INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CAMPUS PIRIPIRI
TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CAIRON FERREIRA PRADO

AMPLIAÇÃO DO FEEDBACK SENSORIAL EM SIMULADORES DE CORRIDA:
desenvolvimento de um painel físico de telemetria com ESP32

PIRIPIRI
2026

CAIRON FERREIRA PRADO

AMPLIAÇÃO DO FEEDBACK SENSORIAL EM SIMULADORES DE CORRIDA:
desenvolvimento de um painel físico de telemetria com ESP32

Trabalho de Conclusão de Curso (relatório técnico) apresentado como exigência parcial para obtenção do diploma do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Piripiri.

Orientador: Prof. Me. Jonathas Jivago de Almeida Cruz.
Coorientador: Prof. Dr. Iallen Gabio de Sousa Santos.

PIRIPIRI
2026

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD

Prado, Cairon Ferreira

P896a Aplicação do feedback sensorial em simuladores de corrida : desenvolvimento de um painel físico de telemetria com ESP32 / Cairon Ferreira Prado. - 2026.

45 f.

Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Piripiri, 2026.

Orientador : Prof Me. Jonathas Jivago de Almeida Cruz.

Coorientador : Prof Dr. Iallen Gabio de Sousa Santos.

1. simulador de corrida. 2. sistemas embarcados. 3. telemetria. 4. firmware. I. Título.

CDD - 004.21

Elaborado por Júlio César Oliveira Rodrigues CRB CRB 3/1125

CAIRON FERREIRA PRADO

AMPLIAÇÃO DO FEEDBACK SENSORIAL EM SIMULADORES DE CORRIDA:
desenvolvimento de um painel físico de telemetria com ESP32

Trabalho de Conclusão de Curso (relatório técnico) apresentado como exigência parcial para obtenção do diploma do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Piripiri.

Aprovado em: 15 de julho de 2026.

BANCA EXAMINADORA

Prof. Me. Jonathas Jivago de Almeida Cruz
Orientador – IFPI

Prof. Dr. Iallen Gabio de Sousa Santos
Coorientador – IFPI

Prof. Me. Mike Christian Sousa Araújo
Convidado – IFPI

RESUMO

Este relatório técnico apresenta o desenvolvimento e validação de um painel físico de telemetria baseado em sistema embarcado com ESP32, para converter dados de simuladores de corrida em respostas físicas e visuais. O sistema recebe telemetria via protocolo UDP (formato *Data Out* do *Forza Motorsport 7*) e exibe as informações processadas em instrumentos analógicos e num *display* integrado. O protótipo indica velocidade, rotação do motor, combustível e temperatura dos pneus fisicamente, além de exibir marcha, posição, volta e *status* operacionais na tela. Na construção, o projeto adaptou um painel automotivo real, integrando motores de acionamento para os ponteiros, interface visual e a eletrônica interna. O *firmware* foi estruturado em camadas, influenciado pela arquitetura *Ports and Adapters*, o que reduz o acoplamento e facilita a evolução da solução. Destacam-se o portal assíncrono de configuração Wi-Fi, o serviço de telemetria (com gestão de *timeout*) e a organização por catálogos. A validação incremental englobou testes unitários, integração *host-side*, teste de componente, *smoke test* e integração UDP real. Esse método atestou o núcleo do *firmware* antes da execução embarcada, obtendo aprovação integral nos 68 casos da suíte `test_native`, de um total de 73 casos documentados. Os resultados comprovam que o sistema desenvolvido recebe, processa e representa os dados com sucesso, combinando exibição visual e resposta mecânica.

Palavras-chave: simulador de corrida; sistemas embarcados; ESP32; telemetria; *firmware*.

ABSTRACT

This technical report presents the development and validation of a physical telemetry dashboard based on an ESP32 embedded system, designed to convert racing simulator data into physical and visual responses. The system receives telemetry via the UDP protocol (*Data Out* format of *Forza Motorsport 7*) and displays the processed information on analog instruments and an integrated display. The prototype physically indicates speed, engine rotation, fuel level, and tire temperature, in addition to displaying gear, position, lap, and operational status on the screen. In its construction, the project adapted a real automotive dashboard, integrating actuation motors for the pointers, a visual interface, and internal electronics. The firmware was structured in layers, influenced by the *Ports and Adapters* architecture, which reduces coupling and facilitates the solution's evolution. Highlights include the asynchronous Wi-Fi configuration portal, the telemetry service (with timeout management), and the organization by catalogs. The incremental validation encompassed unit tests, *host-side* integration, component testing, *smoke testing*, and real UDP integration. This method verified the firmware core prior to embedded execution, achieving full approval in all 68 cases of the `test_native` suite, out of a total of 73 documented cases. The results prove that the developed system successfully receives, processes, and represents data, combining visual display and mechanical response.

Keywords: racing simulator; embedded systems; ESP32; telemetry; firmware.
Data de aprovação: 15/07/2026

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ESP32	Microcontrolador utilizado como plataforma de execução do sistema embarcado.
GPIO	*General Purpose Input/Output*; pinos digitais de entrada e saída de propósito geral.
MVP	*Minimum Viable Product*; Produto Mínimo Viável.
RF	Requisito Funcional.
RNF	Requisito Não Funcional.
RPM	Rotações por minuto.
TFT	*Thin Film Transistor*; tecnologia do visor utilizado no protótipo.
UDP	*User Datagram Protocol*; protocolo de transporte utilizado para recepção da telemetria.
USB	*Universal Serial Bus*; barramento usado para alimentação e comunicação durante parte dos testes.

SUMÁRIO

1 INTRODUÇÃO

Os jogos digitais evoluíram significativamente em termos de realismo gráfico, sonoro e interativo, especialmente nos gêneros de simulação. Nesse contexto, os simuladores de corrida, também conhecidos como *sim racing*, buscam reproduzir aspectos da condução automobilística por meio de ambientes virtuais de alta fidelidade, nos quais a percepção do usuário depende não apenas da imagem exibida na tela, mas também da interpretação de sinais físicos e operacionais do veículo. A telemetria, nesse cenário, exerce papel relevante ao permitir o registro, a transmissão e a análise de estados do automóvel em tempo real, contribuindo para treinamento, desempenho e aprimoramento da experiência de uso (Kohwalter; Murta; Clua, 2017; Bugeja; Spina; Buhagiar, 2017).

Apesar dos avanços gráficos e da popularização dos periféricos voltados à simulação, parte expressiva das informações críticas de pilotagem ainda permanece concentrada em interfaces digitais. Dados como velocidade, rotação do motor, marcha, nível de combustível e temperatura dos pneus são geralmente apresentados na tela principal do jogo ou em dispositivos auxiliares específicos. Essa condição pode limitar a experiência do usuário, pois mantém informações relevantes dependentes da leitura visual direta e reduz a correspondência entre a simulação e a lógica perceptiva de um painel automotivo físico.

Além da limitação perceptiva, observa-se uma lacuna tecnológica entre soluções comerciais de alto custo e alternativas artesanais de maior complexidade técnica. O uso de periféricos capazes de ampliar a imersão por meio de estímulos multissensoriais tem sido associado ao crescimento do ecossistema de simuladores e esportes eletrônicos (Calleo; Dall'Osso; Zannoni, 2023). Entretanto, plataformas com retorno háptico e instrumentação avançada tendem a apresentar custo elevado, enquanto soluções baseadas em computação física exigem conhecimentos de eletrônica, programação e integração de sistemas (de Frutos; Castro, 2021; Love; Asempapa, 2022; Vergara; Herskovic, 2025).

Diante desse cenário, o problema abordado neste Trabalho de Conclusão de Curso consiste em converter dados de telemetria de simuladores de corrida em feedback físico e visual por meio de um painel dedicado, reduzindo a dependência exclusiva da tela e aproximando a experiência de uso da dinâmica de um cockpit real. A proposta busca transformar o fluxo de dados do simulador em comportamento observável, permitindo que informações operacionais do veículo sejam representadas por instrumentos analógicos e por uma interface visual embarcada.

Para responder a esse problema, foi desenvolvido um Produto Mínimo Viável (MVP) de um painel físico de telemetria baseado em sistema embarcado. O protótipo utiliza um microcontrolador ESP32 para receber pacotes de telemetria via UDP, decodificar o formato *Data Out* do simulador Forza Motorsport 7 e acionar instrumentos físicos de velocidade, rotação do motor, combustível e temperatura média dos pneus, além de exibir informações complementares em um display TFT. Essa caracterização está alinhada ao escopo funcional descrito no relatório,

que define a recepção, interpretação e representação dos dados de telemetria como núcleo do sistema.

Além da funcionalidade principal, o projeto considera características de uso e evolução do produto, como configuração simplificada, modularidade e possibilidade de expansão. A abordagem *plug-and-play* adotada busca reduzir configurações manuais, permitindo que o usuário realize a conexão inicial do dispositivo e utilize o painel sem alterações diretas no código-fonte. Em sistemas conectados, mecanismos de integração e abstração da tecnologia subjacente contribuem para simplificar a comunicação entre componentes e reduzir barreiras de uso (Al-Fuqaha *et al.*, 2015; Roy; Misra; Raghuwanshi, 2019).

Do ponto de vista de engenharia de software, o firmware foi organizado de forma modular, com inspiração em camadas e em princípios de *Ports and Adapters*. Essa decisão visa reduzir o acoplamento com o hardware, favorecer a manutenção, permitir expansão futura para novos instrumentos ou decodificadores e viabilizar testes em ambiente *host-side*. Assim, a contribuição deste trabalho não se limita à construção do protótipo físico, mas envolve também a organização arquitetural, a validação automatizada e a documentação técnica necessárias para evolução controlada do sistema.

Como resultado das implementações propostas, obteve-se um protótipo funcional e robusto que preserva a estética e a ergonomia de um painel automotivo real. O dispositivo mostrou-se capaz de processar os dados de telemetria, integrando com sucesso as informações visuais no *display* TFT e o acionamento mecânico dos ponteiros analógicos originais. O acondicionamento dos componentes eletrônicos no interior da carcaça, somado à disposição das conexões de alimentação e do botão de controle, validou a viabilidade técnica da solução, entregando o *feedback* imersivo pretendido para o simulador

1.1 JUSTIFICATIVA

A evolução dos simuladores de corrida ampliou a busca por experiências mais próximas da condução automobilística real. Nesse contexto, a imersão do usuário não depende apenas da qualidade gráfica ou sonora, mas também da forma como informações do veículo são percebidas durante a simulação. Dados como velocidade, rotação do motor, marcha, temperatura dos pneus e nível de combustível são fundamentais para a experiência de pilotagem, porém, em muitos casos, permanecem concentrados na tela ou em dispositivos auxiliares específicos.

Essa concentração das informações em interfaces digitais limita parte da experiência do jogador, pois exige leitura visual direta de dados que, em um veículo real, são percebidos por meio de instrumentos físicos, sinais táteis, resposta mecânica e disposição espacial do painel. Com a popularização dos simuladores de corrida e dos esportes eletrônicos (*e-sports*), cresceu também a demanda por periféricos e interfaces capazes de reproduzir estímulos multissensoriais e aprofundar a imersão do usuário (Calleo; Dall'Osso; Zannoni, 2023).

Esse movimento também é observado em indicadores de mercado: segundo a Global Market Insights, o mercado global de simuladores de corrida foi avaliado em US$ 1,8 bilhão em 2023,

com projeção de crescimento a uma taxa composta anual superior a 3% entre 2024 e 2032, conforme apresenta a Tabela 1 (Global Market Insights, 2024).

Tabela 1 – Estimativa de evolução do mercado global de simuladores de corrida

Ano	Valor estimado	Observação
2023	US$ 1,8 bilhão	Valor estimado do mercado global de simuladores de corrida.
2024–2032	CAGR superior a 3%	Projeção de crescimento anual composto para o período.

Fonte: elaborada pelos autores com base em Global Market Insights (2024).

Os dados apresentados na Tabela 1 reforçam que a simulação de corrida constitui um segmento em expansão, impulsionado por fatores como a popularização dos *e-sports*, a aproximação entre fabricantes e o automobilismo virtual, além do avanço de tecnologias voltadas à ampliação do realismo e da experiência imersiva (Global Market Insights, 2024). Dessa forma, o desenvolvimento de interfaces físicas capazes de representar dados de telemetria se mostra alinhado a uma demanda contemporânea por soluções que ampliem a percepção sensorial e operacional do usuário durante a simulação.

Entretanto, o ecossistema de simulação automotiva apresenta uma barreira de acesso relevante. Plataformas comerciais providas de retorno háptico de alta fidelidade e instrumentação avançada possuem custo de aquisição restritivo (de Frutos; Castro, 2021; Murphy *et al.*, 2026). Por outro lado, soluções alternativas de baixo custo baseadas em computação física (*physical computing*) geralmente exigem domínio simultâneo de eletrônica, programação, montagem de circuitos e integração de sistemas, o que impõe elevada carga cognitiva a usuários leigos (Love; Asempapa, 2022; Vergara; Herskovic, 2025). Evidencia-se, portanto, uma lacuna tecnológica entre soluções comerciais financeiramente inacessíveis e montagens manuais de alta complexidade técnica.

Diante dessa lacuna, justifica-se o desenvolvimento de uma solução intermediária, capaz de converter dados de telemetria de simuladores de corrida em feedback físico e visual por meio de um painel dedicado. A proposta busca reduzir a dependência exclusiva da tela e ampliar a percepção do usuário sobre o estado do veículo simulado, aproximando a experiência de uso da lógica operacional de um cockpit real.

Em sistemas embarcados, decisões de modularidade e configurabilidade precisam considerar restrições de custo, tempo real, consumo de energia e recursos disponíveis (Friedrich *et al.*, 2001). Além disso, em sistemas conectados, a heterogeneidade de dispositivos e protocolos reforça a necessidade de mecanismos de integração capazes de simplificar a comunicação entre componentes (Al-Fuqaha *et al.*, 2015). No mesmo sentido, propostas de integração *plug-and-play* indicam que a abstração da tecnologia subjacente contribui para reduzir barreiras de uso e melhorar o custo-benefício de soluções baseadas em dispositivos físicos (Roy; Misra; Raghuwanshi, 2019).

Dessa forma, o projeto justifica-se por propor um painel físico de telemetria modular e ex-

pansível, baseado em sistema embarcado, capaz de receber, decodificar e representar dados do simulador em instrumentos físicos e interface visual. Ao adotar uma abordagem *plug-and-play*, o sistema busca reduzir configurações manuais e facilitar o uso do protótipo. A contribuição esperada, portanto, não se limita à construção do dispositivo, abrangendo também a organização arquitetural, a validação automatizada e a documentação técnica necessárias para que a solução possa evoluir de maneira controlada.

1.2 OBJETIVOS

Nesta seção, são delimitados o objetivo geral e os objetivos específicos do projeto, definindo o propósito central do trabalho e as etapas necessárias para o alcance dos resultados pretendidos.

1.2.1 OBJETIVO GERAL

Desenvolver e validar uma solução de painel físico de telemetria destinada a ampliar a percepção sensorial do usuário em simuladores de corrida, convertendo dados operacionais do veículo simulado em respostas físicas e visuais por meio de instrumentos analógicos e interface embarcada.

1.2.2 OBJETIVOS ESPECÍFICOS

- Adaptar um painel automotivo real para funcionar como painel físico de telemetria em simuladores de corrida.
- Integrar instrumentos analógicos de velocidade, rotação, combustível e temperatura, permitindo que os dados recebidos por telemetria sejam representados por ponteiros físicos.
- Implementar a configuração e a reconexão de rede local por meio de portal Wi-Fi embarcado.
- Receber continuamente pacotes de telemetria via UDP na rede local utilizada pelo simulador.
- Decodificar o formato *Forza Motorsport 7 (Data Out)* e extrair velocidade, rotação, marcha, volta, posição, combustível e temperaturas dos pneus.
- Traduzir os sinais de telemetria para atuação física nos instrumentos do painel, incluindo o controle dos ponteiros por meio de módulos de acionamento.
- Exibir, em interface visual embarcada, estados de conectividade, marcha, status operacional e *layouts* de acompanhamento da telemetria.
- Organizar a solução embarcada de forma modular, favorecendo manutenção, testabilidade e expansão futura para novos instrumentos ou novos *decoders*.
- Validar a solução por meio de testes automatizados *host-side*, execução controlada no ESP32 e verificação do comportamento físico do protótipo.

1.3 MÉTODO DE TRABALHO

A condução deste trabalho pauta-se na metodologia de pesquisa aplicada, que se caracteriza pela busca de soluções para problemas específicos por meio da criação de um artefato tecnológico (Wazlawick, 2017). O desenvolvimento foi estruturado de forma iterativa, combinando o levantamento de requisitos técnicos do protocolo *Data Out* (*Forza Motorsport 7*) com uma implementação incremental e prototipagem embarcada.

Conforme pontuam Huang et al. (2008), sistemas que integram simulação e componentes físicos exigem uma verificação sistemática de comportamento e resposta operacional; por esse motivo, a etapa de validação foi central no processo, buscando assegurar que o *feedback* físico correspondesse fielmente aos eventos gerados pelo simulador.

No que tange à engenharia do *firmware*, adotou-se o padrão arquitetural *Ports and Adapters* para garantir a separação entre a lógica de domínio e os detalhes de *hardware*. Essa decisão metodológica permitiu a realização de testes em ambiente *host-side* (*desktop*), garantindo a consistência do processamento da telemetria antes de sua transposição para o microcontrolador ESP32.

Para lidar com a integração dos periféricos, seguiu-se a estratégia de divulgação progressiva proposta por Vergara e Herskovic (2025), permitindo que *displays* e atuadores fossem validados de forma modular. Ao final, o método adotado buscou preservar a plena rastreabilidade entre os requisitos iniciais, a organização arquitetural, a implementação do código-fonte e as evidências de teste obtidas nas sessões de simulação.

2 TECNOLOGIAS UTILIZADAS

Este capítulo apresenta as principais tecnologias utilizadas no desenvolvimento do painel físico de telemetria. A seleção tecnológica considerou a necessidade de integrar comunicação em rede local, processamento embarcado, acionamento de instrumentos físicos, interface visual e validação automatizada. Como o projeto não envolve *backend* web, banco de dados ou infraestrutura de *deploy* em nuvem, o foco recai sobre os recursos necessários à construção e execução de um sistema embarcado capaz de receber dados de telemetria e convertê-los em respostas físicas e visuais no protótipo, conforme detalhado no Quadro 1.

Quadro 1 – Tecnologias utilizadas no projeto

Tecnologia	**Papel no projeto**
ESP32 Dev Module	Plataforma de execução do *firmware* e integração com Wi-Fi, GPIO e periféricos do protótipo.
C/C++	Linguagem empregada na implementação do *firmware* e das suítes de teste.
Arduino Framework	Base de acesso ao ciclo `setup/loop`, temporização, pinos digitais, Wi-Fi e demais recursos do microcontrolador.
PlatformIO	Orquestração de *build*, transferência do programa para a placa, ambientes de teste e reprodução da validação.
UDP	Transporte de telemetria em rede local entre o simulador e o dispositivo embarcado.
WiFi, WiFiUDP, WebServer e Preferences	Conectividade local, recepção de pacotes, portal de configuração e persistência de credenciais.
TFT_eSPI	Renderização dos *layouts* no *display* ST7789 de 240x320 pixels.
TMC2208	Driver empregado no acionamento dos motores associados aos ponteiros do painel.
Unity	*Framework* de testes automatizados no computador e no ESP32.
PlantUML e Markdown	Documentação arquitetural, modelagem técnica e rastreabilidade textual do TCC.

Fonte: Elaborada pelos autores (2026).

A síntese apresentada no quadro acima deve ser compreendida a pilha tecnológica efetivamente materializada no repositório e no protótipo, e não como um inventário genérico de ferramentas. No nível de execução, o ESP32 Dev Module atua como núcleo do sistema embarcado e concentra, em um mesmo ciclo operacional, a inicialização do dispositivo, a conexão de rede, a recepção da telemetria, a atualização da interface visual e o acionamento dos instrumentos físicos. Essa centralização está alinhada à composição principal do projeto, na qual o microcontrolador coordena portal Wi-Fi, serviço de telemetria, *layouts* de exibição e o conjunto de instrumentos que compõem o painel.

No plano de implementação, a escolha por C/C++ e Arduino Framework decorre da necessidade de acesso direto a temporização, GPIO, conectividade sem fio e ao ciclo `setup /loop` exigido pelo ESP32. Essas tecnologias sustentam uma organização modular do *firmware*, em que classes como `App`, `TelemetryService`, `DisplayService` e

`InstrumentCluster` distribuem responsabilidades sem concentrar a lógica em um bloco monolítico. Com isso, a base do sistema permanece apta a receber novos instrumentos, novos *layouts* ou até novos *decoders*, preservando o núcleo da aplicação e favorecendo a manutenção do código.

As tecnologias de comunicação e persistência listadas na tabela possuem relação direta com o uso pretendido para o protótipo. O protocolo UDP foi adotado porque corresponde ao meio pelo qual o simulador entrega os dados de telemetria em rede local, enquanto as bibliotecas `WiFi`, `WiFiUDP`, `WebServer` e `Preferences` viabilizam, respectivamente, a conectividade sem fio, a leitura contínua dos datagramas, o portal de configuração e o armazenamento de credenciais. No contexto deste trabalho, essa combinação permite que o painel tente reconectar-se automaticamente a uma rede já conhecida e, em caso de falha, exponha o ponto de acesso `SimHub` para configuração sem necessidade de reprogramação do dispositivo.

Na camada de saída, o `TFT_eSPI` e o TMC2208 cumprem papéis complementares na materialização do *feedback*. O primeiro é responsável por renderizar no visor ST7789 as informações de marcha, volta, posição, conectividade e estados operacionais por meio de diferentes *layouts*. O segundo é empregado no acionamento dos motores que movimentam os ponteiros dos indicadores de velocidade, rotação, combustível e temperatura média dos pneus. Dessa forma, a pilha tecnológica apresentada mantém coerência com o objetivo central do trabalho, que é converter eventos digitais provenientes do simulador em respostas visuais e mecânicas percebidas pelo usuário em tempo real.

Por fim, PlatformIO, Unity, PlantUML e Markdown sustentam a engenharia do projeto para além da execução embarcada. O PlatformIO organiza os ambientes de compilação, gravação e teste, permitindo separar a versão destinada ao ESP32 das suítes executadas em ambiente nativo. O Unity dá suporte à validação automatizada de serviços, *decoders*, interfaces e fluxos de integração, reduzindo o risco de regressões ao longo da evolução do *firmware*. Já PlantUML e Markdown foram empregados para registrar arquitetura, diagramas e evidências de teste, assegurando que a documentação permanecesse vinculada à implementação efetivamente entregue neste trabalho.

3 MODELAGEM

Este capítulo apresenta a modelagem do projeto a partir de uma sequência progressiva. Inicialmente, são descritos os requisitos e os casos de uso que delimitam o escopo funcional do MVP. Em seguida, apresenta-se a arquitetura geral do sistema, responsável por situar o *firmware* embarcado em relação ao simulador, à rede local e ao painel físico. Na sequência, são discutidos os modelos comportamentais e estruturais do software, incluindo o diagrama de atividades, o diagrama de classes e os diagramas de sequência. Por fim, são apresentadas as interfaces visuais implementadas no *display* embarcado.

3.1 REQUISITOS DO SISTEMA

O catálogo de requisitos do projeto foi mantido e atualizado ao longo do desenvolvimento, servindo como referência para o recorte do produto mínimo viável (MVP) final. A definição desses requisitos considerou a necessidade central de receber dados de telemetria em tempo real, interpretá-los e convertê-los em respostas físicas e visuais. Tendo em vista que o escopo inicialmente concebido era mais amplo do que a implementação efetivamente entregue, os requisitos apresentados neste relatório devem ser compreendidos sob três estados de maturidade: integralmente implementados, parcialmente contemplados e previstos como evoluções futuras.

Requisitos Funcionais

- RF01 – conexão automática à rede local: implementado por meio do `WiFiConfigPortal`, com reaproveitamento de credenciais salvas e abertura de ponto de acesso quando a conexão falha.
- RF02 – comunicação com o jogo via UDP: implementado pelo adaptador `UdpReceiver` e pelo ambiente de telemetria do *firmware*.
- RF03 – interpretação dos dados recebidos: implementado pelo `Forza7Decoder` e pelo `TelemetryService`.
- RF04 – controle do indicador de velocidade: implementado por `SpeedGauge` e `GaugeMotorTmc2208`.
- RF05 – controle do indicador de rotação do motor: implementado por `RpmGauge` e `GaugeMotorTmc2208`.
- RF06 – controle do indicador de marcha: implementado na interface TFT por meio do `DisplayService` e dos *layouts* embarcados.
- RF07 – detecção automática de módulos conectados: permanace fora do MVP entregue no firmware atual.
- RF08 – inicialização automática do sistema: implementado na classe `App`, que inicializa portal, telemetria, interface e instrumentos ao receber energia.

- RF09 – indicação de status operacional: implementado por meio do estado `UiStatus` e das telas de conexão, portal e telemetria.
- RF10 a RF13 – compatibilidade formal com Xbox Series, PlayStation 4, PlayStation 5 e PC: implementado, todas as plataformas são compatíveis.
- RF14 e RF15 – efeitos adicionais de vibração: permanecem fora do MVP entregue no *firmware* atual.

Requisitos Não Funcionais

- RNF01 – facilidade de uso: parcialmente atendido pelo fluxo de portal Wi-Fi, que reduz configurações manuais e dispensa recompilação para troca de rede.
- RNF02 – baixa latência: orientou a escolha de UDP e o desenho do *firmware*, mas não possui medição formal fim a fim concluída nesta versão do trabalho.
- RNF04 – modularidade e expansibilidade: atendido pela organização em camadas, pelas portas e pelo registro por catálogos de *layouts* e instrumentos.
- RNF05, RNF11, RNF12 e RNF13 – compatibilidade por plataforma: permanecem como objetivo de validação futura.
- RNF06 – robustez: contemplado nos testes por cenários de pacote inválido, descarte, *timeout*, rajadas de pacotes e recuperação.
- RNF08 – escalabilidade funcional: parcialmente contemplado, pois a arquitetura permite novos *decoders*, ainda que apenas um protocolo concreto esteja implementado.
- RNF09 – manutenibilidade: atendido pela organização do código, pela injeção de dependências e pela automação de testes.
- RNF10 – confiabilidade física: parcialmente contemplado. O *firmware* implementa limites lógicos e calibração dos instrumentos, mas a validação eletromecânica completa depende de fechamento experimental da bancada.

3.2 CASOS DE USO

De acordo com Sommerville (2019), a modelagem de casos de uso é uma técnica essencial da engenharia de requisitos, pois permite representar as interações esperadas entre o sistema e o seu ambiente externo por meio de cenários práticos e compreensíveis. No contexto deste trabalho, a definição estruturada desses casos foi fundamental para mapear os requisitos brutos em ações concretas, delimitando exatamente como os estímulos do simulador deveriam ser processados pela rede e convertidos em respostas físicas pelo *hardware*. Dessa forma, a análise dos casos de uso orientou as etapas de desenvolvimento do projeto. Considerando o escopo final consolidado, os casos de uso centrais efetivamente materializados pelo *firmware* podem ser sintetizados conforme o Quadro 2.

Quadro 2 – Casos de uso centrais do MVP final

ID	Caso de uso	Materialização no projeto
UC11	Conectar automaticamente à rede local	O *firmware* tenta reutilizar credenciais salvas, inicia tentativa assíncrona de conexão e abre o portal `SimHub` quando necessário.
UC12	Receber telemetria via UDP	O adaptador `UdpReceiver` abre a porta 5300 e fornece pacotes ao serviço de telemetria.
UC13	Interpretar dados de telemetria	O `Forza7Decoder` converte o pacote bruto em um `TelemetryFrame` consumido pela aplicação.
UC01	Exibir velocidade no painel físico	O `SpeedGauge` converte velocidade em passos e movimenta o ponteiro físico através do `MotorController`.
UC02	Exibir rotação do motor no painel físico	O `RpmGauge` controla o instrumento de RPM com suavização e resposta acelerada em variações bruscas.
UC20	Exibir temperatura dos pneus no painel físico	O adaptador `TireTempGauge` converte temperatura do motor em posição do ponteiro físico.
UC21	Exibir nível de combustível no painel físico	O adaptador `FuelGauge` controla o indicador de combustível baseado nos dados de telemetria.
UC03	Exibir marcha no *display* TFT	O `DisplayService` apresenta a marcha de forma destacada no centro da interface com atualização em tempo real.
UC16	Exibir velocidade no *display* TFT	O `DisplayService` renderiza o indicador de velocidade com escala dinâmica e unidades de velocidade.
UC18	Exibir rotação do motor no *display* TFT	O `DisplayService` apresenta o tacômetro com visualização de RPM em tempo real.
UC17	Exibir temperatura no *display* TFT	O `DisplayService` exibe a temperatura média dos pneus com indicação visual de estados críticos.
UC22	Exibir combustível no *display* TFT	O `DisplayService` renderiza o nível de combustível como percentual restante atualizado continuamente.
UC19	Exibir volta atual no *display* TFT	O `DisplayService` exibe a volta atual da corrida.
UC04	Consultar status operacional do sistema	O estado `UiStatus` concentra conectividade, qualidade da telemetria, volta, posição e contadores de falha.

Fonte: Elaborada pelos autores (2026).

Casos de uso relacionados à compatibilidade multiplataforma e a efeitos adicionais de vibração continuam pertencendo ao escopo analítico do trabalho, mas não foram integralmente convertidos em funcionalidade entregue no *firmware* versionado.

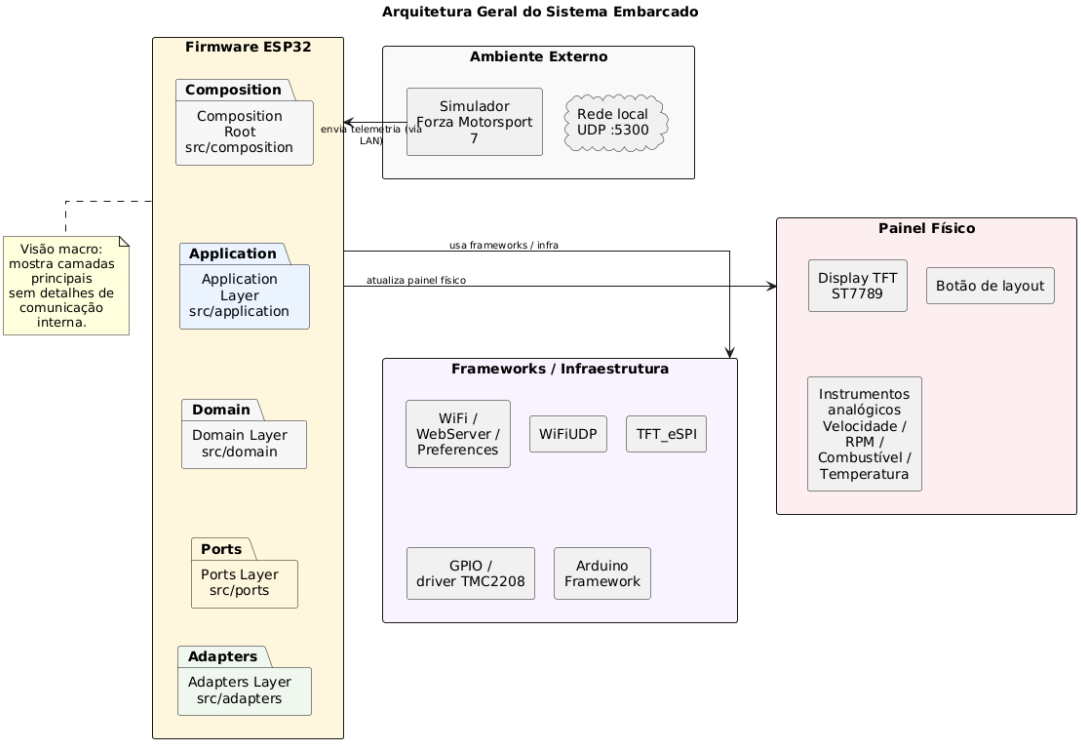
3.3 ARQUITETURA DA SOLUÇÃO

A arquitetura adotada combina organização em camadas com princípios de inversão de dependência e *Ports and Adapters*. Nesse tipo de organização, a lógica central da aplicação depende de contratos e não diretamente de implementações concretas de infraestrutura (Cockburn, 2005; Martin, 2017). No projeto, essa separação permite isolar a lógica de telemetria, atualização da interface e acionamento dos instrumentos em relação a bibliotecas específicas do ESP32, como Wi-Fi, UDP, GPIO e *display*.

A Figura 1 apresenta a visão macro da arquitetura do sistema embarcado. Nessa representação, o objetivo é evidenciar a relação entre o ambiente externo, o *firmware* executado no ESP32,

os recursos de infraestrutura e o painel físico utilizado pelo usuário.

Figura 1 – Arquitetura geral do sistema embarcado.



Fonte: Elaborada pelos autores (2026).

Na arquitetura geral, o simulador Forza Motorsport 7 envia dados de telemetria pela rede local utilizando pacotes UDP na porta 5300. Esses dados são recebidos pelo *firmware* embarcado no ESP32, processados pela aplicação e convertidos em respostas no painel físico, incluindo a movimentação dos instrumentos analógicos e a atualização da interface visual no *display* TFT. A representação arquitetural não detalha classes individuais, pois essa responsabilidade é assumida pelo diagrama de classes apresentado posteriormente. Seu papel é demonstrar como os principais blocos do sistema se relacionam em nível macro: o simulador como fonte dos dados, a rede local como meio de comunicação, o ESP32 como unidade de processamento, os recursos de infraestrutura como suporte operacional e o painel físico como meio de saída perceptível ao usuário.

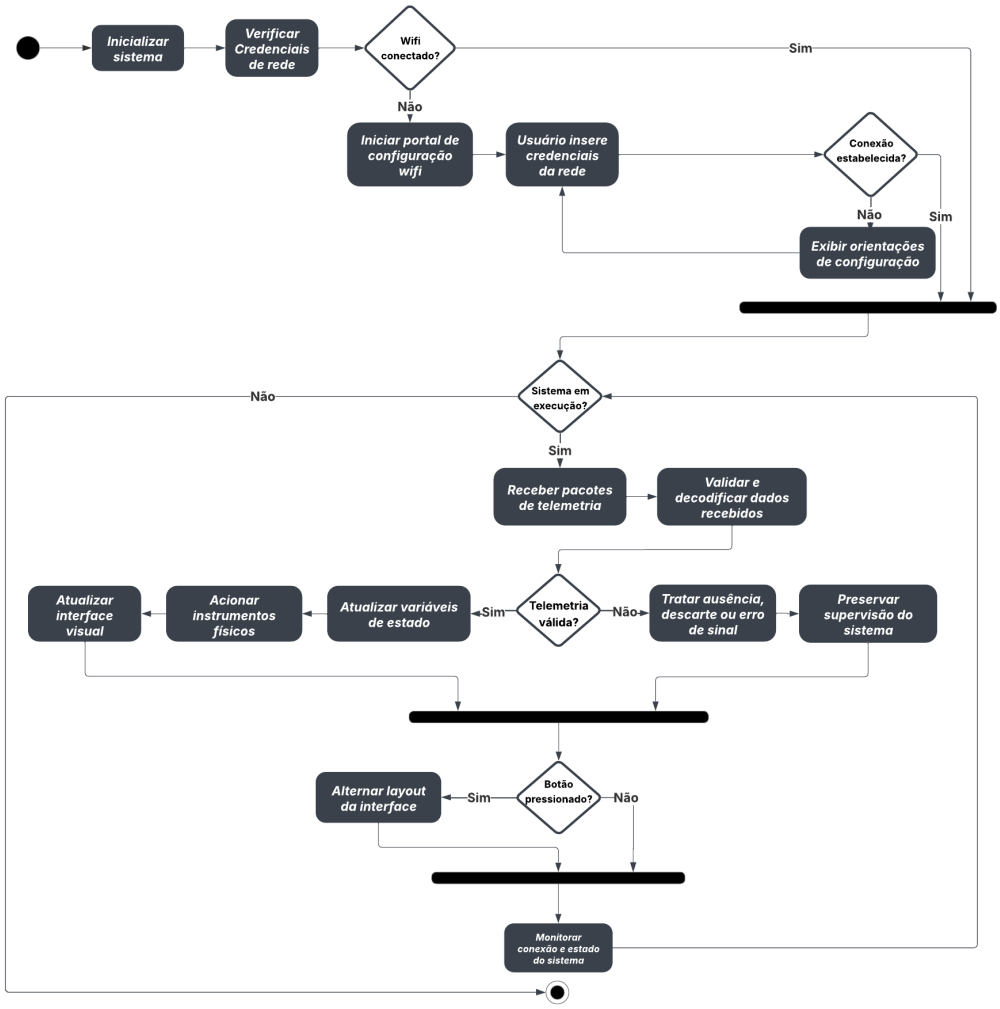
Essa separação evita a repetição entre os modelos. A arquitetura geral descreve a integração do sistema com o ambiente externo, o diagrama de atividades apresenta o fluxo operacional do sistema, o diagrama de classes descreve a estrutura interna do *firmware* e os diagramas de sequência detalham o comportamento temporal das principais operações.

3.4 DIAGRAMA DE ATIVIDADES

Após a apresentação da arquitetura geral, a modelagem comportamental do sistema é detalhada por meio do diagrama de atividades. Segundo Sommerville (2019), este diagrama é útil para ilustrar o fluxo de processamento de dados e a sequência de atividades operacionais necessárias para concluir um fluxo de trabalho, evidenciando pontos de decisão, processamentos paralelos, desvios condicionais e interações sequenciais de controle.

No contexto deste projeto, o diagrama de atividades representa o ciclo operacional do *firmware* desde o momento em que o hardware é energizado até a execução contínua das rotinas de conectividade, recepção de telemetria, atualização dos instrumentos e supervisão do sistema, conforme ilustrado na Figura 2.

Figura 2 – Diagrama de atividades do sistema.



Fonte: Elaborada pelos autores (2026).

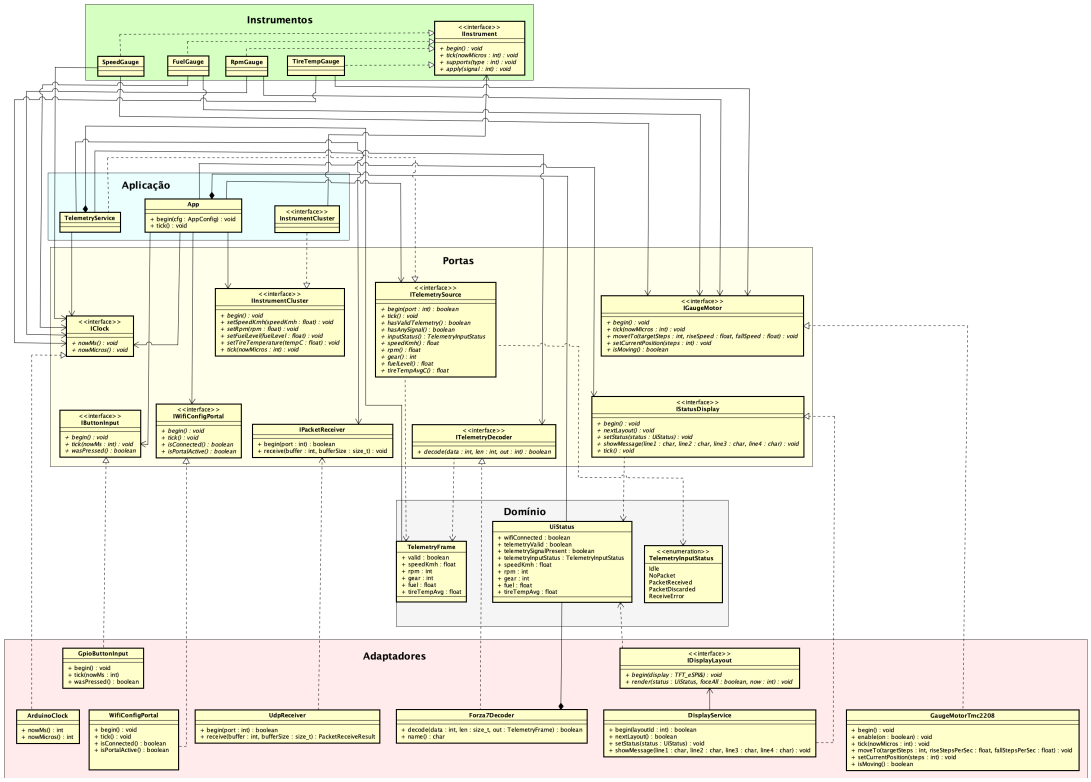
O fluxo representado é estruturado em três macroblocos principais de execução sequencial:

- **Inicialização:** Compreende o ponto de partida do sistema, realizando a inicialização física e lógica do hardware, a configuração dos componentes de software principais e a verificação inicial das credenciais de rede armazenadas no dispositivo.
- **Conexão Wi-Fi:** Gerencia o estado de conectividade de forma condicional. Se o dispositivo já se conecta automaticamente à rede, o fluxo prossegue diretamente. Caso contrário, o *firmware* ativa o portal de configuração Wi-Fi e entra em um ciclo de interação: exibe as orientações de configuração no painel, recebe as credenciais submetidas pelo usuário e tenta efetuar a conexão com a rede local até que o acesso seja estabelecido com sucesso.
- **Modo Operacional:** Consiste no loop de execução principal da aplicação, subdividido em dois módulos que rodam continuamente enquanto o sistema estiver ativo:
  - **Telemetria:** Responsável por receber os pacotes via rede UDP, validar sua integridade e decodificar os dados brutos. Havendo telemetria válida, o sistema atualiza as variáveis internas de estado, aciona os instrumentos físicos (ponteiros analógicos) e atualiza a interface visual do display TFT. Caso os dados recebidos sejam inválidos (ou em caso de ausência de pacotes), o fluxo desvia para o tratamento de descarte ou erro de sinal, preservando as telas de supervisão do sistema.
  - **Interação e Supervisão:** Responsável por escutar interações físicas do usuário. Caso o botão físico seja pressionado, o *firmware* alterna ciclicamente o layout ativo da interface TFT. Em seguida, monitora de forma contínua a integridade da conexão Wi-Fi e o estado geral do sistema antes de reiniciar o ciclo de recepção.

3.5 DIAGRAMA DE CLASSES

A Figura 3 apresenta a visão estrutural principal do *firmware*. Diferentemente da arquitetura geral, que descreve a integração entre o simulador, a rede, o ESP32 e o painel físico, o diagrama de classes concentra-se na organização interna do software, evidenciando a classe orquestradora `App`, os serviços de aplicação, os modelos de domínio, os contratos de porta e os adaptadores concretos de infraestrutura.

Figura 3 – Diagrama de classes do *firmware*.



Fonte: Elaborada pelos autores (2026).

Do ponto de vista de responsabilidade, a classe `App` coordena o ciclo de vida do sistema; o `TelemetryService` transforma a entrada de rede em estado de telemetria confiável; o `InstrumentCluster` despacha os valores tratados para instrumentos especializados; e o `DisplayService` consolida a renderização dos *layouts* do painel. A modelagem também evidencia o uso de interfaces como `IPacketReceiver`, `ITelemetryDecoder`, `IGaugeMotor`, `IStatusDisplay` e `IWifiConfigPortal`, decisão central para reduzir acoplamento e favorecer a testabilidade do projeto.

3.5.1 EXTENSIBILIDADE PARA NOVOS JOGOS E ATUADORES

A organização estrutural apresentada no diagrama de classes também evidencia um objetivo de projeto que extrapola o MVP atual: a possibilidade de expansão controlada do sistema para novos jogos e novos elementos de saída física. Essa expansibilidade decorre da separação entre contratos e implementações concretas. No fluxo atual, a origem dos dados do jogo é abstraída por portas de telemetria, enquanto os mecanismos de atuação física são desacoplados da aplicação por contratos próprios de instrumento e acionamento.

Para novos jogos, o principal ponto de extensão está no decodificador de telemetria. Como a aplicação consome um modelo neutro de dados e não depende diretamente do formato bruto dos pacotes, a inclusão de outro simulador pode ser feita por meio de um novo adaptador com-

patível com o contrato `ITelemetryDecoder`. Nessa abordagem, o novo jogo passa a ser incorporado sem necessidade de reestruturar a lógica principal da aplicação, desde que os dados relevantes sejam traduzidos para o quadro comum utilizado pelo sistema.

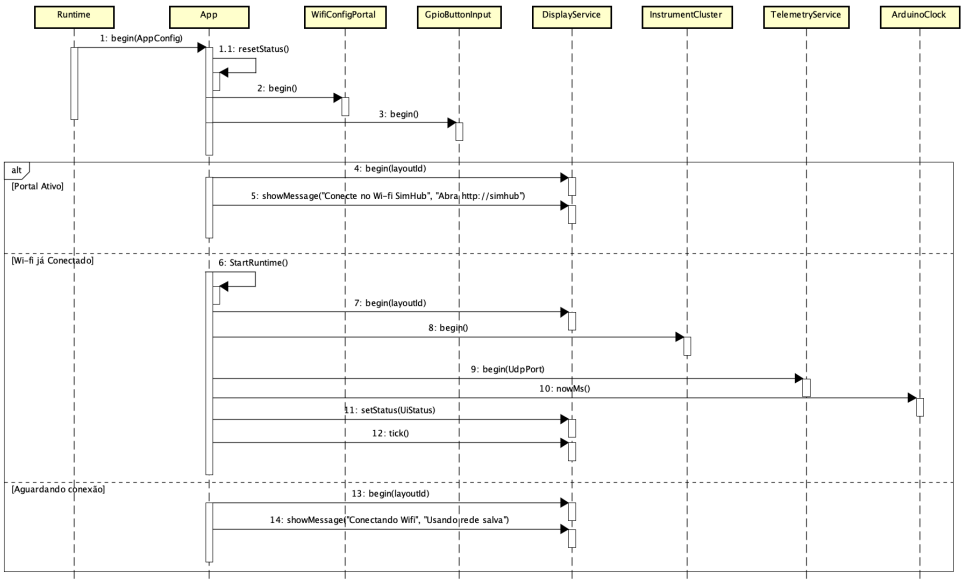
Para novos atuadores, a expansão ocorre de maneira semelhante. O `InstrumentCluster` já atua como despachante de sinais para classes especializadas que implementam o contrato `IInstrument`. Isso significa que um novo dispositivo físico pode ser incorporado como mais um consumidor dos sinais da aplicação, preservando a lógica central já existente. Quando o novo atuador puder responder a uma grandeza já disponível no sistema, a adaptação tende a ser localizada. Quando a nova resposta física depender de um evento ainda não representado no domínio, a evolução requerida concentra-se na introdução do novo sinal e em seu encaminhamento pelos contratos correspondentes, sem descaracterizar a arquitetura adotada.

3.6 DIAGRAMAS DE SEQUÊNCIA

Segundo Sommerville (2019), os diagramas de sequência são ferramentas fundamentais da modelagem dinâmica de sistemas, pois ilustram a ordem temporal em que os objetos interagem e trocam mensagens para concluir uma tarefa. Para abranger o ciclo de vida da aplicação deste projeto de forma clara, a modelagem desse fluxo foi dividida em três cenários principais: inicialização, operação da rede e processamento da telemetria.

A Figura 4 ilustra a rotina de inicialização (*boot*) do *firmware*. Nesta etapa, a classe orquestradora aciona os serviços base e tenta estabelecer conectividade. Se as credenciais de rede não estiverem disponíveis ou falharem, o sistema levanta o portal Wi-Fi de configuração. Após o sucesso da rede, os periféricos (TFT e motores) são instanciados e preparados para uso.

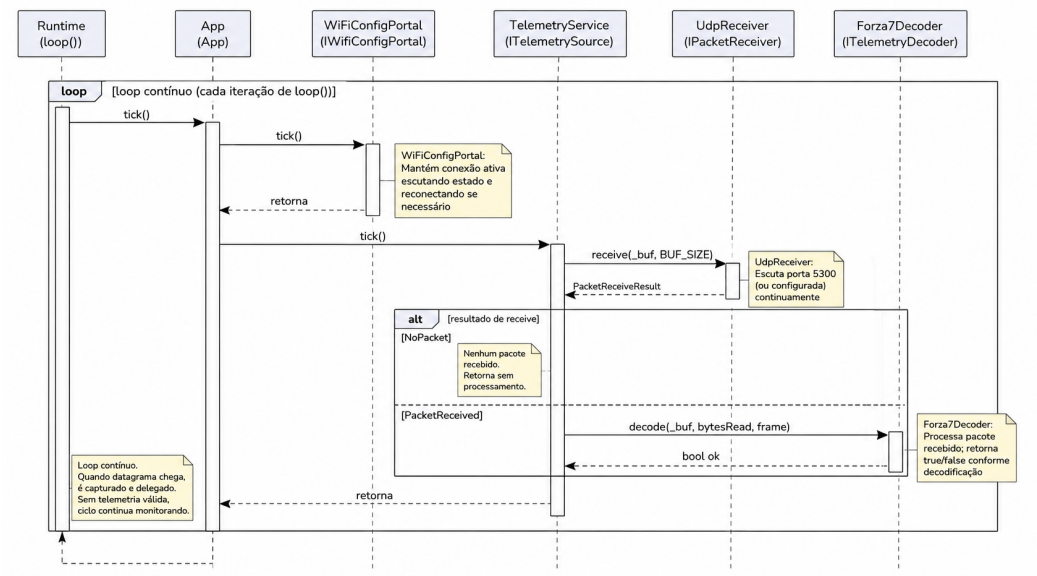
Figura 4 – Diagrama de sequência da inicialização do *firmware*.



Fonte: Elaborada pelos autores (2026).

Uma vez inicializado, o sistema entra em seu fluxo contínuo de operação, conforme detalhado na Figura 5. O *loop* principal é responsável por manter a conexão ativa e escutar a porta UDP. Quando o adaptador de rede detecta a chegada de um datagrama, ele captura o pacote e o delega para processamento.

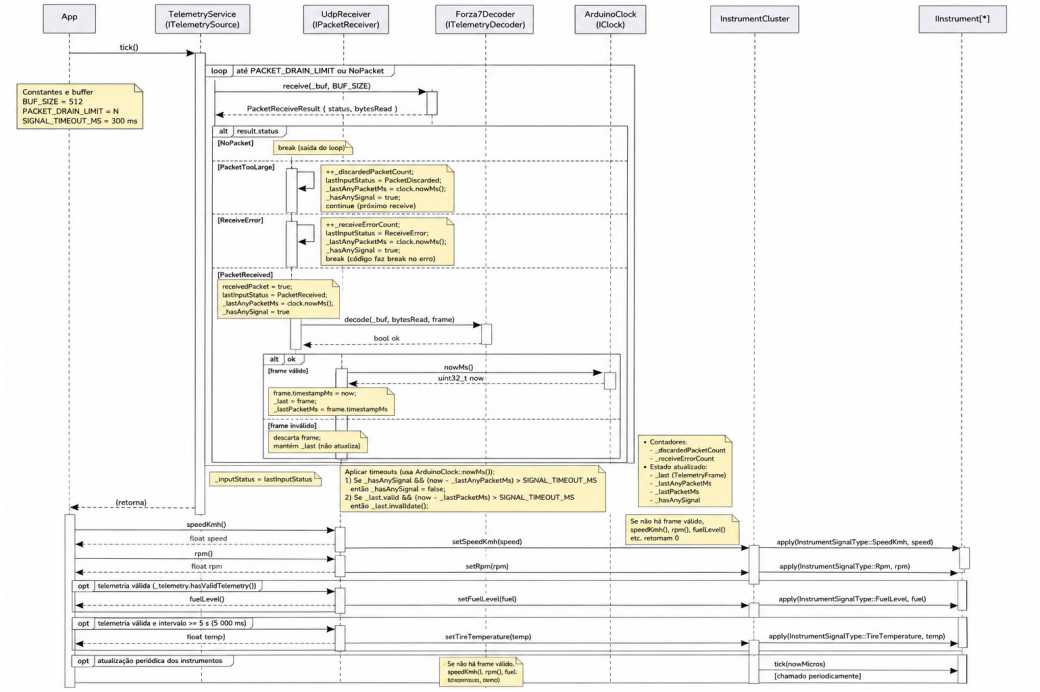
Figura 5 – Diagrama de sequência da operação principal do *firmware*.



Fonte: Elaborada pelos autores (2026).

Por fim, a Figura 6 demonstra o fluxo interno de conversão dos dados brutos em ações físicas perceptíveis. O pacote UDP recebido é submetido ao decodificador específico do jogo (`Forza7Decoder`), que extrai os *bytes* e os consolida em um modelo padronizado neutro (`TelemetryFrame`). Esse quadro é então despachado em duas frentes: para o `InstrumentCluster`, que calcula os passos do motor de passo e movimenta os ponteiros analógicos, e para o `DisplayService`, que atualiza a marcha, velocidade e afins na interface gráfica em tempo real.

Figura 6 – Diagrama de sequência de telemetria e instrumentos.



Fonte: Elaborada pelos autores (2026).

3.7 INTERFACES

A interface do sistema é inteiramente embarcada e se divide em dois contextos: mensagens de estado operacional e *layouts* de telemetria exibidos no *display* TFT. O projeto visual prioriza a legibilidade e o contraste, garantindo que o piloto identifique informações críticas de forma instantânea. As especificações de cada *layout* implementado estão consolidadas no Quadro 3.

Quadro 3 – Interfaces visuais implementadas no *display*

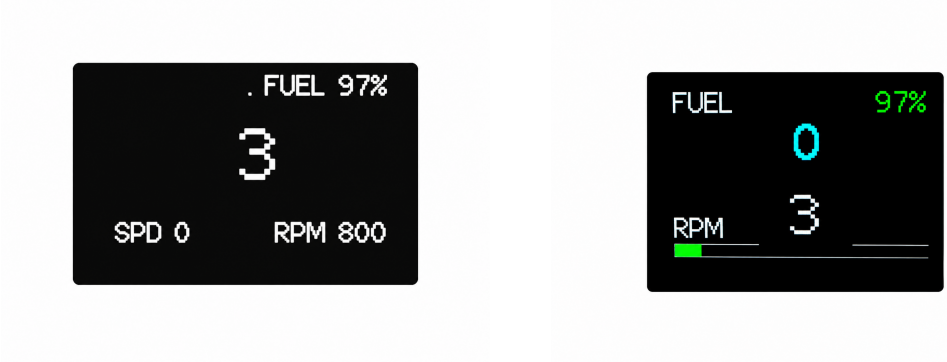
Interface	**Conteúdo principal**
`Layout0Basic`	Exibe posição de corrida, volta atual e marcha em destaque, privilegiando leitura rápida de estado competitivo.
`Layout1Minimal`	Exibe marcha central, velocidade, RPM e percentual de combustível, servindo como painel sintético geral.
`Layout2Performance`	Exibe velocidade ampliada, marcha, combustível com codificação por cor e barra de RPM.
`Layout3Tires`	Exibe temperaturas individuais `FL`, `FR`, `RL` e `RR` com codificação cromática por faixa térmica.
Telas de estado	Apresentam mensagens de conexão, orientação para acesso ao portal `SimHub` e aguardam configuração ou estabelecimento de rede.

Fonte: Elaborada pelos autores (2026).

A navegação entre os contextos é realizada por um botão físico ligado ao pino 16. O *display* opera com resolução de 240x320 pixels sob o controlador ST7789. As capturas de tela a seguir

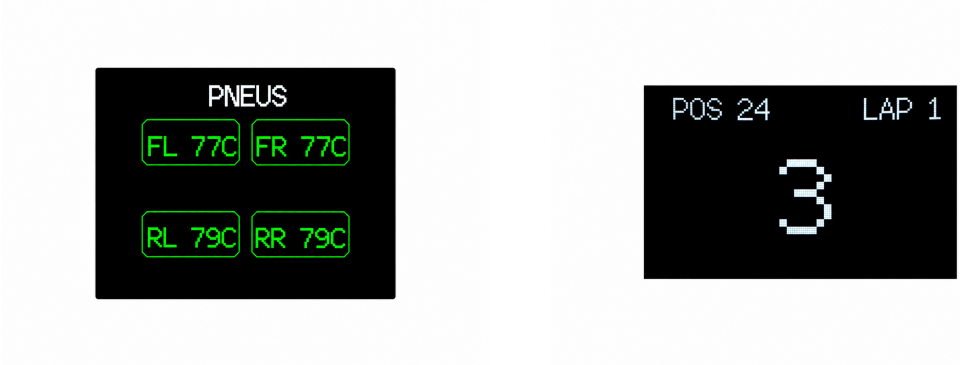
ilustram a implementação real de cada interface descrita.

Figura 7 – Interface de telemetria: Layout0Basic e Layout1Minimal.



Fonte: Elaborada pelos autores (2026).

Figura 8 – Interface de telemetria: Layout2Performance e Layout3Tires.



Fonte: Elaborada pelos autores (2026).

Figura 9 – Telas de estado e inicialização do sistema.

Fonte: Elaborada pelos autores (2026).

4 SOFTWARE

Este capítulo apresenta a materialização do software embarcado do projeto, contemplando sua implantação, seu fluxo de execução e a estratégia empregada para validar o *firmware* no computador e no ESP32. O objetivo é complementar a modelagem anterior com uma visão mais operacional do produto efetivamente entregue.

4.1 IMPLEMENTAÇÃO

A implantação do projeto corresponde ao processo de preparação, compilação, gravação e verificação inicial do firmware no ESP32. Essa etapa é importante porque conecta o código-fonte desenvolvido ao funcionamento real do protótipo, permitindo que o sistema seja executado no microcontrolador e validado de forma progressiva. No projeto, esse processo começa pela preparação do ambiente local, passa pela revisão dos arquivos de configuração, segue para a compilação e gravação do firmware e termina com a execução das suítes de teste previstas. Como parte do fluxo depende de ferramentas instaladas localmente e da organização específica do repositório, os comandos apresentados foram mantidos de acordo com os procedimentos efetivamente utilizados durante o desenvolvimento.

4.1.1 PREPARAÇÃO DO AMBIENTE E CONFIGURAÇÕES DO PROJETO

O primeiro passo consiste em obter o repositório e preparar um ambiente virtual local para as ferramentas de desenvolvimento. Esse arranjo reproduz o fluxo utilizado nas execuções documentadas do projeto e atende também ao script de cobertura, que procura o executável do `gcovr` dentro de `./.venv`. Um roteiro funcional é apresentado a seguir:

```
git clone https://github.com/CaironFerreira/TCC
cd TCC
python3 -m venv .venv
source .venv/bin/activate
pip install platformio gcovr
export PLATFORMIO_CORE_DIR=$PWD/.platformio-core
./.venv/bin/platformio --version
```

Nos exemplos seguintes, adota-se o executável `./.venv/bin/platformio`. Quando o atalho `pio` já estiver disponível no PATH, ele pode ser utilizado como equivalente. Além do ambiente Python, a máquina precisa dispor de compilador C/C++ funcional. Para a medição de cobertura registrada neste trabalho, o script `./scripts/run_native_coverage.sh` foi preparado para macOS e utiliza `xcrunllvm-covgcov`.

Antes da compilação, também é necessário revisar os arquivos `include/User_Setup.h` e `src/config/BoardConfig.h`. O arquivo `include/User_Setup.h` concentra a configuração do display ST7789, enquanto o arquivo `src/config/BoardConfig.h` define parâmetros diretamente relacionados ao funcionamento do protótipo, como porta UDP, botão de troca de layout, pinos dos motores, escalas e limites lógicos dos instrumentos. Essa verificação é necessária porque o build pode ser concluído com sucesso mesmo quando a pinagem física do protótipo não corresponde às definições presentes no código. Assim, a revisão dos arquivos de configuração evita que problemas de montagem ou parametrização sejam interpretados incorretamente como falhas de implementação.

4.1.2 COMPILAÇÃO, GRAVAÇÃO E PRIMEIRA EXECUÇÃO

Com o ambiente preparado e os arquivos de configuração revisados, a próxima etapa é compilar o firmware principal. O ambiente embarcado de produção está definido em `platformio.ini` como `esp32dev`. A compilação pode ser realizada com:

```
./.venv/bin/platformio run -e esp32dev
```

Na primeira execução, o PlatformIO pode levar mais tempo, pois precisa baixar a plataforma `espressif32`, preparar a cadeia de compilação do microcontrolador e instalar a dependência `TFT_eSPI`. Após a conclusão do build, a gravação do binário na placa pode ser feita com:

```
./.venv/bin/platformio run -e esp32dev -t upload
```

Depois do upload, o comportamento do dispositivo pode ser acompanhado pelo monitor serial configurado a 115200 bps:

```
./.venv/bin/platformio device monitor -b 115200
```

O monitor serial não é obrigatório para a gravação do programa, mas auxilia na observação de mensagens de estado, reinicializações, tentativas de conexão e possíveis falhas durante os ensaios embarcados. Com o firmware gravado, o dispositivo tenta reutilizar credenciais de rede previamente salvas. Se a conexão for bem-sucedida, o protótipo entra no modo operacional e passa a aguardar telemetria na porta UDP 5300. Em caso de falha, o ESP32 ativa o ponto de acesso `SimHub`, sem senha, e o display orienta o usuário a abrir `http://simhub` para informar o SSID e a senha da rede local.

Após a conexão Wi-Fi, o simulador deve ser configurado para enviar telemetria no formato Forza Motorsport 7 Data Out para o endereço IP do ESP32 na mesma rede local e na porta UDP 5300. A partir daí, o `UdpReceiver` recebe os pacotes, o `Forza7Decoder` extrai os campos relevantes e o `TelemetryService` distribui os valores processados para a interface visual e para os instrumentos físicos.

4.1.3 VALIDAÇÃO DA IMPLANTAÇÃO POR TESTES

A validação da implantação foi organizada de forma incremental. Primeiro, executam-se os testes no ambiente host-side, sem depender do ESP32 conectado. Em seguida, realiza-se o smoke test embarcado. Por fim, quando as condições de bancada permitem, executa-se a integração UDP real no microcontrolador. Essa sequência reduz o risco de iniciar a validação diretamente pelo hardware sem antes verificar o núcleo lógico do firmware.

O primeiro comando a ser executado é:

```
./.venv/bin/platformio test -e test_native
```

Esse ambiente reúne as suítes unitárias, de integração host-side e de componente. Ele não depende do ESP32 conectado e serve para validar o núcleo do firmware. Sua primeira execução também faz o PlatformIO baixar o framework Unity em `.pio/libdeps/test_native/`, dependência utilizada posteriormente pelo script de cobertura.

Depois da aprovação do ambiente `test_native`, recomenda-se executar o teste embarcado mais simples:

```
./.venv/bin/platformio test -e test_esp32_smoke
```

Esse comando compila a suíte de smoke, envia o binário de teste para o ESP32 e coleta o resultado pela serial. Seu objetivo é confirmar que o fluxo principal da aplicação inicia no microcontrolador e executa corretamente o caminho básico previsto pela suíte.

A etapa seguinte corresponde à integração UDP real:

```
./.venv/bin/platformio test -e test_esp32_udp_integration
```

Esse ambiente ativa Wi-Fi real no ESP32, cria o ponto de acesso `TCC-Test`, envia pacotes UDP reais para o `UdpReceiver` e verifica o comportamento conjunto de `WiFiUDP`, `Forza7Decoder` e `TelemetryService`. A documentação do projeto registra, contudo, uma limitação objetiva dessa etapa: com alimentação exclusivamente via USB, o protótipo pode entrar em `brownout` durante a ativação do rádio.

Por fim, a cobertura de código consolidada pode ser reproduzida com:

```
./scripts/run_native_coverage.sh
```

Embora exista um ambiente `test_native_coverage` no `platformio.ini`, a medição consolidada registrada neste trabalho é gerada pelo script acima. Para que ele funcione corretamente, o ambiente virtual deve existir com `gcovr` instalado e o comando `./.venv/bin/platformiotest-etest_native` deve ter sido executado ao menos uma vez, de modo que a dependência Unity esteja presente em `.pio/libdeps/test_native/`. A saída esperada é a geração de relatórios em `coverage/reports/`.

Assim, a implantação completa do projeto corresponde a um ciclo técnico verificável: preparar o ambiente, revisar as configurações do protótipo, compilar, gravar o firmware e validar seu comportamento no computador e no microcontrolador.

4.2 IMPLEMENTAÇÃO DO FIRMWARE

A implementação do *firmware* concretiza, no código-fonte, a arquitetura definida na etapa de modelagem. O sistema foi organizado de modo a separar responsabilidades entre recepção de dados, decodificação da telemetria, coordenação da aplicação, exibição visual, acionamento dos instrumentos físicos e gerenciamento da conectividade sem fio. Essa divisão favorece a manutenção do código, facilita a substituição de componentes específicos e reduz o acoplamento direto entre regras de aplicação e detalhes de *hardware*.

No contexto do projeto, o *firmware* não atua apenas como um leitor de pacotes UDP. Ele coordena um fluxo contínuo em que dados produzidos pelo simulador são recebidos, interpretados, normalizados e transformados em respostas visuais e físicas no painel. Assim, a implementação precisa lidar simultaneamente com comunicação de rede, atualização de interface, acionamento mecânico, persistência de credenciais e controle de estados de operação.

4.2.1 COMPOSIÇÃO, INICIALIZAÇÃO E CICLO DE EXECUÇÃO

O ponto de composição do *firmware* encontra-se em `src/composition/main.cpp`. Nesse arquivo são instanciados e conectados os principais elementos do sistema, incluindo o portal Wi-Fi, o receptor UDP, o decodificador de telemetria, os serviços de aplicação, o *display*, os motores e os instrumentos analógicos. Essa concentração da montagem em um único ponto facilita a compreensão da estrutura final do protótipo, pois torna explícito quais componentes participam da aplicação e como eles são associados antes do início da execução.

A composição adotada é majoritariamente estática. Objetos como `TelemetryService`, `DisplayService`, `InstrumentCluster` e `App` são declarados de forma persistente e ligados antes da entrada nos ciclos principais do ambiente Arduino, representados pelas funções `setup()` e `loop()`. Essa estratégia é adequada ao contexto embarcado, pois reduz alocações dinâmicas durante a execução e torna o comportamento do sistema mais previsível em um microcontrolador com recursos limitados.

Nesse mesmo ponto também são organizados dois catálogos centrais para o funcionamento do painel: o conjunto de *layouts* disponíveis para o *display* e o conjunto de instrumentos físicos reunidos no `InstrumentCluster`. O primeiro permite alternar diferentes formas de apresentação visual da telemetria, enquanto o segundo centraliza os instrumentos responsáveis por representar fisicamente grandezas como velocidade, rotação, combustível e temperatura. Dessa forma, o arquivo de composição funciona como o local em que a arquitetura abstrata do sistema é efetivamente transformada em uma aplicação executável.

A classe `App` atua como orquestradora principal do *firmware*. Durante a chamada de `begin()`, ela redefine o estado inicial da interface, prepara o botão físico de troca de *layout*

e aciona o portal Wi-Fi. Nessa etapa, a aplicação diferencia três situações principais: abertura imediata do portal de configuração, entrada direta em modo operacional quando há credenciais válidas ou exibição de uma tela transitória enquanto a conexão é tentada.

Após a inicialização, a rotina `tick()` passa a conduzir o ciclo contínuo do sistema. A cada iteração, a aplicação atualiza o estado do portal Wi-Fi, verifica a interação do usuário com o botão de troca de telas, aciona o serviço de telemetria, atualiza os instrumentos físicos e repassa periodicamente o estado consolidado ao *display*. Esse fluxo garante que o sistema permaneça responsivo tanto à chegada de novos pacotes quanto às ações locais do usuário.

A implementação também diferencia intervalos de atualização para tarefas distintas. A interface visual, a conectividade e o cálculo da temperatura média dos pneus não precisam ser atualizados necessariamente com a mesma frequência. Ao separar esses intervalos, o *firmware* evita processamento desnecessário, reduz oscilações visuais e preserva maior estabilidade durante a operação contínua do painel.

4.2.2 PIPELINE DE TELEMETRIA E ATUALIZAÇÃO DAS SAÍDAS

O núcleo funcional do produto está no pipeline formado por `UdpReceiver`, `Forza7Decoder` e `TelemetryService`. O `UdpReceiver` é responsável por receber os datagramas UDP enviados pelo simulador; o `Forza7Decoder` interpreta os dados conforme o formato *Forza Motorsport 7 (Data Out)*; e o `TelemetryService` organiza o resultado da decodificação em valores consumíveis pela aplicação.

Na implementação atual, o `Forza7Decoder` opera sobre deslocamentos fixos do pacote de telemetria. Essa abordagem é compatível com o formato enviado pelo simulador, no qual cada grandeza relevante ocupa posições conhecidas dentro do datagrama. Além da leitura dos campos, o decodificador já aplica transformações necessárias ao uso prático dos dados, como a conversão da velocidade para quilômetros por hora e o cálculo da temperatura média dos pneus.

O `TelemetryService` complementa esse processo ao acrescentar controle de estado sobre o fluxo de dados. Entre suas responsabilidades estão a limitação da quantidade de pacotes processados por ciclo, a contagem de descartes e erros, a identificação de presença ou ausência de sinal, a invalidação de dados por *timeout* e a normalização dos valores antes do repasse à camada superior. Com isso, a aplicação não depende diretamente da leitura bruta dos pacotes UDP, mas de um estado de telemetria já tratado e coerente para uso no painel.

Depois do processamento, os dados são distribuídos por duas rotas principais: a interface visual e os instrumentos físicos. Na interface, o `DisplayService` mantém o catálogo de *layouts*, inicializa o *display* quando necessário e controla a troca de telas conforme a interação do usuário. Cada *layout* representa uma forma específica de apresentar a telemetria, permitindo adaptar a visualização às informações mais relevantes em cada contexto de uso.

No plano físico, o `InstrumentCluster` atua como despachante dos sinais destinados aos instrumentos analógicos. Em vez de a aplicação acionar diretamente cada motor ou cada instrumento, ela encaminha os valores consolidados ao agrupador, que distribui cada grandeza

apenas aos instrumentos compatíveis. Essa separação evita que a lógica principal da aplicação fique dependente dos detalhes de acionamento mecânico e preserva a possibilidade de adicionar, remover ou substituir instrumentos com menor impacto sobre o restante do código.

Essa organização é coerente com o objetivo central do projeto: converter a telemetria digital do simulador em uma experiência híbrida, composta por resposta gráfica e resposta física. O *firmware*, portanto, não se limita a exibir números em tela, mas transforma os dados recebidos em comportamento perceptível no painel, aproximando o protótipo da experiência de um painel automotivo dedicado.

4.2.3 CONECTIVIDADE, PERSISTÊNCIA E SUPORTE OPERACIONAL

A conectividade sem fio é um elemento essencial da implementação, pois a recepção da telemetria depende de o ESP32 estar conectado à mesma rede do equipamento que executa o simulador. Para reduzir a necessidade de recompilação sempre que o ambiente de rede muda, foi implementado o componente `WiFiConfigPortal`. Esse componente permite configurar as credenciais de rede por meio de uma interface local, tornando o protótipo mais adequado ao uso prático em diferentes ambientes.

Durante a inicialização, o `WiFiConfigPortal` procura credenciais previamente persistidas, tenta estabelecer conexão com a rede configurada e, em caso de falha, ativa automaticamente um ponto de acesso de configuração. Nesse modo, o usuário pode acessar a interface web local e informar o SSID e a senha da rede desejada. Após o sucesso da conexão, as credenciais são armazenadas para reutilização em execuções futuras, evitando que o usuário precise repetir o procedimento a cada reinicialização do dispositivo.

Além da abertura do portal de configuração, o componente administra a leitura e gravação de dados em `Preferences`, diferencia estados de conexão, processa requisições HTTP, encerra o ponto de acesso após a configuração bem-sucedida e mantém tentativas de reaproveitamento das credenciais salvas. Dessa forma, a conectividade deixa de ser tratada como uma configuração fixa inserida no código-fonte e passa a integrar o comportamento operacional do próprio produto.

Esse mecanismo é importante porque o protótipo foi concebido para funcionar como um dispositivo embarcado independente. Em um cenário de uso real, não seria adequado exigir alterações no código ou nova gravação do *firmware* sempre que o painel fosse utilizado em outra rede. A presença do portal Wi-Fi, portanto, aproxima o sistema de uma solução mais autônoma e reduz a dependência de intervenção técnica para sua utilização.

4.2.4 PONTOS DE EXPANSÃO DO FIRMWARE

A implementação atual também define pontos de extensão que permitem evoluir o projeto sem reestruturar toda a arquitetura. No caso de suporte a novos jogos, o caminho natural consiste em adicionar um novo par de arquivos em `src/adapters/telemetry/`, seguindo o padrão do `Forza7Decoder` e implementando o contrato `ITelemetryDecoder`. Se o

protocolo do novo simulador disponibilizar apenas os campos já contemplados pelo sistema, o novo adaptador pode traduzir diretamente esses dados para o modelo `TelemetryFrame`. Caso o novo protocolo ofereça informações adicionais relevantes, a ampliação passa por acrescentar novos campos em `src/domain/TelemetryFrame.h` e, quando necessário, expô-los também por meio do serviço de telemetria consumido pela aplicação.

Após a criação do novo decodificador, sua integração ocorre no ponto de composição principal, em `src/composition/main-.cpp`, onde atualmente é instanciado o `Forza7Decoder`. Essa escolha mantém a troca ou adição de suporte a outro jogo concentrada na borda de entrada da arquitetura. Para preservar a confiabilidade do sistema durante essa evolução, a inclusão de um novo decodificador deve ser acompanhada por testes equivalentes aos existentes para o protocolo atual. Esses testes devem verificar tanto a leitura dos campos do pacote quanto o comportamento integrado do pipeline de telemetria, garantindo que os dados continuem sendo normalizados e distribuídos corretamente à aplicação.

A expansão para novos atuadores segue lógica semelhante. Quando o novo dispositivo puder operar a partir de uma grandeza já tratada pelo sistema, como velocidade, rotação, combustível ou temperatura, basta implementar uma nova classe em `src/application/instruments/` compatível com o contrato `IInstrument` e registrá-la no arranjo do `InstrumentCluster` em `src/composition/main.cpp`. Se esse atuador exigir um mecanismo físico de acionamento ainda não contemplado, um adaptador correspondente pode ser acrescentado em `src/adapters/motors/`, seguindo o mesmo princípio utilizado para encapsular detalhes concretos de *hardware*.

Para respostas como vibração, alerta tátil ou outros efeitos hápticos, a expansão também permanece localizada. Se o comportamento puder ser derivado de um sinal já presente no sistema, a inclusão resume-se à implementação do novo instrumento e ao seu registro na composição. Se a atuação depender de um evento ainda não modelado, como perda de aderência ou saída de pista, o procedimento envolve acrescentar esse evento ao modelo de telemetria e ao encaminhamento de sinais da aplicação. Ainda assim, trata-se de uma evolução incremental sobre contratos já definidos, e não de uma reformulação ampla do *firmware*.

4.3 TESTES E VALIDAÇÃO

A estratégia de testes do projeto foi organizada em cinco camadas: testes unitários, integração *host-side*, teste de componente, *smoke test* embarcado e integração UDP real no ESP32. Essa divisão foi adotada para permitir regressão rápida no computador sem abrir mão de evidências mínimas em *hardware* real. A validação incremental também é coerente com recomendações de engenharia de software para redução de falhas e aumento da confiabilidade do produto (Sommerville, 2019). Os diveros tipos de teste com suas respectivas suítes, casos e situação são apresentados no 4 a seguir.

Quadro 4 – Quadro consolidado de testes do projeto

Tipo de teste	**Suítes**	**Casos**	**Situação**
Unitário	12	51	Implementado, executado e aprovado em `test_native`.
Integração *host-side*	3	9	Implementado, executado e aprovado em `test_native`.
Componente	1	8	Implementado, executado e aprovado em `test_native`.
Smoke embarcado	1	2	Implementado, executado e aprovado em `test_esp32_smoke`.
Integração embarcada real	1	3	Implementado; *build* e transferência para a placa validados; execução final limitada pela bancada elétrica.

Fonte: Elaborada pelos autores (2026).

Os documentos de validação do repositório registram 18 suítes e 73 casos definidos, dos quais 70 já tiveram execução concluída com evidência coletada. A diferença entre o total documentado e a execução *host-side* decorre do fato de que `test_native` cobre apenas as suítes seguras para execução no computador, enquanto os demais casos pertencem aos ambientes embarcados. Em verificação local realizada em 4 de maio de 2026, a suíte `test_native` foi executada novamente no estado atual do repositório e produziu 68/68 casos aprovados em aproximadamente 13 segundos. Esse resultado reafirma a integridade da automação *host-side* na versão final utilizada para a escrita deste relatório.

Quanto à cobertura real de código, a última medição consolidada registrada no repositório aponta os percentuais apresentados na Tabela 2. Esses dados foram produzidos pelo script `./scripts/run_native_coverage.sh`.

Tabela 2 – Resumo da cobertura *host-side* registrada no projeto

Grupo	**Linhas**	**Funções**	**Branches**
Unitário	66,1%	79,5%	41,5%
Integração	13,4%	21,1%	6,6%
Componente	12,7%	15,3%	4,9%
Consolidado *host-side*	76,9%	84,8%	46,2%

Fonte: Elaborada pelos autores (2026).

Do ponto de vista qualitativo, os testes cobrem os pontos tecnicamente mais sensíveis do MVP: recepção UDP, deslocamentos do protocolo, conversões de unidade, tratamento de *timeout*, contagem de erros, filtros de entrada, suavização de instrumentos, mudança de *layout*, transições entre portal Wi-Fi e modo operacional e renderização básica da interface. A evidência embarcada já confirma o *smoke test* no ESP32; por outro lado, a integração UDP real permanece parcialmente limitada por restrições físicas da bancada, especialmente condições de queda de tensão quando o rádio Wi-Fi é ativado sob alimentação exclusivamente via USB.

5 HARDWARE

Este capítulo descreve os elementos físicos utilizados na construção do painel de telemetria e a forma como esses componentes foram organizados para sustentar a execução do sistema embarcado. O objetivo não é apenas listar materiais, mas demonstrar como processamento, acionamento, interface visual e alimentação foram integrados para transformar a telemetria recebida em respostas físicas e visuais no protótipo.

No contexto deste trabalho, o *hardware* foi definido em função das exigências do *firmware* já apresentado: recepção de telemetria por Wi-Fi, renderização em *display* SPI, leitura de botão físico e acionamento simultâneo de quatro instrumentos analógicos. Por isso, a seleção dos componentes considerou compatibilidade elétrica, disponibilidade de GPIOs, capacidade de montagem no painel automotivo e viabilidade de expansão futura do protótipo.

A disposição visual dos itens empregados na montagem do painel físico é ilustrada na Figura 10. Em complemento, as especificações técnicas e as respectivas quantidades de cada componente estão consolidadas no Quadro 5.

Figura 10 – Componentes utilizados na confecção do produto.

Fonte: Elaborada pelos autores (2026).

Quadro 5 – Lista de componentes de *hardware* utilizados no projeto

Quantidade	**Componente e especificações**
1	Painel de instrumentos automotivo
4	Módulos *driver* TMC2208 para controle de motores de passo
1	*Display* TFT colorido de 2,4 polegadas, com interface SPI, resolução de 240 × 320 pixels e controlador ST7789
1	Botão físico
1	Fonte de alimentação de 5 V / 2 A
1	Placa ESP32 Dev Module com conectividade Wi-Fi
Vários	Fios e *jumpers* para interligação dos componentes
1	Regulador de tensão LM2596
1	Conector de alimentação P4 fêmea
2	Entradas/conectores RJ45

Fonte: Elaborada pelos autores (2026).

O Quadro 5 evidencia que o protótipo foi organizado em quatro blocos físicos principais: processamento e conectividade, interface visual, atuação eletromecânica e alimentação. Essa decomposição ajuda a compreender que o painel não depende de um único componente central apenas, mas de uma integração coordenada entre placa controladora, periféricos de saída e elements de suporte elétrico.

No bloco de processamento e conectividade, o ESP32 Dev Module atua como unidade de execução do *firmware* e como ponto de convergência das interfaces do sistema. É nele que se concentram a recepção da telemetria via rede sem fio, a leitura do botão físico de troca de *layout* e o controle dos periféricos conectados aos GPIOs. Assim, a escolha dessa placa não se justifica apenas pela presença de Wi-Fi embarcado, mas também pela necessidade de reunir comunicação, temporização e acionamento em uma mesma plataforma compacta.

No bloco de interface visual, o *display* ST7789 de 240 × 320 pixels cumpre a função de complementar os instrumentos analógicos com informações que não seriam adequadamente representadas apenas por ponteiros, como marcha, posição, volta e estados operacionais do sistema. Sua adoção também se mostrou coerente com o espaço disponível na estrutura original do painel, permitindo inserir uma saída digital sem descaracterizar a aparência geral do conjunto.

No bloco de atuação eletromecânica, os quatro módulos TMC2208 dão suporte ao acionamento independente dos indicadores de velocidade, rotação do motor, combustível e temperatura média dos pneus. Essa organização acompanha diretamente a modelagem implementada no *firmware*, que trata cada grandeza com pinos dedicados, limites próprios e rotinas de calibração específicas. Em termos construtivos, isso significa que a expansão para novos instrumentos pode ocorrer de forma incremental, desde que haja disponibilidade física de interface e estratégia compatível de acionamento.

Por fim, o bloco de alimentação e interligação tem papel decisivo na estabilidade do protótipo. A combinação entre fonte externa de 5 V / 2 A, conector P4 fêmea e regulador LM2596 foi adotada para sustentar o ESP32 e os demais módulos sem depender exclusivamente da ali-

mentação USB utilizada em etapas de gravação e teste. Essa decisão torna-se especialmente relevante diante do comportamento observado na validação embarcada, em que a ativação do Wi-Fi sob alimentação restrita pode comprometer a estabilidade elétrica. Já os conectores RJ45 foram incorporados como interfaces físicas auxiliares para organização do cabeamento e para possíveis expansões externas, sem alterar o fato de que, na versão atual do produto, o fluxo principal de telemetria ocorre por rede sem fio.

6 RESULTADOS OBTIDOS

Este capítulo apresenta o resultado final obtido a partir da integração entre a modelagem do sistema, o software embarcado e os componentes físicos selecionados. O objetivo é demonstrar como as decisões técnicas descritas nos capítulos anteriores foram concretizadas em um protótipo funcional, capaz de reunir painel automotivo adaptado, instrumentos analógicos, interface visual embarcada, botão de interação e conexões externas em uma solução única de telemetria para simuladores de corrida.

A montagem final do protótipo teve como premissa preservar a estética e a ergonomia originais do painel automotivo, acondicionando os novos circuitos internamente e disponibilizando as interfaces de uso e alimentação de forma acessível. A Figura 11 ilustra a visão frontal do dispositivo finalizado. O principal destaque desta face é a integração do *display* TFT de 2,4 polegadas na porção central inferior, ocupando harmonicamente o espaço destinado ao antigo visor digital do veículo. Os ponteiros analógicos responsáveis pelos indicadores de velocidade, rotação (RPM), temperatura e nível de combustível foram mantidos e acoplados aos motores de passo controlados pelo sistema embarcado.

Figura 11 – Visão frontal do painel com o *display* integrado.

Fonte: Elaborada pelos autores (2026).

Para viabilizar a interação do usuário com o sistema especificamente a alternância entre os *layouts* de telemetria apresentados no capítulo anterior , optou-se pela instalação de um botão físico embutido na parte superior da carcaça, conforme demonstrado na Figura 12. Esta posição estratégica permite um acionamento rápido e confortável durante a operação no simulador, sem

obstruir a visão da tela.

Figura 12 – Visão superior detalhando o botão de controle de interface.

Fonte: Elaborada pelos autores (2026).

Por fim, a modificação estrutural traseira do painel foi projetada para concentrar as conexões externas, garantindo uma instalação limpa na bancada do simulador. A Figura 13 apresenta a organização dessas interfaces, onde se observa a adição de um conector P4 fêmea centralizado, responsável por receber a alimentação da fonte externa de 5 V. Na porção superior esquerda, foram embutidas duas portas RJ45, utilizadas como interfaces físicas auxiliares para organização do cabeamento e possibilidades de expansão do protótipo. Na versão atual do produto, contudo, o fluxo principal de telemetria permanece baseado em comunicação Wi-Fi.

Figura 13 – Visão traseira do painel com as conexões de alimentação e interfaces externas.

Fonte: Elaborada pelos autores (2026).

De modo geral, o acondicionamento dos componentes (placa ESP32, módulos *driver* e regulador de tensão) no interior da estrutura original mostrou-se bem-sucedido, resultando em um dispositivo de telemetria robusto, funcional e com apelo visual idêntico ao de um painel automotivo real.

6.1 ANÁLISE DE CUSTOS E VIABILIDADE COMERCIAL

Para que a solução desenvolvida cumpra plenamente a justificativa de contornar as barreiras financeiras impostas por plataformas comerciais de alto custo, é fundamental demonstrar a viabilidade econômica do projeto. Dessa forma, realizou-se o levantamento dos custos diretos envolvidos na construção do protótipo, bem como uma projeção para fabricação em pequena escala e estimativa de valor de venda. A Tabela 3 detalha os valores investidos na aquisição dos componentes eletrônicos e estruturais listados anteriormente no projeto.

Tabela 3 – Custo direto estimado para a montagem do protótipo

Componente	**Quantidade**	**Valor Unitário**	**Subtotal**
Painel de instrumentos automotivo usado	1	R$ 0,00	R$ 0,00
Módulo driver TMC2208	4	R$ 24,02	R$ 96,08
Display TFT 2,4", 240x320, ST7789	1	R$ 32,04	R$ 32,04
Placa ESP32 DevKit V1	1	R$ 34,49	R$ 34,49
Botão físico tipo *push button*	1	R$ 14,84	R$ 14,84
Fonte de alimentação 5 V / 2 A	1	R$ 20,14	R$ 20,14
Fios e *jumpers* para interligação	1 lote	R$ 19,00	R$ 19,00
Regulador de tensão LM2596	1	R$ 14,12	R$ 14,12
Conector de alimentação P4 fêmea	1	R$ 3,65	R$ 3,65
Conectores RJ45 para painel	2	R$ 19,23	R$ 38,46
		Custo direto estimado	**R$ 257,98**

Fonte: Elaborada pelos autores (2026).

O custo total de R$ 257,98 reflete a materialização do Produto Mínimo Viável (MVP). Cabe ressaltar que a carcaça do painel automotivo não gerou custos nesta etapa, tratando-se de uma peça de reaproveitamento, o que contribuiu para manter o valor final reduzido. Contudo, para a transição do estágio de prototipagem para a fabricação em série, o modelo de montagem sofreria adequações voltadas à otimização e padronização.

No cenário de fabricação, o lote de fios e *jumpers* seria substituído por uma Placa de Circuito Impresso (PCB) dedicada. Uma cotação preliminar para a manufatura de 20 unidades de PCBs customizadas indicou um custo total de US$ 35,00, o que representa aproximadamente US$ 1,75 (menos de R$ 10,00) por placa. Essa substituição não apenas absorve o custo atual dos fios, como reduz drasticamente o tempo de soldagem, a margem de erro humano e o risco de falhas por mau contato, aumentando a confiabilidade do produto final. Ademais, a carcaça reciclada daria lugar a um invólucro padronizado, produzido via modelagem e impressão 3D ou injeção plástica, o que adicionaria um custo estimado de R$ 60,00 a R$ 90,00 por unidade, dependendo do material e volume de produção.

Considerando os ajustes para manufatura, horas técnicas de montagem, impostos e margem de lucro, estima-se que o painel físico de telemetria desenvolvido neste trabalho possa ser comercializado por um valor final situado entre R$ 650,00 e R$ 850,00. Esse posicionamento de preço confirma o atendimento ao problema de pesquisa levantado na introdução. Enquanto soluções comerciais que integram instrumentação analógica real e *displays* de telemetria costumam ultrapassar a faixa de milhares de reais (frequentemente associadas a custos de importação), o sistema proposto entrega um *feedback* imersivo, ergonômico e de arquitetura expansível por uma fração do custo de mercado.

7 CONCLUSÃO

O desenvolvimento realizado neste trabalho permitiu atingir, no escopo definido para o MVP, o objetivo de conceber e validar um painel físico de telemetria capaz de ampliar a percepção sensorial do usuário em simuladores de corrida. A solução entregue demonstrou ser capaz de receber dados operacionais do jogo por meio da rede local, interpretá-los no sistema embarcado e convertê-los em respostas físicas e visuais no painel. Com isso, o trabalho não permaneceu no plano conceitual da modelagem, mas resultou em um protótipo funcional, materializado em um painel automotivo adaptado com instrumentos analógicos ativos, *display* embarcado, conectividade sem fio e interação local por botão físico.

Do ponto de vista dos objetivos específicos, o projeto consolidou a adaptação estrutural do painel automotivo, a integração dos indicadores de velocidade, rotação, combustível e temperatura, a implementação do portal Wi-Fi para configuração e reconexão, a recepção contínua de telemetria via UDP e a decodificação do formato *Forza Motorsport 7 (Data Out)*. Também se confirmou, em nível de produto entregue, a capacidade de representar estados operacionais no *display* e de converter sinais digitais em movimentação física dos ponteiros. Esse conjunto de resultados mostra que a proposta central do trabalho foi efetivamente transformada em uma solução utilizável e tecnicamente coerente com o escopo inicialmente delimitado.

Uma contribuição relevante do trabalho está na forma como a solução foi organizada em termos de engenharia de software e integração embarcada. Em vez de concentrar toda a lógica em um código monolítico, o *firmware* foi estruturado com separação entre domínio, serviços de aplicação, portas e adaptadores, o que favoreceu manutenção, testabilidade e evolução controlada. Essa organização permitiu isolar recepção de telemetria, decodificação, atualização da interface e acionamento dos instrumentos em componentes especializados. Na prática, isso significa que o valor do trabalho não se resume ao protótipo físico demonstrado, mas inclui uma base arquitetural que sustenta expansão futura para novos simuladores, novos *decoders* e novos atuadores sem exigir reestruturação ampla do sistema.

No campo da validação, os resultados também foram consistentes com a proposta do relatório. A existência de testes unitários, de integração *host-side*, de componente e de validação embarcada elevou o grau de confiança técnica sobre o comportamento do *firmware*. A execução aprovada da suíte `test_native`, somada ao *smoke test* validado no ESP32 e à preparação da integração UDP real, demonstra que o projeto foi tratado não apenas como montagem experimental, mas como produto de software embarcado submetido a verificação progressiva. Do ponto de vista físico, a integração final no painel mostrou-se satisfatória ao preservar a estética automotiva original e, ao mesmo tempo, incorporar com sucesso os recursos eletrônicos necessários ao funcionamento do sistema.

As limitações do trabalho, contudo, permanecem relevantes e precisam ser registradas com clareza. A implementação atual concentra-se em um único protocolo de jogo, correspondente

ao formato *Forza Motorsport 7 (Data Out)*, e ainda não fecha compatibilidade formal com múltiplas plataformas. Além disso, os efeitos adicionais de vibração previstos no escopo inicial não foram incorporados ao MVP final, e a validação completa da integração UDP embarcada continua condicionada a uma bancada elétrica mais estável, capaz de eliminar o risco de queda de tensão durante a ativação do Wi-Fi. Essas restrições não invalidam o resultado alcançado, mas delimitam com precisão o estágio de maturidade da solução apresentada.

Como continuidade natural, os próximos passos do projeto consistem em concluir a validação embarcada em cenário elétrico estável, medir o comportamento temporal do sistema em uso real, ampliar o suporte para novos simuladores e evoluir o painel com novos modos de atuação física, inclusive efeitos hápticos adicionais. A arquitetura já implementada e documentada oferece base concreta para essa evolução. Assim, conclui-se que o trabalho entrega uma solução funcional, modular e academicamente defensável, ao mesmo tempo em que estabelece fundamentos técnicos consistentes para futuras ampliações do produto.

REFERÊNCIAS

AL-FUQAHA, A. et al. Internet of things: a survey on enabling technologies, protocols, and applications. **IEEE Communications Surveys & Tutorials**, v. 17, n. 4, p. 2347–2376, 2015.

BUGEJA, K.; SPINA, S.; BUHAGIAR, F. Telemetry-based optimisation for user training in racing simulators. In: **2017 9th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games)**. [S. l.: s. n.], 2017. p. 31–38.

CALLEO, A.; DALL'OSSO, G.; ZANNONI, M. A Systematic Analysis for Multisensory Virtual Artifacts Design in Immersive E-Sport Applications and Sim-Racing. *IRIS - Institutional Research Information System*, 2023. Disponível em: <https://cris.unibo.it/handle/11585/934615>. Acesso em: 10 maio 2026.

COCKBURN, A. Hexagonal architecture. 2005. Disponível em: <https://alistair.cockburn.us/hexagonal-architecture/>. Acesso em: 10 maio 2026.

DE FRUTOS, S. H.; CASTRO, M. Assessing sim racing software for low-cost driving simulator to road geometric research. *ResearchGate*, 2021. Disponível em: <https://www.researchgate.net/publication/356884957>. Acesso em: 10 maio 2026.

FRIEDRICH, L. F. et al. A survey of configurable, component-based operating systems for embedded applications. **IEEE Micro**, v. 21, n. 3, p. 54–68, 2001.

GLOBAL MARKET INSIGHTS. **Racing Simulator Market Size & Share 2024–2032**. 2024. Disponível em: <https://www.gminsights.com/industry-analysis/racing-simulator-market>. Acesso em: 24 maio 2026.

HUANG, Y. et al. Design validation testing of vehicle instrument cluster using machine vision and hardware-in-the-loop. In: **2008 IEEE International Conference on Vehicular Electronics and Safety (ICVES)**. [S. l.: s. n.], 2008. p. 265–270.

KOHWALTER, T. C.; MURTA, L. G. P.; CLUA, E. W. G. Capturing game telemetry with provenance. In: **2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)**. [S. l.: s. n.], 2017. p. 66–75.

LOVE, T. S.; ASEMPAPA, R. S. Examining middle school students' attitudes toward computing after participating in a physical computing unit. *ResearchGate*, 2022. Disponível em: <https://www.researchgate.net/publication/369593809>. Acesso em: 10 maio 2026.

MARTIN, R. C. **Clean architecture**: a craftsman's guide to software structure and design. Boston: Prentice Hall, 2017.

MURPHY, C. J.; TOTH, A. J.; HOJAJI, F.; CAMPBELL, M. J. Force feedback drives sim racing performance: The influence of force and vibrotactile haptic feedback in simulator steering wheels. *Computers in Human Behavior Reports*, v. 22, p. 100993, 2026.

ROY, S.; MISRA, S.; RAGHUWANSHI, N. S. SensPnP: seamless integration of heterogeneous sensors with IoT devices. **IEEE Transactions on Consumer Electronics**, v. 65, n. 2, p. 203–213, 2019.

SOMMERVILLE, I. **Engenharia de software**. 10. ed. São Paulo: Pearson Universidades, 2019.

VERGARA, K.; HERSKOVIC, V. Boxy Board: Supporting Novice Learners Through a Progressive Disclosure Approach for Physical Computing. *International Journal of Human–Computer Interaction*, p. 1–22, 2025. DOI: <https://doi.org/10.1080/10447318.2025.2563744>.

WAZLAWICK, R. *Metodologia de pesquisa para ciência da computação*. 2. ed. Rio de Janeiro: Elsevier Brasil, 2017.

APÊNDICE A – MANUAL DE CONFIGURAÇÃO RÁPIDA

Fonte: Elaborada pelos autores (2026).

ANEXO A – DECLARAÇÃO DE ENTREGA DO CÓDIGO-FONTE

Declaro que o código-fonte correspondente a este trabalho foi entregue à coordenação do curso e encontra-se organizado no repositório técnico utilizado na elaboração deste relatório.

gov.br
Documento assinado digitalmente
CAIRON FERREIRA PRADO
Data: 05/08/2026 20:14:29-0300
Verifique em https://validar.iti.gov.br

Assinatura: _____

Data: 15/07/2026

ANEXO B – DECLARAÇÃO DE SUBMISSÃO AO NIT

Declaro que as providências institucionais relativas à submissão do software ao NIT – IFPI foram adotadas conforme as exigências aplicáveis ao presente trabalho.

Documento assinado digitalmente
gov.br
CAIRON FERREIRA PRADO
Data: 05/08/2026 20:12:47-0300
Verifique em https://validar.iti.gov.br

Assinatura: _____

Data: 15/07/2026