com verificação de tamanho do buffer
static void printIfChanged(int x, int y, char *cache, size_t cachesz, const char *novo) {
  if (strncmp(cache, novo, cachesz - 1) != 0) {
    display.setCursor(x, y);
    display.setTextColor(ST77XX_YELLOW, ST77XX_BLACK);
    display.print(novo);
    strncpy(cache, novo, cachesz - 1);
    cache[cachesz - 1] = '\0';
  }
}

// calcula X_VAL para caber o maior campo (data). tamanho máximo: 16 chars (~96
px)
// usamos "DD/MM/AA HH:MM" = 14 chars (~84 px) para sobrar margem.
static void ajustarColunas() {
  int W = display.width();        // no ST7735 costuma ser 160 em landscape
  const int larguraValoresPx = 84 + 4; // 14 chars * 6 + margem (4)
  // garante ao menos 2 colunas: rótulo à esquerda e valores à direita:
  X_VAL = max(48, W - larguraValoresPx);
}
```

```
// zera caches + últimos valores (força 1ª pintura)
static void resetStatusCaches() {
  cacheHora[0] = cacheOpId[0] = cacheBatt[0] = cacheTLM35[0] =
  cacheTRTC[0] = cacheIncl[0] = cacheVel[0] = '\0';
  lastBatt = lastLM = lastRTC = lastInc = lastVel = NAN;
}

// ================= PERGUNTAS DO CHECKLIST ================
const String perguntasChecklist[] = {
  "         FREIOS E\n\n"
  "    ESTACIONAMENTO OK?",         // Pergunta 1 - Freios e estacionamento
  "      AVISOS SONOROS\n\n"
  "       FUNCIONANDO?",            // Pergunta 2 - Avisos sonoros
  "        GIROFLEX E\n\n"
  "       RED ZONE OK?",            // Pergunta 3 - Giroflex e red zone
  "     PDS FUNCIONANDO?",          // Pergunta 4 - PDS (Painel de Segurança)
  "       FAROIS, RE E\n\n"
  "        SETAS OK?",              // Pergunta 5 - Faróis, ré e setas
  "    VAZAMENTO DE OLEO?",         // Pergunta 6 - Vazamento de óleo
  "  BOTAO EMERGENCIA OK?",         // Pergunta 7 - Botão de emergência
  "      BATERIA COM FIO\n\n"
  "         EXPOSTO?",             // Pergunta 8 - Fios da bateria expostos
  "     AVISOS VISIVEIS?",          // Pergunta 9 - Avisos visíveis
  "    PNEUS COM DESGASTE?",        // Pergunta 10 - Desgaste de pneus
  "    SINAIS DE COLISAO?",         // Pergunta 11 - Sinais de colisão
  "  ESPELHOS DANIFICADOS?",        // Pergunta 12 - Espelhos danificados
  "    SINTO DE SEGURANCA\n\n"
  "           OK?",                // Pergunta 13 - Cinto de segurança
  "    EQUIPAMENTO LIMPO?",         // Pergunta 14 - Limpeza do equipamento
  "  TORRE MOVIMENTA OK?",          // Pergunta 15 - Movimento da torre
  "     GARFO MOVIMENTA\n\n"
  "        SEM RUIDO?",            // Pergunta 16 - Movimento do garfo sem ruído
  "    DIRECAO COM FOLGA?",         // Pergunta 17 - Folga na direção
  "    GRADE E TORRE OK?",          // Pergunta 18 - Estado da grade e torre
  "   USANDO MEDICAMENTO?",         // Pergunta 19 - Uso de medicamentos
  "  CONDICOES FISICAS OK?"         // Pergunta 20 - Condições físicas do operador
};

//================== FUNÇÕES DE ÁUDIO ==================

// Função para gerar um beep curto
void beep(int duracao) {
  digitalWrite(pinoBuzzer, HIGH);  // Ativa o buzzer
  delay(duracao);                  // Mantém ativo pelo tempo especificado
  digitalWrite(pinoBuzzer, LOW);   // Desativa o buzzer
}

// Função para gerar alertas sonoros
void alertaBuzzer(int bips) {
```

```
  for (int i = 0; i < bips; i++) {
    digitalWrite(pinoBuzzer, HIGH);  // Ativa o buzzer
    delay(200);                      // Tempo ativo
    digitalWrite(pinoBuzzer, LOW);   // Desativa o buzzer
    delay(200);                      // Tempo inativo
  }
}

// ================= FUNÇÕES DO SISTEMA =================

// Inicializa o display
void inicializarDisplay() {
  SPI.begin(TFT_SCLK, SD_MISO, TFT_MOSI);  // Inicializa SPI com os pinos
definidos
  display.initR(INITR_BLACKTAB);       // Inicializa o ST7735 com o layout comum
  display.setRotation(3);              // Ajusta a orientação se necessário
  display.fillScreen(ST77XX_BLACK);    // Limpa a tela com fundo preto
  display.setTextSize(1);              // Define tamanho do texto
  display.setTextColor(ST77XX_YELLOW); // Define cor do texto
  display.setCursor(0, 30);            // Posiciona cursor
  display.println("");                 // Linha em branco
  display.println("");                 // Linha em branco
  display.println("");                 // Linha em branco
  display.println("  Iniciando sistema..."); // Mensagem de inicialização
  delay(1000);                         // Aguarda 1 segundo
  display.fillScreen(ST77XX_BLACK);    // Limpa a tela novamente
}

// ================ Inicializa e carrega a senha da EEPROM ===============
void inicializarSenha() {

  // Verifica se já foi inicializada antes
  byte flag = EEPROM.read(0);       // Lê flag de inicialização

  if(flag != 255) {                 // Se já foi inicializada
    // Carrega senha da EEPROM
    for(int i = 0; i < TAMANHO_SENHA; i++) {
      senhaCorreta[i] = EEPROM.read(i+1); // Lê cada dígito da senha
    }
  } else {
    // Grava senha padrão na EEPROM
    for(int i = 0; i < TAMANHO_SENHA; i++) {
      EEPROM.write(i+1, senhaCorreta[i]); // Grava cada dígito da senha
    }
    EEPROM.write(0, 1);             // Marca como inicializada
    EEPROM.commit();                // Salva as alterações na EEPROM
  }
}

// ================ Mostra a tela de digitação da senha ==============
```

```
void mostrarTelaSenha() {
  display.fillScreen(ST77XX_BLACK); // Limpa a tela com fundo preto
  display.setTextSize(1);           // Define tamanho do texto
  display.setTextColor(ST77XX_WHITE); // Define cor do texto
  display.setCursor(0, 5);          // Posiciona cursor
  display.println("");              // Linha em branco
  display.println(" DIGITE A SENHA:"); // Instrução para usuário
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco

  // Exibe cada dígito da senha
  for(int i = 0; i < TAMANHO_SENHA; i++) {
    if(i < digitoAtual) {
      display.print("*");          // Dígitos já digitados como asterisco
    } else if(i == digitoAtual) {
      display.print(senhaDigitada[i]);  // Dígito atual como número
    } else {
      display.print("   _");       // Dígitos faltantes como sublinhado
    }
    display.print(" ");            // Espaço entre dígitos
  }
  display.println("");             // Nova linha
}

// Processa a entrada da senha pelos botões
void processarSenha() {
  static unsigned long ultimoDebounce = 0;  // Timestamp do último debounce
  const unsigned long intervaloDebounce = 200;  // Tempo para evitar bounce

  if(millis() - ultimoDebounce < intervaloDebounce) return; // Aguarda intervalo de
debounce

  // Botão SIM - incrementa dígito atual
  if(digitalRead(botaoSim) == LOW) {
    senhaDigitada[digitoAtual] = (senhaDigitada[digitoAtual] + 1) % 10; // Incrementa
dígito (0-9)
    ultimoDebounce = millis();     // Atualiza timestamp do debounce
    beep(50);                      // Feedback sonoro
    mostrarTelaSenha();            // Atualiza display
    return;
  }

  // Botão NÃO - decrementa dígito atual
  if(digitalRead(botaoNao) == LOW) {
    senhaDigitada[digitoAtual] = (senhaDigitada[digitoAtual] + 9) % 10; // Decrementa
dígito (0-9)
    ultimoDebounce = millis();     // Atualiza timestamp do debounce
    beep(50);                      // Feedback sonoro
    mostrarTelaSenha();            // Atualiza display
    return;
```

```
}

// Botão ENTER - confirma dígito ou senha
if(digitalRead(botaoEnter) == LOW) {
beep(100);                    // Feedback sonoro
  digitoAtual++;                // Avança para próximo dígito

  // Verifica se completou a senha
  if(digitoAtual >= TAMANHO_SENHA) {
    bool senhaOk = true;          // Assume que senha está correta

    // Compara senha digitada com a correta
    for(int i = 0; i < TAMANHO_SENHA; i++) {
      if(senhaDigitada[i] != senhaCorreta[i]) {
        senhaOk = false;          // Senha incorreta
        break;                    // Sai do loop
      }
    }

    if(senhaOk) {
      senhaValidada = true;       // Marca senha como validada
      display.fillScreen(ST77XX_BLACK); // Limpa tela
      display.setCursor(30, 0);  // Posiciona cursor
      display.println("");        // Linha em branco
      display.println("");        // Linha em branco
      display.println("");        // Linha em branco
      display.println("");        // Linha em branco
      display.println("");        // Linha em branco
      display.setTextColor(ST77XX_GREEN); // Texto verde
      display.println("       SENHA CORRETA!     "); // Mensagem de sucesso
      delay(1000);                // Aguarda 1 segundo
    } else {
      display.fillScreen(ST77XX_BLACK); // Limpa tela
      display.setCursor(30, 0);  // Posiciona cursor
      display.println("");        // Linha em branco
      display.println("");        // Linha em branco
      display.println("");        // Linha em branco
      display.println("");        // Linha em branco
      display.println("");        // Linha em branco
      display.setTextColor(ST77XX_RED); // Texto vermelho
      display.println("      SENHA INCORRETA     "); // Mensagem de erro
      delay(1000);                // Aguarda 1 segundo
      digitoAtual = 0;            // Reinicia índice do dígito
      // Reseta senha digitada
      for(int i = 0; i < TAMANHO_SENHA; i++) {
        senhaDigitada[i] = 0;     // Zera todos os dígitos
      }
      mostrarTelaSenha();         // Mostra tela de senha novamente
    }
  } else {
```

```
      mostrarTelaSenha();          // Mostra próximo dígito
    }

    ultimoDebounce = millis();      // Atualiza timestamp do debounce
  }
}

// ================ Mostra mensagem inicial do sistema ===============
void mostrarMensagemInicial() {
  display.fillScreen(ST77XX_BLACK); // Limpa tela com fundo preto
  display.setTextSize(1);           // Tamanho do texto
  display.setTextColor(ST77XX_GREEN); // Cor do texto
  display.setCursor(30, 0);         // Posiciona cursor
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco
  display.println("  SISTEMA DE TELEMETRIA\n"); // Título do sistema
  display.println("  PARA MONITORAMENTO E\n "); // Continuação
  display.println("       SEGURANCA EM\n     "); // Continuação
  display.println("      EQUIPAMENTOS DE\n  "); // Continuação
  display.println("       MOVIMENTACAO\n      "); // Continuação
  delay(8000);                      // Aguarda 8 segundos

  display.fillScreen(ST77XX_BLACK); // Limpa tela
  display.setTextColor(ST77XX_YELLOW); // Nova cor do texto
  display.setTextSize(1);           // Tamanho do texto
  display.setCursor(10, 10);        // Posiciona cursor

  display.fillScreen(ST77XX_BLACK); // Limpa tela
  display.setTextSize(1);           // Tamanho do texto
  display.setTextColor(ST77XX_GREEN); // Cor do texto
  display.setCursor(30, 0);         // Posiciona cursor
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco
  display.println("  ELETRONICA AUTOMOTIVA"); // Informação institucional
  display.println("");              // Linha em branco
  display.println("       FATEC TAUBATE    "); // Informação institucional
  delay(5000);                      // Aguarda 5 segundos

  display.fillScreen(ST77XX_BLACK); // Limpa tela
  display.setCursor(20, 0);         // Posiciona cursor
  display.setTextColor(ST77XX_CYAN); // Cor do texto
  display.setTextSize(1);           // Tamanho do texto
  display.println("");              // Linha em branco
  display.println("       INTEGRANTES      "); // Título integrantes
  display.println(" _____________________  "); // Linha separadora
  display.println("");              // Linha em branco
```

```
  display.setTextColor(ST77XX_WHITE); // Cor do texto
  display.println("      GILMAR APARECIDO    "); // Nome integrante
  display.println("");              // Linha em branco
  display.println("     JULIO CESAR VALIM    "); // Nome integrante
  display.println("");              // Linha em branco
  display.println("    MARCO AURELIO ALVES   "); // Nome integrante
  display.println("");              // Linha em branco
  display.println("    RAFAEL PIRES TOSETTO  "); // Nome integrante
  display.println("");              // Linha em branco
  display.println("   RODRIGO CARDOSO MUNIZ  "); // Nome integrante

  delay(5000);                      // Aguarda 5 segundos
}

// ================ IDENTIFICAR OPERADOR ================
void identificarOperador() {
  display.fillScreen(ST77XX_BLACK);
  display.setCursor(10, 10);
  display.setTextSize(1);
  display.setTextColor(ST77XX_YELLOW);
  display.println("ID OPERADOR (3 DIGITOS):");

  display.setTextColor(ST77XX_WHITE);
  int digitos[3] = {0, 0, 0};
  int atual = 0;
  int ultimoValor = -1;

  unsigned long ultimoUpdate = millis();

  while (atual < 3) {
    if (digitos[atual] != ultimoValor) {
      display.fillRect(0, 30, 128, 20, ST77XX_BLACK);
      display.setCursor(0, 30);
      for (int i = 0; i < 3; i++) {
        if (i == atual) display.print("[");
        display.print(digitos[i]);
        if (i == atual) display.print("]");
        else display.print(" ");
      }
      ultimoValor = digitos[atual];
    }

    if (digitalRead(botaoSim) == LOW) {
      digitos[atual] = (digitos[atual] + 1) % 10;
      beep(50);
      delay(200);
    } else if (digitalRead(botaoNao) == LOW) {
      digitos[atual] = (digitos[atual] + 9) % 10;
      beep(50);
      delay(200);
```

```
    } else if (digitalRead(botaoEnter) == LOW) {
      atual++;
      ultimoValor = -1;
      beep(100);
      delay(200);
    }

    // Pequena pausa para não sobrecarregar o processador
    delay(10);
  }

  operadorID = digitos[0] * 100 + digitos[1] * 10 + digitos[2];
  operadorAtual = "ID " + String(operadorID);

  // Mostra confirmação por apenas 2 segundo
  display.fillScreen(ST77XX_BLACK);
  display.setCursor(10, 30);
  display.setTextColor(ST77XX_GREEN);
  display.print("OPERADOR ");
  display.setTextColor(ST77XX_WHITE);
  display.print(operadorID);
  display.setTextColor(ST77XX_GREEN);
  display.println(" CONFIRMADO");
  delay(2000);
}

// ================ Mostra a pergunta atual do checklist ================
void mostrarPerguntaChecklist() {
  display.fillScreen(ST77XX_BLACK); // Limpa tela
  display.setTextColor(ST77XX_YELLOW); // Cor do texto
  display.setCursor(10, 10);        // Posiciona cursor
  display.println("CHECKLIST:");     // Título
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco
  display.setTextColor(ST77XX_WHITE); // Cor do texto
  display.println(perguntasChecklist[perguntaAtual]); // Mostra pergunta atual
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco
  display.println("");              // Linha em branco
  display.setTextColor(ST77XX_YELLOW); // Cor do texto
  display.println("       SIM        NAO"); // Opções de resposta
}

// Processa as respostas do checklist
void processarChecklist() {
  if (digitalRead(botaoSim) == LOW) {
    respostasChecklist[perguntaAtual] = true; // Resposta SIM
    avancarChecklist();           // Avança para próxima pergunta
    beep(50);                     // Feedback sonoro
```

```
    delay(200);                  // Debounce simples
  }
  else if (digitalRead(botaoNao) == LOW) {
    respostasChecklist[perguntaAtual] = false; // Resposta NÃO
    avancarChecklist();          // Avança para próxima pergunta
    beep(50);                    // Feedback sonoro
    delay(200);                  // Debounce simples
  }
}

// Avança para próxima pergunta ou finaliza checklist
void avancarChecklist() {
  perguntaAtual++;               // Incrementa índice da pergunta

  if (perguntaAtual < 20) {
    mostrarPerguntaChecklist();  // Mostra próxima pergunta
  } else {
    // Checklist completo
    checklistCompleto = true;    // Atualiza variável na nuvem

    display.fillScreen(ST77XX_BLACK); // Limpa tela
    display.setCursor(0, 30);    // Posiciona cursor
    display.println("");         // Linha em branco
    display.println("");         // Linha em branco
    display.println("");         // Linha em branco
    display.setTextColor(ST77XX_YELLOW); // Cor do texto
    display.println("    CHECKLIST CONCLUIDO"); // Mensagem de conclusão
    delay(3000);                 // Aguarda 3 segundos

    digitalWrite(pinoRele, HIGH);  // Libera empilhadeira (ativa relé)
    beep(200);                   // Feedback sonoro

    display.fillScreen(ST77XX_BLACK); // Limpa tela
    display.setCursor(0, 20);    // Posiciona cursor
    display.println("");         // Linha em branco
    display.println("");         // Linha em branco
    display.println("");         // Linha em branco
    display.println("");         // Linha em branco
    display.println("");         // Linha em branco
    display.setTextColor(ST77XX_CYAN); // Cor do texto
    display.println("       EMPILHADEIRA"); // Mensagem de liberação
    display.setCursor(0, 40);    // Posiciona cursor
    display.println("         LIBERADA"); // Mensagem de liberação
    delay(3000);                 // Aguarda 3 segundos
  }
}

// ================= FUNÇÕES DE CONEXÃO ====================
// Função para verificar e reconectar ao Wi-Fi e Arduino Cloud
void verificarConexaoWiFi() {
```

```
  static unsigned long ultimaVerificacao = 0;
  const unsigned long intervaloVerificacao = 10000;

  if (millis() - ultimaVerificacao < intervaloVerificacao) {
    return;
  }
  ultimaVerificacao = millis();

  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("Wi-Fi desconectado. Tentando reconectar...");
    reconectarWiFi();
  }
}

// Função para reconectar ao Wi-Fi (não-bloqueante)
void reconectarWiFi() {
  static unsigned long ultimaTentativa = 0;
  const unsigned long intervaloTentativaWiFi = 30000; // 30 segundos entre tentativas

  if (millis() - ultimaTentativa < intervaloTentativaWiFi) {
    return;
  }
  ultimaTentativa = millis();

  Serial.println("Reconectando ao Wi-Fi...");
  WiFi.disconnect();
  delay(100);
  WiFi.reconnect();
}

// Função para verificar conexão Cloud (não-bloqueante)
void verificarConexaoCloud() {
  static unsigned long ultimaVerificacao = 0;
  const unsigned long intervaloVerificacao = 15000;

  if (millis() - ultimaVerificacao < intervaloVerificacao) {
    return;
  }
  ultimaVerificacao = millis();

  // Só tenta reconectar Cloud se Wi-Fi estiver conectado
  if (WiFi.status() == WL_CONNECTED && !ArduinoCloud.connected()) {
    Serial.println("Reconectando ao Arduino Cloud...");
    ArduinoCloud.disconnect();
    delay(100);
    ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  }
}

//================== FUNÇÕES DE SENSORES ==================
```

```
// Monitora todos os sensores e aciona alertas se necessário
void monitorarSensores() {
  // 1) Lê sensores para variáveis locais
  float tempLocal  = lerTemperatura();
  float inclLocal  = lerInclinacao();
  float velLocal   = obterVelocidade();
  float battLocal  = lerTensaoBateria();

  ui_tlm35 = tempLocal;
  ui_incl  = inclLocal;
  ui_vel   = velLocal;
  ui_vbat  = battLocal;

  // 2) Se conectado, empurra valores LOCAIS para a Cloud
  if (ArduinoCloud.connected()) {
    temperatura    = ui_tlm35;
    inclinacao     = ui_incl;
    velocidade     = ui_vel;
    tensaoBateria  = ui_vbat;
  }

  // 3) Lógica de alertas com base nos valores LOCAIS
  if (fabs(ui_vbat) < BATERIA_MIN || fabs(ui_vbat) > BATERIA_MAX) {
    alertaBuzzer(2);
    mostrarAlerta("   Bateria\n", ui_vbat);
  }
  else if (ui_tlm35 > TEMP_MAX) {
    alertaBuzzer(4);
    mostrarAlerta("   Temperatura\n", ui_tlm35);
  }
  else if (fabs(ui_incl) > INCLINACAO_MAX || fabs(ui_incl) < INCLINACAO_MIN) {
    alertaBuzzer(3);
    mostrarAlerta("   Inclinacao\n", ui_incl);
  }
  else if (ui_vel > VELOC_MAX) {
    alertaBuzzer(5);
    mostrarAlerta("   Velocidade\n", ui_vel);
  }
  else {
    tempRTC = rtc.getTemperature();
    mostrarStatusNormal(tempRTC);
  }
}

// ============= FUNÇÃO DE CONEXÃO NÃO-BLOQUEANTE ==========
void inicializarCloudNaoBloqueante() {
  Serial.println("Iniciando conexão Arduino Cloud...");
  initProperties();

  // Configuração para debug
```

```
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  // Inicializa Cloud de forma não-bloqueante
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  // Não espera bloqueando - a verificação será feita no loop
  Serial.println("Inicialização Cloud iniciada (não-bloqueante)");
}

// Tela dedicada para alertar sobre a presença de pessoas próximas
void mostrarAlertaPresenca() {
  display.fillScreen(ST77XX_BLACK); // Limpa tela
  display.setTextColor(ST77XX_RED); // Cor do texto
  display.setTextSize(1);           // Tamanho do texto
  display.setCursor(0, 0);          // Posiciona cursor
  display.println("");              // Linha em branco
  display.println("    ALERTA DE PRESENCA "); // Título do alerta

  // Desenha triângulo de alerta (contorno)
  display.drawTriangle(cx-halfBase, cy+triHeight,
                       cx+halfBase, cy+triHeight,
                       cx,          cy-triHeight,
                       ST77XX_WHITE); // Desenha triângulo branco

  // Desenha ponto de exclamação dentro do triângulo
  display.drawLine(cx, (cy-triHeight)+5,
                   cx, (cy+triHeight)-8,
                   ST77XX_WHITE); // Desenha linha vertical
  display.fillRect(cx-1, (cy+triHeight)-5,
                   3, 3,
                   ST77XX_WHITE); // Desenha ponto inferior

  display.setCursor(0, 75);       // Posiciona cursor
  display.setTextColor(ST77XX_YELLOW); // Cor do texto
  display.println("    PESSOAS PROXIMAS AO "); // Mensagem de alerta
  display.println("       EQUIPAMENTO");     // Continuação
  display.setTextColor(ST77XX_WHITE); // Cor do texto
  display.println("");              // Linha em branco
  display.println("  Verifique a area com"); // Instrução
  display.println("         cautela");       // Instrução

  alertaBuzzer(10);               // Alerta sonoro prolongado

  delay(7000);                    // Mostra por 7 segundos

  // Reset do layout para forçar redesenho ao voltar ao status normal
  statusLayoutPronto = false;
}
```

```
// ========== FUNÇÃO: Mostrar status normal sem cintilação ============
// Chama SEMPRE no loop; ela só redesenha o que mudou
void mostrarStatusNormal(float tempRTC_in) {
  if (!statusLayoutPronto) {
    desenharLayoutStatus();
  }

  uint32_t now = millis();

  // ---------- Data/Hora (1 Hz) ----------
  if ((int32_t)(now - tHora) >= 0) {
    // Monta "DD/MM/AA HH:MM"
    int dia    = rtc.getDate();
    int mes    = rtc.getMonth(century);
    int ano2   = rtc.getYear();                  // 0..99 (DS3231)
    int hora   = rtc.getHour(h12Flag, pmFlag);
    int minuto = rtc.getMinute();

    char horaData[20];
    snprintf(horaData, sizeof(horaData),
           "%02d/%02d/%02d %02d:%02d",
           dia, mes, ano2, hora, minuto);

    printIfChanged(X_VAL, Y_HORA, cacheHora, sizeof(cacheHora), horaData);
    tHora = now + 1000;
  }

  // ---------- Operador (muda raramente) ----------
  char opStr[16];
  snprintf(opStr, sizeof(opStr), "%4d        ", operadorID);
  printIfChanged(X_VAL, Y_OPID, cacheOpId, sizeof(cacheOpId), opStr);

  // ---------- Bateria (5 Hz + histerese 20 mV) ----------
  if ((int32_t)(now - tBatt) >= 0) {
    if (isnan(lastBatt) || fabsf(ui_vbat - lastBatt) > H_BATT) {
      char s[16];
      snprintf(s, sizeof(s), "%6.2f V   ", ui_vbat);
      printIfChanged(X_VAL, Y_BATT, cacheBatt, sizeof(cacheBatt), s);
      lastBatt = ui_vbat;
    }
    tBatt = now + 200;
  }

  // ---------- Temperaturas (5 Hz + histerese 0.1 °C) ----------
  if ((int32_t)(now - tTemp) >= 0) {
    if (isnan(lastLM) || fabsf(ui_tlm35 - lastLM) > H_TEMP) {
      char s[16];
      snprintf(s, sizeof(s), "%5.1f C     ", ui_tlm35);
      printIfChanged(X_VAL, Y_TLM35, cacheTLM35, sizeof(cacheTLM35), s);
      lastLM = ui_tlm35;
```

```
    }
    if (isnan(lastRTC) || fabsf(tempRTC_in - lastRTC) > H_TEMP) {
      char s2[16];
      snprintf(s2, sizeof(s2), "%5.1f C    ", tempRTC_in);
      printIfChanged(X_VAL, Y_TRTC, cacheTRTC, sizeof(cacheTRTC), s2);
      lastRTC = tempRTC_in;
    }
    tTemp = now + 200;
  }

  // ---------- Inclinacao (10 Hz + histerese 0.2 °) ----------
  if ((int32_t)(now - tIncl) >= 0) {
    if (isnan(lastInc) || fabsf(ui_incl - lastInc) > H_INCL) {
      char s[16];
      snprintf(s, sizeof(s), "%5.1f deg  ", ui_incl);
      printIfChanged(X_VAL, Y_INCL, cacheIncl, sizeof(cacheIncl), s);
      lastInc = ui_incl;
    }
    tIncl = now + 100;
  }

  // ---------- Velocidade (10 Hz + histerese 0.1 km/h) ----------
  if ((int32_t)(now - tVel) >= 0) {
    if (isnan(lastVel) || fabsf(ui_vel - lastVel) > H_VEL) {
      char s[16];
      snprintf(s, sizeof(s), "%5.1f km/h ", ui_vel);
      printIfChanged(X_VAL, Y_VEL, cacheVel, sizeof(cacheVel), s);
      lastVel = ui_vel;
    }
    tVel = now + 100;
  }
}

// ========== FUNÇÃO: Registrar dados em CSV ============
// Salva dados no cartão SD em formato CSV para análise posterior.
void registrarNoSD() {
  if (millis() - tempoUltimoLog < intervaloLog) return; // controla taxa de log
  tempoUltimoLog = millis();

  // ---- Data e hora ----
  int dia = rtc.getDate();
  int mes = rtc.getMonth(century);
  int ano = 2000 + rtc.getYear();
  int hora = rtc.getHour(h12Flag, pmFlag);
  int minuto = rtc.getMinute();

  char horaStr[6];
  sprintf(horaStr, "%02d:%02d", hora, minuto);

  // ---- Nome do arquivo (um por dia) ----
```

```
  char caminho[32];
  sprintf(caminho, "/log_%04d%02d%02d.csv", ano, mes, dia);
  nomeArquivoLog = String(caminho);

  // ---- Cria cabeçalho se ainda não existir ----
  if (!SD.exists(caminho)) {
    File novo = SD.open(caminho, FILE_WRITE);
    if (novo) {

novo.println("Data;Hora;Operador;TempLM35;TempRTC;Bateria;Inclinacao;Velocida
de;Presenca;Checklist");
      novo.close();
    } else {
      // Não conseguiu abrir: aborta esta rodada de log
      return;
    }
  }

  // ---- Abre para append ----
  File file = SD.open(caminho, FILE_APPEND);
  if (!file) return;

  // Checklist: mantém a lógica de imprimir uma única vez (se quiser sempre, remova
essa parte)
  String checklistStr = "";
  if (!checklistImpresso && checklistCompleto) {
    for (int i = 0; i < 20; i++) {
      checklistStr += respostasChecklist[i] ? "1" : "0";
      if (i < 19) checklistStr += ",";
    }
    checklistImpresso = true;
  }

  // Monta linha usando **variáveis locais** (ui_*) para não depender da Cloud
  String linha;
  linha.reserve(160);
  linha  = String(dia) + "/" + String(mes) + "/" + String(ano) + ";";
  linha += String(horaStr) + ";";
  linha += String(operadorID) + ";";          // grava operador em TODAS as linhas
  linha += String(ui_tlm35, 1) + ";";         // TempLM35 (local)
  linha += String(tempRTC, 1) + ";";          // TempRTC (local)
  linha += String(ui_vbat, 2) + ";";          // Bateria (local)
  linha += String(ui_incl, 1) + ";";          // Inclinacao (local)
  linha += String(ui_vel, 1) + ";";           // Velocidade (local)
  linha += String(contadorPresencas) + ";";  // Presenca (contador)
  linha += checklistStr;                      // Checklist (uma vez) ou vazio

  file.println(linha);
  file.close();
}
```

```
// ========== INTERRUPÇÃO: Hall Sensor ============
// Função chamada automaticamente a cada pulso do sensor Hall
void IRAM_ATTR detectarPulsoHall() {
  uint32_t t = micros();
  static uint32_t last = 0;
  if (t - last < HALL_GUARD_US) return; // guarda de tempo para evitar glitches
  hallPulseCount++;
  lastPulseMicros = t;
  last = t;
}

// ========== INICIALIZAÇÃO: Hall Sensor ============
void inicializarSensorHall() {
  pinMode(pinoHall, INPUT);  // Configura pino como entrada
  attachInterrupt(digitalPinToInterrupt(pinoHall), detectarPulsoHall, FALLING);
                              // Ativa interrupção na borda de descida
}

// ========== FUNÇÃO: Obter velocidade ============
float obterVelocidade() {
  static uint32_t t0 = 0;
  static uint32_t count0 = 0;
  static float vFilt = NAN;

  uint32_t now = micros();
  if (now - t0 < VEL_WINDOW_US) {
    // Dentro da janela, retorna último valor filtrado
    return velocidadeAtual;
  }

  // Captura atômica dos contadores
  noInterrupts();
  uint32_t count = hallPulseCount;
  uint32_t lastPulse = lastPulseMicros;
  interrupts();

  uint32_t dp = count - count0;   // pulsos na janela
  float dt = (now - t0) / 1e6f;    // segundos na janela
  count0 = count;
  t0 = now;

  // Converte contagem -> km/h
  float revs = (float)dp / (float)PULSOS_POR_VOLTA;     // voltas na janela
  float mps  = (revs * (2.0f * PI * RAIO_RODA)) / dt;   // m/s
  float kmh  = mps * 3.6f;

  // Se "parado" por muito tempo, zera
  if (now - lastPulse > STOP_TIMEOUT_US) kmh = 0.0f;
```

```
  // Filtro de suavização (EWMA)
  if (isnan(vFilt)) vFilt = kmh;
  else vFilt = (1.0f - ALPHA_VEL) * vFilt + ALPHA_VEL * kmh;

  velocidadeAtual = vFilt;
  return velocidadeAtual;
}

// ========== FUNÇÃO: Ler tensão da bateria ============
float lerTensaoBateria() {
  // Leitura com média aparada (32 amostras, descarta 4 de cada extremo)
  float leitura = adcReadTrimmedMean(pinoBateria, 32, 4);

  // Conversão para tensão no pino (0-3.3V)
  float tensaoPino = (leitura * 3.3f) / 4095.0f;

  // Conversão para tensão real da bateria
  float tensaoBateria = tensaoPino * VDIV_BATT;

  // Filtro EWMA (Média Móvel Ponderada Exponencialmente)
  static float tensaoFiltrada = NAN;
  if (isnan(tensaoFiltrada)) {
    tensaoFiltrada = tensaoBateria;
  } else {
    tensaoFiltrada = (1.0f - ALPHA_VBAT) * tensaoFiltrada + ALPHA_VBAT *
tensaoBateria;
  }

  return tensaoFiltrada;
}

// ========== FUNÇÃO: Ler temperatura LM35 ============
float lerTemperatura() {
  // Leitura com média aparada (32 amostras, descarta 4 de cada extremo)
  float leitura = adcReadTrimmedMean(pinoLM35, 32, 4);

  // Conversão para milivolts (0-1100mV) - ESCALA CORRIGIDA para 0dB
  float milivolts = (leitura * 1100.0f) / 4095.0f;

  // Conversão para temperatura (10mV/°C)
  float temperatura = milivolts * 0.1f;

  // Ajuste de calibração (ganho e offset)
  temperatura = (temperatura * T_GAIN) + T_OFFS;

  // Filtro EWMA (Média Móvel Ponderada Exponencialmente)
  static float temperaturaFiltrada = NAN;
  if (isnan(temperaturaFiltrada)) {
    temperaturaFiltrada = temperatura;
  } else {
```

```
    temperaturaFiltrada = (1.0f - ALPHA_TEMP) * temperaturaFiltrada +
ALPHA_TEMP * temperatura;
  }

  return temperaturaFiltrada;
}

// ========== FUNÇÃO: Ler inclinação com MPU6050 ============
float lerInclinacao() {
  if (!mpuOK) return 0.0f; // se falhou, retorna 0 (ou último valor, se preferir)
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  float incX = atan2(a.acceleration.x,
                     sqrt(a.acceleration.y*a.acceleration.y +
a.acceleration.z*a.acceleration.z)) * 180.0/PI;
  float incY = atan2(a.acceleration.y,
                     sqrt(a.acceleration.x*a.acceleration.x +
a.acceleration.z*a.acceleration.z)) * 180.0/PI;
  return (fabs(incX) > fabs(incY)) ? incX : incY;
}

// ========== FUNÇÃO: Mostrar alerta no display ============
void mostrarAlerta(String tipoAlerta, float valor) {
display.fillScreen(ST77XX_BLACK);  // Limpa tela
display.setTextSize(1);            // Define tamanho pequeno
display.setTextColor(ST77XX_RED);  // Texto vermelho

  display.setCursor(0, 0);           // Cursor inicial
  display.println("");               // Linha em branco
  display.println("       ALERTA CRITICO!\n\n\n\n"); // Mensagem de alerta

  display.setTextColor(ST77XX_WHITE); // Texto branco
  display.print(tipoAlerta);          // Exibe tipo do alerta
  display.println("");
  display.print("  Valor: ");         // Texto fixo
  display.print(valor);               // Mostra valor numérico

  // Adiciona unidade de acordo com o alerta
  String tipo = tipoAlerta;
  tipo.trim();
  if (tipo == "Temperatura") {
    display.println(" C");
  } else if (tipo == "Bateria") {
    display.println(" V");
  } else if (tipo == "Inclinacao") {
    display.println(" Grau");
  } else if (tipo == "Velocidade") {
    display.println(" km/h");
  }
```

```
  display.setTextColor(ST77XX_YELLOW); // Texto amarelo
  display.println("");
  display.println("");
  display.println(" Verifique imediatamente!"); // Instrução

  delay(5000);                    // Mantém alerta visível por 5s

   // Reset do layout para forçar redesenho ao voltar ao status normal
  statusLayoutPronto = false;
}

// ========== FUNÇÃO: Verificar combinação de botões ============
void verificarCombinacaoBotoes() {
  static unsigned long tempoPressionado = 0;
  static bool botonesPressionados = false;

  if (digitalRead(botaoSim) == LOW && digitalRead(botaoNao) == LOW) {
    if (!botonesPressionados) {
      tempoPressionado = millis();
      botonesPressionados = true;
    }

    // Se pressionados por mais de 2 segundos - Debug
    if (millis() - tempoPressionado > 2000 && millis() - tempoPressionado < 3000) {
      mostrarDebugSensores();
    }

    // Se pressionados por mais de 3 segundos - Ajuste de data/hora
    if (millis() - tempoPressionado > 3000) {
      beep(200);
      ajustarDataHora();
      botonesPressionados = false;
    }
  } else {
    botonesPressionados = false;
  }
}

// ========== FUNÇÃO: Ajustar data e hora no RTC ============
void ajustarDataHora() {
  modoAjusteHorario = true;      // Ativa modo ajuste
  etapaAjuste = 0;               // Inicia na primeira etapa

  // Lê valores atuais
  int ano = 2000 + rtc.getYear();
  int mes = rtc.getMonth(century);
  int dia = rtc.getDate();
  int hora = rtc.getHour(h12Flag, pmFlag);
  int minuto = rtc.getMinute();
```

```
int etapaAnterior = -1;        // Para forçar atualização
int maxDias = 31;              // Declara maxDias fora do switch

while (modoAjusteHorario) {
  if (etapaAjuste != etapaAnterior) { // Atualiza tela apenas quando mudar etapa
    display.fillScreen(ST77XX_BLACK);
    display.setCursor(2, 3);
    display.println("AJUSTE DATA/HORA");
    display.setTextColor(ST77XX_CYAN);
    display.println("----------------------");
    display.println("");

    display.setTextColor(ST77XX_YELLOW);
    switch (etapaAjuste) { // Mostra valor a ser ajustado
      case 0: display.print("Ano: "); display.println(ano); break;
      case 1: display.print("Mes: "); display.println(mes); break;
      case 2: display.print("Dia: "); display.println(dia); break;
      case 3: display.print("Hora: "); display.println(hora); break;
      case 4: display.print("Minuto: "); display.println(minuto); break;
    }

    etapaAnterior = etapaAjuste; // Atualiza etapa anterior
  }

  // ---- Botão SIM: incrementa valor ----
  if (digitalRead(botaoSim) == LOW) {
  switch (etapaAjuste) {
      case 0: ano++; break;
      case 1: mes = (mes % 12) + 1; break;
      case 2:
        // Limite de dias conforme o mês
        maxDias = 31;
        if (mes == 2) maxDias = (ano % 4 == 0) ? 29 : 28;
        else if (mes == 4 || mes == 6 || mes == 9 || mes == 11) maxDias = 30;
        dia = (dia % maxDias) + 1;
        break;
      case 3: hora = (hora + 1) % 24; break;
      case 4: minuto = (minuto + 1) % 60; break;
    }
    etapaAnterior = -1;
    beep(50); delay(200);
  }
  // ---- Botão NÃO: decrementa valor ----
  else if (digitalRead(botaoNao) == LOW) {
  switch (etapaAjuste) {
      case 0: ano--; break;
      case 1: mes = (mes + 10) % 12 + 1; break;
      case 2:
        // Limite de dias conforme o mês
        maxDias = 31;
```

```
          if (mes == 2) maxDias = (ano % 4 == 0) ? 29 : 28;
          else if (mes == 4 || mes == 6 || mes == 9 || mes == 11) maxDias = 30;
          dia = (dia + maxDias - 1) % maxDias + 1;
          break;
        case 3: hora = (hora + 23) % 24; break;
        case 4: minuto = (minuto + 59) % 60; break;
      }
      etapaAnterior = -1;
      beep(50); delay(200);
    }
    // ---- Botão ENTER: confirma etapa ----
    else if (digitalRead(botaoEnter) == LOW) {
    etapaAjuste++;
      beep(100); delay(300);
      if (etapaAjuste > 4) {   // Se concluiu ajuste
        rtc.setYear(ano - 2000);
        rtc.setMonth(mes);
        rtc.setDate(dia);
        rtc.setHour(hora);
        rtc.setMinute(minuto);
        modoAjusteHorario = false; // Sai do loop
      }
      etapaAnterior = -1;
    }
  }
}

// ========== FUNÇÃO: Configurar ADC do ESP32 ============
void setupADC() {
  // Configura a atenuação para os pinos do ADC
analogSetPinAttenuation(pinoLM35, ADC_0db);     // Até ~1.1V: ideal para LM35
analogSetPinAttenuation(pinoBateria, ADC_11db); // Até ~3.3V: mantém para
bateria

  // Configura a resolução para 12 bits (0-4095)
  analogReadResolution(12);
}

// ========== FUNÇÃO: Leitura com média aparada ============
// Lê N amostras do ADC, descarta as K menores e K maiores e faz média
float adcReadTrimmedMean(int pin, uint8_t N, uint8_t K) {
  if (K * 2 >= N) K = N / 4; // Previne configuração inválida

  // 1) Leitura dummy para estabilizar o mux/capacitor de amostragem
  (void)analogRead(pin);
  delayMicroseconds(100);

  // 2) Coleta das amostras
  uint16_t amostras[64]; // Suporta até 64 amostras
  for (uint8_t i = 0; i < N; i++) {
```

```
    amostras[i] = analogRead(pin);
    delayMicroseconds(200); // Pequeno intervalo entre amostras
  }

  // 3) Ordenação por inserção (eficiente para poucos elementos)
  for (uint8_t i = 1; i < N; i++) {
    uint16_t chave = amostras[i];
    int8_t j = i - 1;
    while (j >= 0 && amostras[j] > chave) {
      amostras[j + 1] = amostras[j];
      j--;
    }
    amostras[j + 1] = chave;
  }

  // 4) Cálculo da média aparada (descartando K extremos de cada lado)
  uint32_t soma = 0;
  for (uint8_t i = K; i < N - K; i++) {
    soma += amostras[i];
  }

  return (float)soma / (N - 2 * K); // Retorna a média
}

// ========== FUNÇÃO: Desenhar layout fixo (apenas uma vez) ============
static void desenharLayoutStatus() {
  display.fillScreen(ST77XX_BLACK);
  display.setTextWrap(false);
  display.setTextSize(1);

  ajustarColunas();  // <<< calcula X_VAL conforme a largura real
  resetStatusCaches();// <<< força primeira pintura de todos os campos

  // Título
  display.setTextColor(ST77XX_CYAN, ST77XX_BLACK);
  display.setCursor(X_LABEL, Y_TIT);
  display.print("STATUS NORMAL");
  display.drawFastHLine(0, Y_TIT + 10, display.width(), ST77XX_CYAN);

  // Rótulos
  display.setTextColor(ST77XX_YELLOW, ST77XX_BLACK);
  display.setCursor(X_LABEL, Y_HORA);  display.print("Data/Hora:");
  display.setCursor(X_LABEL, Y_OPID);  display.print("Operador ID:");
  display.setCursor(X_LABEL, Y_BATT);  display.print("Bateria:");
  display.setCursor(X_LABEL, Y_TLM35); display.print("Temp Motor:");
  display.setCursor(X_LABEL, Y_TRTC);  display.print("Temp Case:");
  display.setCursor(X_LABEL, Y_INCL);  display.print("Inclinacao:");
  display.setCursor(X_LABEL, Y_VEL);   display.print("Velocidade:");

  // Linhas separadoras
```

```
  int W = display.width();
  display.drawFastHLine(0, Y_OPID - 3, W, ST77XX_CYAN);
  display.drawFastHLine(0, Y_BATT - 3, W, ST77XX_CYAN);
  display.drawFastHLine(0, Y_TLM35 - 3, W, ST77XX_CYAN);
  display.drawFastHLine(0, Y_TRTC - 3, W, ST77XX_CYAN);
  display.drawFastHLine(0, Y_INCL - 3, W, ST77XX_CYAN);
  display.drawFastHLine(0, Y_VEL - 3,  W, ST77XX_CYAN);

  statusLayoutPronto = true;

  // Inicia janelas de atualização
  uint32_t now = millis();
  tHora = tBatt = tTemp = tIncl = tVel = now;
}

// ========== FUNÇÃO: Debug de sensores ============
void mostrarDebugSensores() {
display.fillScreen(ST77XX_BLACK);
display.setTextColor(ST77XX_YELLOW, ST77XX_BLACK);
display.setCursor(0,0);
  display.println("DEBUG SENSORES");
  uint16_t rawBat = (uint16_t)adcReadTrimmedMean(pinoBateria, 16, 2);
  uint16_t rawLM  = (uint16_t)adcReadTrimmedMean(pinoLM35,  16, 2);
  display.setTextColor(ST77XX_WHITE, ST77XX_BLACK);
  display.printf("ADC Bat raw: %u\n", rawBat);
  display.printf("ADC LM35 raw:%u\n", rawLM);
  display.printf("VBat UI: %.2f V\n", ui_vbat);
  display.printf("LM35 UI: %.1f C\n", ui_tlm35);
  display.printf("Incl UI: %.1f deg\n", ui_incl);
  display.printf("MPU OK: %s\n", mpuOK ? "sim" : "nao");
  delay(2000);
  statusLayoutPronto = false;
}

//====================== SETUP ==============================
// A função setup() roda apenas uma vez quando o ESP32 é ligado ou resetado.
void setup() {
  Serial.begin(115200);
  Wire.begin(PIN_SDA, PIN_SCL);
  tempRTC = rtc.getTemperature();
  EEPROM.begin(512);
  delay(1500);

  // ---- Configuração dos pinos ----
  pinMode(botaoEnter, INPUT_PULLUP);
  pinMode(pinoBuzzer, OUTPUT);
  pinMode(pinoRele, OUTPUT);
  pinMode(botaoSim, INPUT_PULLUP);
  pinMode(botaoNao, INPUT_PULLUP);
  digitalWrite(pinoRele, LOW);
```

```
pinMode(pinoPresenca, INPUT);
digitalWrite(pinoPresenca, LOW);

// Configuração do ADC
setupADC();

// ---- Inicializações de módulos ----
inicializarDisplay();
inicializarSenha();
inicializarSensorHall();

// ---- Inicialização do cartão SD ----
if (!SD.begin(SD_CS)) {
display.fillScreen(ST77XX_RED);
display.setCursor(5, 20);
  display.setTextColor(ST77XX_WHITE);
  display.println("ERRO SD CARD");
  // Não travar completamente - apenas continuar sem SD
  delay(2000);
}

 // ---- Inicialização do sensor MPU6050 ----
mpuOK = false;
if (!mpu.begin(0x68)) {
  if (!mpu.begin(0x69)) {
    display.fillScreen(ST77XX_RED);
    display.setCursor(5, 20);
    display.setTextColor(ST77XX_WHITE);
    display.println("ERRO MPU6050");
    delay(2000);      // Continua sem inclinacao
  } else {
    mpuOK = true;
  }
} else {
  mpuOK = true;
}

if (mpuOK) {
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
}

// ---- Autenticação do operador ----
mostrarTelaSenha();
while(!senhaValidada) {
processarSenha();
  delay(100);
}
```

```
  // ---- Exibição inicial ----
  mostrarMensagemInicial();
  identificarOperador();

  // Configuração Wi-Fi não-bloqueante
  WiFi.setAutoReconnect(true);
  WiFi.persistent(true);

  // ---- Inicialização Arduino Cloud NÃO-BLOQUEANTE ----
  inicializarCloudNaoBloqueante();

  // ---- Início do checklist ----
  sistemaAtivo = true;
  perguntaAtual = 0;
  mostrarPerguntaChecklist();
}

// ============= MODIFICAÇÃO NO LOOP PRINCIPAL ============
// A função loop() roda continuamente enquanto o ESP32 está ligado.
void loop() {
  if (sistemaAtivo) {
    verificarCombinacaoBotoes();

    if (perguntaAtual < 20) {
      processarChecklist();
    } else {
      checklistCompleto = true;

      if (digitalRead(pinoRele) == HIGH) {
        // Verificações de conexão (não-bloqueantes)
        verificarConexaoWiFi();
        verificarConexaoCloud();

        // Atualização rápida da Cloud
        ArduinoCloud.update();

        // Monitoramento principal (funciona offline)
        monitorarSensores();
        registrarNoSD();

        // Detecção de presença
        if (digitalRead(pinoPresenca) == HIGH) {
          contadorPresencas++;
          mostrarAlertaPresenca();
        }

        // Reset do contador de presença
        if (digitalRead(botaoEnter) == LOW) {
          delay(200);
          while (digitalRead(botaoEnter) == LOW);
```

```
            contadorPresencas = 0;
          }
        }
      }
    }
  }
  // Pequeno delay para não sobrecarregar o processador
  delay(10);
}
```