



**UNIVERSIDADE FEDERAL DO CEARÁ  
CENTRO DE TECNOLOGIA  
DEPARTAMENTO DE ENGENHARIA DA COMPUTAÇÃO  
CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO**

**INÁCIO RODRIGUES DE MATOS GALVÃO**

**IMPLEMENTAÇÃO DE SISTEMA DE TELEOPERAÇÃO COM FEEDBACK  
HÁPTICO BASEADO EM RASPBERRY PI E ARDUINO**

**FORTALEZA  
2026**

INÁCIO RODRIGUES DE MATOS GALVÃO

IMPLEMENTAÇÃO DE SISTEMA DE TELEOPERAÇÃO COM FEEDBACK HÁPTICO  
BASEADO EM RASPBERRY PI E ARDUINO

Trabalho de Conclusão de Curso apresentado  
ao Curso de Graduação em Engenharia da  
Computação do Centro de Tecnologia da  
Universidade Federal do Ceará, como requisito  
parcial à obtenção do grau de bacharel em  
Engenharia da Computação.

Orientador: Prof. Dr. José Marques Soa-  
res

FORTALEZA

2026

INÁCIO RODRIGUES DE MATOS GALVÃO

IMPLEMENTAÇÃO DE SISTEMA DE TELEOPERAÇÃO COM FEEDBACK HÁPTICO  
BASEADO EM RASPBERRY PI E ARDUINO

Trabalho de Conclusão de Curso apresentado  
ao Curso de Graduação em Engenharia da  
Computação do Centro de Tecnologia da  
Universidade Federal do Ceará, como requisito  
parcial à obtenção do grau de bacharel em  
Engenharia da Computação.

Aprovada em:

BANCA EXAMINADORA

---

Prof. Dr. José Marques Soares (Orientador)  
Universidade Federal do Ceará (UFC)

---

Prof. Dr. XXXXXXXX XXXXXX XXXXXXXX  
Universidade do Membro da Banca Dois (SIGLA)

---

Prof. Dr. XXXXXXXX XXXXXX XXXXXXXX  
Universidade do Membro da Banca Três (SIGLA)

---

Prof. Dr. XXXXXXXX XXXXXX XXXXXXXX  
Universidade do Membro da Banca Quatro (SIGLA)

À minha família, pelo cuidado e oportunidades que me deram. Mãe e pai, a segurança de ter vocês me possibilitou superar minhas expectativas. À Marina, que esteve ao meu lado em cada etapa deste projeto.

## **AGRADECIMENTOS**

Ao Prof. Dr. José Marques Soares por me orientar em minha tese de graduação.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

Agradeço a todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender.

"Nenhuma quantidade de dinheiro pode comprar  
um segundo de tempo."

(Tony Stark, Vingadores: Ultimato (2019))

## RESUMO

Este trabalho apresenta o desenvolvimento de um sistema completo de controle remoto de veículos com interface háptica utilizando protocolo UDP para comunicação de baixa latência. O objetivo foi projetar e implementar uma arquitetura de três camadas compreendendo: veículo teleoperado baseado em Raspberry Pi 4 com câmera OV5647 e sensores BMI160; cliente em Python/Tkinter para interface gráfica e comunicação UDP otimizada para latência inferior a 5ms; e simulador físico com volante, pedais e force feedback controlado via ESP32. A metodologia empregou desenvolvimento incremental e modular com prototipagem evolutiva, implementando algoritmos simplificados de force feedback em tempo real para cálculo de forças G, detecção de vibrações e controle de atuadores através de mapeamento linear PWM, validados por métricas de performance, qualidade de vídeo e experiência do usuário. Durante sessão experimental de 15 minutos foram coletados mais de 90.000 pontos de telemetria e 26.925 frames de vídeo, alcançando latência média de  $1.94\text{ms} \pm 0.41\text{ms}$  (2.6x melhor que targets típicos de 5ms), FPS de 29.9 (3x superior ao estado da arte de 10 FPS), packet loss inferior a 0.28%, e detecção eficaz de 11.000 eventos de force feedback com precisão de 97.2% no cálculo de forças G. Os resultados demonstram que a abordagem simplificada com UDP simples e algoritmos diretos de force feedback supera protocolos complexos como UDP-RT para aplicações de teleoperação de baixo custo, validando a viabilidade técnica e comercial do sistema com custo total de R\$ 1.300 e eficiência energética de 123.8 MB/Wh, estabelecendo uma solução acessível para controle remoto com feedback háptico em tempo real.

**Palavras-chave:** Controle remoto. Force feedback. Protocolo UDP. Sistemas embarcados. Raspberry Pi. Interface háptica.

## ABSTRACT

This work presents the development of a complete remote vehicle control system with haptic interface utilizing UDP protocol for low-latency communication. The objective was to design and implement a three-layer architecture comprising: a remotely controlled vehicle based on Raspberry Pi 4 with OV5647 camera and BMI160 sensors; UDP communication optimized for latency below 5ms; and a physical simulator with steering wheel, pedals and force feedback controlled via ESP32 integrated with a Python/Tkinter graphical interface. The methodology employed incremental and modular development with evolutionary prototyping, implementing simplified real-time force feedback algorithms for G-force calculation, vibration detection, and actuator control through linear PWM mapping, validated by performance metrics, video quality, and user experience assessments. During a 15-minute experimental session, over 90,000 telemetry data points and 26,925 video frames were collected, achieving an average latency of  $1.94\text{ms} \pm 0.41\text{ms}$  (2.6x better than typical 5ms targets), 29.9 FPS (3x superior to the state-of-the-art 10 FPS), packet loss below 0.28%, and effective detection of 11,000 force feedback events with 97.2% accuracy in G-force calculations. The results demonstrate that the simplified approach using plain UDP and direct force feedback algorithms outperforms complex protocols like UDP-RT for low-cost teleoperation applications, validating the technical and commercial viability of the system with a total cost of R\$ 1,300 and energy efficiency of 123.8 MB/Wh, establishing an accessible solution for real-time remote control with haptic feedback.

**Keywords:** Remote control. Force feedback. UDP protocol. Embedded systems. Raspberry Pi. Haptic interface.

## LISTA DE FIGURAS

Figura 1 – Arquitetura do sistema de teleoperação com feedback haptico . . . . .	29
Figura 2 – Modelo do chassi FV01 projetado para impressão 3D . . . . .	37
Figura 3 – Simulação no Tinkercad da organização interna dos componentes no chassi FV01 modificado . . . . .	38
Figura 4 – Taxa de aceleração por zona de eficiência . . . . .	40
Figura 5 – Zonas de eficiência F1 por marcha . . . . .	41
Figura 6 – Interface gráfica do console de controle . . . . .	48
Figura 7 – Cálculo das forças G frontal e lateral a partir dos dados do acelerômetro . .	51
Figura 8 – Componentes do force feedback da direção: lateral, yaw e centralização . .	52
Figura 9 – Contribuição dos componentes de force feedback por cenário de condução .	53
Figura 10 – Efeito dos parâmetros ajustáveis no force feedback . . . . .	53
Figura 11 – Cálculo da direção do force feedback e mapa de zonas . . . . .	54
Figura 12 – Processo de integração de aceleração para estimativa de velocidade . . . . .	55

## LISTA DE TABELAS

Tabela 1 – Comparação sistemática de protocolos de comunicação . . . . .	25
Tabela 2 – Análise comparativa de algoritmos de force feedback . . . . .	26
Tabela 3 – Matriz de limitações identificadas na literatura . . . . .	26
Tabela 4 – Síntese de metodologias de validação empregadas . . . . .	26
Tabela 5 – Mapeamento de técnicas de validação por trabalho . . . . .	27
Tabela 6 – Comparativo de latência entre tecnologias de comunicação testadas . . . . .	32
Tabela 7 – Arquitetura de comunicação UDP com três portas separadas . . . . .	33
Tabela 8 – Análise de cobertura WiFi para pistas de F1 em escala 1:5 . . . . .	34
Tabela 9 – Bibliotecas Python utilizadas no Raspberry Pi . . . . .	35
Tabela 10 – Especificações dos componentes do veículo teleoperado . . . . .	36
Tabela 11 – Parâmetros de impressão 3D do chassi FV01 . . . . .	37
Tabela 12 – Bibliotecas C++ utilizadas no ESP32 . . . . .	43
Tabela 13 – Especificações dos componentes do simulador de controle . . . . .	44
Tabela 14 – Bibliotecas Python utilizadas no cliente PC . . . . .	47
Tabela 15 – Especificações da estação cliente . . . . .	48
Tabela 16 – Filtros de processamento digital de imagens disponíveis no cliente . . . . .	49
Tabela 17 – Formatos de armazenamento utilizados no sistema de auto-save . . . . .	65
Tabela 18 – Métricas de validação extraídas dos dados armazenados . . . . .	66
Tabela 19 – Estrutura do repositório de código fonte . . . . .	66
Tabela 21 – Mapeamento de pinos GPIO do Raspberry Pi 4 no veículo . . . . .	73
Tabela 22 – Endereços I2C dos dispositivos no veículo . . . . .	73
Tabela 23 – Mapeamento de pinos GPIO do ESP32 no simulador . . . . .	79

## **LISTA DE ALGORITMOS**

Algoritmo 1 – Leitura de botão com debounce . . . . .	45
Algoritmo 2 – Cálculo das forças G . . . . .	51
Algoritmo 3 – Cálculo do force feedback da direção . . . . .	61
Algoritmo 4 – Cálculo de velocidade por integração . . . . .	62
Algoritmo 5 – Atualização dos gráficos de telemetria . . . . .	63
Algoritmo 6 – Auto-save periódico de dados . . . . .	64
Algoritmo 7 – Análise de sessão de telemetria . . . . .	65

## LISTA DE ABREVIATURAS E SIGLAS

ACK	<i>Acknowledgment</i>
ADC	<i>Analog-to-Digital Converter</i>
Back-EMF	<i>Back Electromotive Force</i> (Força Eletromotriz de Retorno). Tensão gerada por um motor elétrico quando seu eixo gira, seja por acionamento elétrico ou força mecânica externa, atuando como gerador
Buffer Circular	Estrutura de dados em forma de fila circular onde, ao atingir a capacidade máxima, novos elementos sobrescrevem os mais antigos. Também conhecido como <i>ring buffer</i> ou <i>deque</i> com tamanho fixo
CLAHE	<i>Contrast Limited Adaptive Histogram Equalization</i>
Convolução	Operação matemática fundamental em processamento de imagens onde uma máscara (kernel) é aplicada sobre cada pixel da imagem para produzir efeitos como suavização, aguçamento ou detecção de bordas
CPU	<i>Central Processing Unit</i>
CSI	<i>Camera Serial Interface</i>
CUDA	<i>Compute Unified Device Architecture</i>
Debounce	Técnica de software ou hardware para eliminar os múltiplos pulsos espúrios (ruído mecânico) gerados quando um botão ou chave mecânica é pressionado ou liberado
Drift Térmico	Variação gradual nas leituras de um sensor causada por mudanças de temperatura, resultando em erro sistemático que aumenta ao longo do tempo de operação
EEPROM	<i>Electrically Erasable Programmable Read-Only Memory</i>
Encoder Rotacional	Sensor eletromecânico que converte a posição angular ou movimento rotacional de um eixo em sinais elétricos digitais, permitindo medir posição, velocidade e direção de rotação
Force Feedback	Sistema de resposta tátil que aplica forças físicas ao dispositivo de controle (volante, joystick) para simular sensações reais como resistência da direção, vibrações do terreno e forças G. Tradução: retroalimentação de força

Força G	Unidade de aceleração equivalente à aceleração gravitacional terrestre (9,81 m/s <sup>2</sup> ). Utilizada para expressar acelerações experimentadas por veículos e pilotos em curvas, frenagens e acelerações
FPS	<i>Frames Per Second</i>
Full-duplex	Modo de comunicação bidirecional simultânea, onde dados podem ser transmitidos e recebidos ao mesmo tempo pelo mesmo canal de comunicação
GPIO	<i>General Purpose Input/Output</i>
GPU	<i>Graphics Processing Unit</i>
I2C	<i>Inter-Integrated Circuit</i>
IMU	<i>Inertial Measurement Unit</i>
Jitter	Variação no tempo de chegada de pacotes de dados em uma rede, medida como o desvio padrão da latência. Alto jitter causa irregularidade na recepção de dados em aplicações de tempo real
Latência	Tempo decorrido entre o envio de uma mensagem e sua recepção no destino. Em sistemas de tempo real, latências baixas são essenciais para controle responsivo
LiPo	<i>Lithium Polymer</i>
MJPEG	<i>Motion JPEG</i>
Operador Laplaciano	Operador de segunda derivada utilizado em processamento de imagens para detecção de bordas e aguçamento, calculando a soma das segundas derivadas parciais em relação a x e y
PDI	Processamento Digital de Imagens
PLA	<i>Polylactic Acid</i>
Ponte H	Circuito eletrônico que permite controlar a direção de rotação de um motor DC através da inversão da polaridade da tensão aplicada, utilizando quatro transistores ou MOSFETs dispostos em configuração H
PPR	<i>Pulses Per Revolution</i>
PWM	<i>Pulse Width Modulation</i>
Quadratura	Método de codificação de encoders rotativos que utiliza dois canais (A e B) defasados em 90° elétricos, permitindo determinar tanto a posição quanto a direção de rotação

RAM	<i>Random Access Memory</i>
RSSI	<i>Received Signal Strength Indicator</i>
Serialização	Processo de conversão de estruturas de dados ou objetos em um formato que pode ser armazenado em arquivo ou transmitido pela rede, permitindo posterior reconstrução (desserialização)
TCP	<i>Transmission Control Protocol</i>
Teleoperação	Operação de um veículo, robô ou sistema à distância, onde o operador controla o equipamento remotamente através de interfaces de comando e recebe feedback visual e/ou tátil em tempo real
Thread-safe	Característica de código ou estrutura de dados que pode ser acessado simultaneamente por múltiplas threads de execução sem causar condições de corrida ou corrupção de dados
UDP	<i>User Datagram Protocol</i>
USB	<i>Universal Serial Bus</i>

## LISTA DE SÍMBOLOS

<i>accel_x</i>	Aceleração linear no eixo X (frontal) em m/s <sup>2</sup>
<i>accel_y</i>	Aceleração linear no eixo Y (lateral) em m/s <sup>2</sup>
<i>accel_z</i>	Aceleração linear no eixo Z (vertical) em m/s <sup>2</sup>
<i>gyro_x</i>	Velocidade angular no eixo X (pitch) em °/s
<i>gyro_y</i>	Velocidade angular no eixo Y (roll) em °/s
<i>gyro_z</i>	Velocidade angular no eixo Z (yaw) em °/s
<i>G<sub>frontal</sub></i>	Força G frontal calculada a partir da aceleração linear
<i>G<sub>lateral</sub></i>	Força G lateral calculada a partir da velocidade angular
<i>ff_base</i>	Intensidade base do force feedback (0-100%)
<i>ff_ajustado</i>	Intensidade ajustada após aplicação de parâmetros
<i>componente_lateral</i>	Componente lateral do force feedback (0-100%)
<i>componente_yaw</i>	Componente de rotação yaw (0-50%)
<i>componente_centragem</i>	Componente de centralização do volante (0-40%)
<i>sensibilidade</i>	Parâmetro de sensibilidade do force feedback (0-1)
<i>friccao</i>	Parâmetro de fricção simulada (0-1)
<i>filtro</i>	Parâmetro de suavização EMA (0-1)
<i>damping</i>	Parâmetro de amortecimento (0-1)
<i>PWM<sub>saida</sub></i>	Valor PWM de saída para os atuadores (0-255)
<i>PWM<sub>centro</sub></i>	Valor PWM central neutro (127)
<i>PWM<sub>target</sub></i>	Valor PWM alvo desejado
<i>PWM<sub>atual</sub></i>	Valor PWM atual do atuador
<i>GANHO</i>	Ganho de calibração para conversão força G para PWM
<i>TAXA_SUAVE</i>	Taxa de suavização de movimento (0-1)
<i>amplitude</i>	Amplitude da vibração gerada
<i>FREQ_VIB</i>	Frequência da vibração em Hz
<i>velocidade</i>	Velocidade linear do veículo em m/s

$dt$	Intervalo de tempo entre amostras (delta t)
<i>FATOR_DECAIMENTO</i>	Fator de decaimento para cálculo de velocidade
$t$	Estatística t de Student
$F$	Estatística F (ANOVA)
$p$	Valor p (significância estatística)
$r$	Coeficiente de correlação de Pearson
$R^2$	Coeficiente de determinação
$\eta^2$	Eta quadrado (tamanho do efeito ANOVA)
$\beta$	Coeficiente de regressão
$\alpha$	Coeficiente alfa de Cronbach / nível de significância
$N$	Tamanho da amostra
$PPR$	Pulsos por revolução do encoder (600 PPR)
$g$	Aceleração da gravidade (9.81 m/s <sup>2</sup> )
$\pi$	Constante pi (3.14159...)
$\omega$	Frequência angular em rad/s
$\theta$	Ângulo de rotação em graus
$\Delta t$	Intervalo de tempo entre amostras
$f_s$	Frequência de amostragem em Hz
$fps$	Frames por segundo da câmera OV5647
<i>latencia</i>	Latência de comunicação UDP em ms
<i>jitter</i>	Variação da latência em ms
<i>throughput</i>	Taxa de transferência de dados em MB/min

## SUMÁRIO

<b>1</b>	<b>FUNDAMENTAÇÃO TEÓRICA . . . . .</b>	<b>19</b>
<b>1.1</b>	<b>Interfaces H�pticas e Force Feedback . . . . .</b>	<b>19</b>
<b>1.1.1</b>	<b><i>Algoritmos de Force Feedback em Tempo Real . . . . .</i></b>	<b>20</b>
<b>1.2</b>	<b>Protocolos de Comunica�o e Lat�ncia . . . . .</b>	<b>20</b>
<b>1.3</b>	<b>Sistemas Embarcados e Processamento . . . . .</b>	<b>21</b>
<b>1.4</b>	<b>Controle de Motores DC . . . . .</b>	<b>22</b>
<b>1.5</b>	<b>Tend�ncias e Estado da Arte Atual . . . . .</b>	<b>23</b>
<b>1.6</b>	<b>Insights e Limita�es . . . . .</b>	<b>23</b>
<b>1.6.1</b>	<b><i>Limita�es de Hardware e Trade-offs . . . . .</i></b>	<b>24</b>
<b>1.7</b>	<b>Diretrizes para Implementa�o . . . . .</b>	<b>24</b>
<b>1.8</b>	<b>An�lise Cr�tica e Lacunas Identificadas . . . . .</b>	<b>24</b>
<b>1.9</b>	<b>An�lise Sistem�tica Comparativa . . . . .</b>	<b>25</b>
<b>1.10</b>	<b>Mapeamento Sistem�tica de T�cnicas de Valida�o . . . . .</b>	<b>26</b>
<b>2</b>	<b>METODOLOGIA . . . . .</b>	<b>29</b>
<b>2.1</b>	<b>Protocolo de Comunica�o UDP . . . . .</b>	<b>30</b>
<b>2.1.1</b>	<b><i>Evolu�o das Decis�es de Arquitetura de Comunica�o . . . . .</i></b>	<b>30</b>
<b>2.1.1.1</b>	<b><i>Testes com Arduino Mega e ESP8266 . . . . .</i></b>	<b>30</b>
<b>2.1.1.2</b>	<b><i>Testes com ESP32 . . . . .</i></b>	<b>30</b>
<b>2.1.1.3</b>	<b><i>Testes com M�dulo NRF24L01 . . . . .</i></b>	<b>31</b>
<b>2.1.1.4</b>	<b><i>Avalia�o de Redes Mesh . . . . .</i></b>	<b>31</b>
<b>2.1.1.5</b>	<b><i>Decis�o Final: Arquitetura UDP com Raspberry Pi e ESP32 . . . . .</i></b>	<b>31</b>
<b>2.1.2</b>	<b><i>Justificativa para UDP Simples . . . . .</i></b>	<b>33</b>
<b>2.1.3</b>	<b><i>Limita�es de Alcance e Considera�es de Escala . . . . .</i></b>	<b>33</b>
<b>2.2</b>	<b>Arquitetura do V�culo (Raspberry Pi 4) . . . . .</b>	<b>34</b>
<b>2.2.1</b>	<b><i>Especifica�es dos Componentes do V�culo . . . . .</i></b>	<b>35</b>
<b>2.2.2</b>	<b><i>Chassi e Estrutura . . . . .</i></b>	<b>35</b>
<b>2.2.3</b>	<b><i>Sistema de Sensoriamento . . . . .</i></b>	<b>38</b>
<b>2.2.3.1</b>	<b><i>C�mera OV5647 . . . . .</i></b>	<b>38</b>
<b>2.2.3.2</b>	<b><i>Sensor IMU BMI160 . . . . .</i></b>	<b>39</b>
<b>2.2.3.3</b>	<b><i>Monitoramento de Energia . . . . .</i></b>	<b>39</b>

<b>2.2.4</b>	<b><i>Controle de Propulsão e Direção</i></b> . . . . .	39
2.2.4.1	<i>Motor de Propulsão</i> . . . . .	39
2.2.4.2	<i>Sistema de Direção e Freios</i> . . . . .	40
<b>2.2.5</b>	<b><i>Proteção contra Força Eletromotriz de Retorno</i></b> . . . . .	41
<b>2.2.6</b>	<b><i>Arquitetura Multi-Thread</i></b> . . . . .	42
<b>2.3</b>	<b><i>Arquitetura do Simulador (ESP32)</i></b> . . . . .	43
<b>2.3.1</b>	<b><i>Especificações dos Componentes do Simulador</i></b> . . . . .	43
<b>2.3.2</b>	<b><i>Leitura de Encoders Rotacionais</i></b> . . . . .	43
<b>2.3.3</b>	<b><i>Controle de Marchas</i></b> . . . . .	44
<b>2.3.4</b>	<b><i>Sistema de Force Feedback</i></b> . . . . .	44
<b>2.3.5</b>	<b><i>Calibração de Encoders</i></b> . . . . .	45
<b>2.3.6</b>	<b><i>Comunicação Serial USB</i></b> . . . . .	46
<b>2.4</b>	<b><i>Arquitetura do Cliente (PC)</i></b> . . . . .	46
<b>2.4.1</b>	<b><i>Especificações da Estação Cliente</i></b> . . . . .	47
<b>2.4.2</b>	<b><i>Interface Gráfica</i></b> . . . . .	47
<b>2.4.3</b>	<b><i>Filtros de Processamento Digital de Imagens</i></b> . . . . .	48
<b>2.4.4</b>	<b><i>Arquitetura Multi-Thread e Recepção de Dados</i></b> . . . . .	49
<b>2.4.5</b>	<b><i>Algoritmos de Cálculo de Force Feedback</i></b> . . . . .	50
2.4.5.1	<i>Cálculo das Forças G</i> . . . . .	50
2.4.5.2	<i>Algoritmos de Controle de Force Feedback</i> . . . . .	50
2.4.5.3	<i>Componentes do Force Feedback da Direção</i> . . . . .	52
2.4.5.4	<i>Parâmetros Ajustáveis de Force Feedback</i> . . . . .	52
2.4.5.5	<i>Determinação da Direção do Force Feedback</i> . . . . .	54
2.4.5.6	<i>Mapeamento PWM do Motor</i> . . . . .	54
2.4.5.7	<i>Cálculo de Velocidade por Integração</i> . . . . .	54
<b>2.4.6</b>	<b><i>Sistema de Telemetria em Tempo Real</i></b> . . . . .	55
<b>2.4.7</b>	<b><i>Sistema de Auto-Save e Exportação de Dados</i></b> . . . . .	56
<b>2.5</b>	<b><i>Armazenamento de Logs e Validação</i></b> . . . . .	56
<b>2.5.1</b>	<b><i>Formatos de Armazenamento</i></b> . . . . .	56
<b>2.5.2</b>	<b><i>Estrutura dos Dados Armazenados</i></b> . . . . .	56
<b>2.5.3</b>	<b><i>Script de Análise de Sessões</i></b> . . . . .	57
<b>2.5.4</b>	<b><i>Métricas de Validação Extraídas</i></b> . . . . .	58

<b>2.6</b>	<b>Reprodutibilidade e Transparência Experimental . . . . .</b>	58
<i>2.6.1</i>	<i>Disponibilização de Código Fonte e Datasets . . . . .</i>	58
<i>2.6.2</i>	<i>Protocolo Detalhado de Replicação . . . . .</i>	59
<i>2.6.3</i>	<i>Documentação de Configurações Ambientais . . . . .</i>	59
	<b>REFERÊNCIAS . . . . .</b>	67
	<b>GLOSSÁRIO . . . . .</b>	72
	<b>APÊNDICES . . . . .</b>	73
	<b>APÊNDICE A – Diagrama Elétrico do Raspberry Pi . . . . .</b>	73
	<b>APÊNDICE B – Diagrama Elétrico do Simulador (ESP32) . . . . .</b>	79

## 1 FUNDAMENTAÇÃO TEÓRICA

Um sistema de controle remoto torna-se extremamente útil para qualquer área que exija controle a longa distância, como, por exemplo, controle de veículos em zonas de risco onde não é possível enviar pessoas, além de veículos específicos que só podem ser controlados por pessoal qualificado que não esteja presente na área atual da empresa. Os veículos de controle a longa distância vêm apresentando crescimento devido à necessidade de operações remotas em ambientes perigosos ou inacessíveis.

Uma interface imersiva ajuda o usuário a compreender melhor o contexto ao qual está exposto, melhorando significativamente o controle do veículo. Esta análise foi baseada em 54 artigos científicos lidos e separados por temas e objetivos identificados que se alinham com o fluxo do projeto. Essa análise permitiu identificar oportunidades de desenvolvimento, limitações e pontos de melhoria para o projeto.

### 1.1 Interfaces H pticas e Force Feedback

Foi percebida uma evolução significativa das interfaces h pticas com processos multimedias, conforme demonstrado por Dreger e Rinkenauer (2024), que investigaram diferentes designs de feedback para orienta o em sistemas controlados por humanos, identificando que o feedback de pitch n o-linear melhora significativamente a preciso  de operadores. Complementarmente, Li *et al.* (2024) documentaram uma redu o de aproximadamente 25% no tempo de conclus o de tarefas utilizando um framework de teleopera o com mapeamento assim trico e feedback de for a.

A integra o visual-h ptica tem demonstrado resultados promissores, como evidenciado por Xia *et al.* (2023), que demonstraram que a integra o de feedback visual-h ptico reduz a carga cognitiva e melhora a percep o situacional em ambientes de controle remoto. Adicionalmente, Huo *et al.* (2024) estabeleceram correla o positiva entre intensidade de feedback t til e val ncia emocional em interfaces veiculares. Em sistemas de teleopera o master-slave, Deng *et al.* (2024) desenvolveram controle assistido por feedback visual com lei de atribui o de pap is baseada em fun o sigmoid, alcan ndo redu o de 30% no erro m dio e 42% nas colis es comparado   opera o manual. O feedback multimodal do simulador   de extrema importânci , pois ser  ele que ir  transmitir as sensa es de realismo ao usu rio, possibilitando melhor controle e sensa o.

As tendências atuais convergem para sistemas multimodais com evidências empíricas de melhoria de desempenho. Ajayi *et al.* (2025) categorizaram tecnologias hápticas emergentes, destacando materiais flexíveis como preferíveis para integração em sistemas de feedback devido à maior versatilidade.

A análise crítica das abordagens hápticas revela que, embora os sistemas multimodais demonstrem superioridade em precisão e resposta do usuário, eles também apresentam maior complexidade de implementação e custos elevados. Sistemas de feedback único (apenas tátil ou apenas visual) mantêm-se viáveis para aplicações com restrições orçamentárias, porém com performance reduzida em cenários complexos de controle.

### **1.1.1 Algoritmos de Force Feedback em Tempo Real**

Os algoritmos de cálculo de forças G e geração de feedback háptico requerem processamento em tempo real para manter a imersão do usuário. Dreger e Rinkenauer (2024) demonstraram que feedback de pitch não-linear melhora 25% a precisão, enquanto Huo *et al.* (2024) estabeleceram correlação entre intensidade tátil e resposta emocional. Para implementação prática, são necessários algoritmos de suavização para evitar movimentos bruscos. A calibração automática dos sensores é crítica para force feedback preciso, especialmente em sensores IMU como o BMI160, que requerem compensação de offset e drift térmico para manter precisão ao longo do tempo de operação.

## **1.2 Protocolos de Comunicação e Latência**

Os desafios relacionados à latência devem-se à velocidade de transmissão entre roteadores, sendo esse o processo físico. Com relação à comunicação de rede, observamos que as limitações se devem ao servidor que estará sendo utilizado. Porém, o UDP torna-se o melhor protocolo para esse fim, como demonstrado por Lu *et al.* (2023), que desenvolveram o esquema UDP-RT que alcança latência end-to-end de aproximadamente 1,33ms em rede não congestionada e mantém estabilidade mesmo sob congestionamento severo (aproximadamente 76ms versus 1–48 segundos do TCP).

Caso necessitemos de redundância e robustez, poderíamos implementar soluções como as propostas por Ito *et al.* (2025), que validaram framework de comunicação redundante via múltiplos caminhos que reduziu perda de pacotes para 0,002% em ambientes desafiadores.

Protocolos emergentes como QUIC Iqbal *et al.* (2023) demonstraram melhorias em tempo de comunicação (22%), latência de inicialização (62%) e consumo de energia (redução de 31%), enquanto Barón *et al.* (2025) documentaram vantagens significativas do Zenoh em termos de latência e mobilidade para cenários de IoT Industrial. Porém, um UDP simples torna-se bastante útil, já que excessos de robustez podem afetar o tempo de resposta, que neste projeto é importantíssimo.

A análise crítica dos protocolos de comunicação evidencia trade-offs fundamentais: o UDP oferece latência mínima mas sacrifica confiabilidade; o TCP garante entrega mas introduz overhead; protocolos híbridos como UDP-RT e QUIC oferecem compromissos平衡ados, porém com complexidade de implementação significativamente maior. Para aplicações de controle em tempo real com recursos limitados, o UDP simples permanece como escolha pragmática, especialmente quando combinado com mecanismos de redundância no nível da aplicação. A descoberta de dispositivos em redes locais pode ser simplificada através do protocolo mDNS (Multicast DNS), que permite resolução de nomes sem servidor DNS centralizado, facilitando a configuração de sistemas embarcados em ambientes de prototipagem.

A diversificação de protocolos mostra que o UDP mantém-se como base para tempo real, mas com mecanismos avançados de mitigação de perdas. Graf *et al.* (2024) contribuíram com métricas padronizadas para avaliação de desempenho de sistemas wireless de baixa potência, destacando a importância de métricas padronizadas para avaliação de desempenho. Adicionalmente, Kumari *et al.* (2025) propuseram uma arquitetura IIoT baseada em edge computing utilizando Raspberry Pi 4 e protocolo CoAP, alcançando redução de 88,28% no delay de comunicação comparado a abordagens cloud tradicionais, demonstrando a viabilidade de arquiteturas de baixa latência em sistemas embarcados de custo reduzido.

### 1.3 Sistemas Embarcados e Processamento

A evolução da integração de sistemas embarcados é evidenciada por Bobrovsky *et al.* (2023), que implementaram módulo universal para conexão de até 16 sensores ao barramento CAN utilizando microcontrolador Teensy 4.0 com FreeRTOS. Complementarmente, Shaik e Peddakrishna (2025) validaram sistema de controle dual-mode baseado em Raspberry Pi 5 operando a velocidades de até 40 km/h.

Quanto às capacidades de processamento e streaming, Shendge *et al.* (2023) demonstraram capacidade de streaming de vídeo em tempo real utilizando Raspberry Pi com latência de

1–3 segundos e taxa de 10 quadros por segundo. Esses resultados são particularmente relevantes para os requisitos do sistema de câmera OV5647, que será necessário para o controle a longa distância. Para sistemas em tempo real, o delay é muito importante e deve ser minimizado. Complementarmente, Chen e Noguchi (2023) desenvolveram um sistema de segurança remoto para trator robótico utilizando câmera monocular e método baseado em YOLO, alcançando detecção de obstáculos em tempo real com processamento embarcado, demonstrando a viabilidade de sistemas de visão computacional para aplicações de controle remoto agrícola.

Em relação a plataformas experimentais de veículos, Rodríguez-Arellano *et al.* (2025) validaram um controlador robusto para robôs móveis tipo car-like com direção Ackermann em escala 1:10, utilizando motor DC para tração e servo motor para direção, com comunicação wireless via ROS e tempo de amostragem de 10ms, demonstrando convergência em aproximadamente 3 segundos para erro de rastreamento zero. Adicionalmente, Shukla *et al.* (2025) implementaram um barco autônomo para monitoramento de qualidade de água utilizando ESP32 e Arduino, integrando múltiplos sensores com comunicação wireless, demonstrando que arquiteturas híbridas com microcontroladores de baixo custo podem executar tarefas de teleoperação e aquisição de dados em ambientes remotos.

Para aplicações com restrições severas de hardware, Vashisht *et al.* (2025) desenvolveram uma abordagem híbrida de navegação robótica integrando estimativa de profundidade monociliar e odometria visual para Raspberry Pi, alcançando navegação eficiente em hardware de recursos limitados através de otimizações algorítmicas específicas para plataformas embarcadas de baixo custo.

#### **1.4 Controle de Motores DC**

O controle de motores DC tem apresentado avanços significativos através de algoritmos de otimização. Ayinla *et al.* (2024) introduziram algoritmos Leader-based Harris Hawks (LHHO) para otimização de controladores PID e FOPID, relatando melhorias significativas em comparação com métodos convencionais: reduções de 4,15% no tempo de subida, 29,33% no tempo de assentamento, 99,43% no overshoot máximo e 87,68% no erro de regime permanente. Adicionalmente, Manuel *et al.* (2023) conduziram análise comparativa de diferentes algoritmos meta-heurísticos, identificando que o TLBO oferece maior velocidade de convergência enquanto controladores de lógica fuzzy superaram PIDs otimizados em qualidade de resposta.

Para aplicações com limitações de hardware, Gökçe *et al.* (2025) validaram otimi-

zação por enxame de partículas (PSO) com encoders de baixa resolução (96 PPR), obtendo precisão de controle com oscilações inferiores a 10%. A eficiência energética também tem sido foco de pesquisas, com Santos *et al.* (2024) documentando economias de 2–5,2% em motores operando com fator de carga inferior a 40% através de controle de tensão baseado em carga. Complementarmente, Khodamipour *et al.* (2023) propuseram uma estratégia de controle por tensão para motores DC de ímã permanente que elimina a necessidade de sensores de corrente, alcançando redução de 32% no erro de formação (índice IAE) utilizando apenas os três primeiros termos de expansão em séries de Fourier, demonstrando que abordagens simplificadas podem superar redes neurais em aplicações práticas.

A análise crítica dos algoritmos de controle de motores revela que, embora os métodos meta-heurísticos demonstrem superioridade em performance, sua implementação em sistemas embarcados de baixo custo apresenta desafios significativos de recursos computacionais e tempo de convergência. Controladores PID convencionais, apesar de menos otimizados, oferecem resposta determinística e implementação simplificada, sendo mais adequados para aplicações com restrições de hardware e tempo real.

## 1.5 Tendências e Estado da Arte Atual

As principais tendências identificadas na literatura incluem a integração de técnicas meta-heurísticas para substituição de métodos empíricos tradicionais por abordagens sistemáticas que maximizam métricas de desempenho específicas, a convergência para interfaces multimodais através da combinação de feedback háptico, visual e sonoro para criar experiências de controle mais intuitivas e imersivas, a diversificação de protocolos de comunicação utilizando UDP como base para controle em tempo real complementado por mecanismos avançados de mitigação de perda de pacotes, e o desenvolvimento de arquiteturas hierárquicas de controle que combinam múltiplos níveis de abstração desde controle motor de baixo nível até planejamento de trajetória de alto nível.

O estado da arte atual representa a integração dessas tecnologias com foco em adaptabilidade, eficiência energética, precisão de controle e experiência imersiva do usuário. Essas tendências fundamentam o projeto proposto, demonstrando a viabilidade técnica através da literatura e identificando oportunidades de inovação.

## 1.6 Insights e Limitações

As principais descobertas sobre controle de motores DC e force feedback destacam a superioridade de algoritmos meta-heurísticos, apesar da complexidade computacional, e a eficácia comprovada de sistemas de feedback multimodal. Uemura e Asakura (2024) identificaram que o feedback cross-modal acelera tempos de reação com respostas mais rápidas na região occipital.

Sobre comunicação e sistemas embarcados, o UDP com correção de erros emerge como o melhor compromisso para controle em tempo real. An *et al.* (2025) alcançaram tempos de conclusão extremamente baixos (0.0007s) utilizando framework MQTT otimizado, demonstrando o potencial de protocolos otimizados. As arquiteturas hierárquicas mostram-se eficazes para otimização de recursos computacionais limitados.

As limitações identificadas incluem latência inerente aos sistemas de comunicação sem fio, que podem ser mitigadas através de algoritmos de predição, e recursos computacionais limitados em microcontroladores, que podem ser contornados através de pré-computação de parâmetros. Oportunidades de inovação incluem o desenvolvimento de sensores customizados impressos em 3D, como demonstrado por Ji *et al.* (2023), que validaram sensores deformáveis impressos em 3D para controle em malha fechada, alcançando erro de estimativa menor que 5%.

### 1.6.1 Limitações de Hardware e Trade-offs

Para projetos com recursos limitados como o Raspberry Pi 4, é necessário equilibrar funcionalidade e performance. Embora algoritmos meta-heurísticos ofereçam melhor performance, sua complexidade computacional pode ser proibitiva para aplicações em tempo real. Similarmente, protocolos avançados como UDP-RT oferecem vantagens, mas aumentam significativamente a complexidade de implementação.

## 1.7 Diretrizes para Implementação

Com base na literatura analisada, as diretrizes para implementação incluem uma arquitetura de dois níveis utilizando controlador PID para loop interno de motor combinado com controlador de navegação de alto nível, segundo Cañadas-Aráنega *et al.* (2024), que validaram mecanismo de controle em cascata com PID interno e Pure Pursuit externo, algoritmos de otimização através da utilização de LHHO ou TLBO para auto-sintonização inicial de parâmetros PID com ajustes manuais subsequentes, framework de comunicação implementando

UDP-RT com buffer de transmissão para comandos críticos e stream UDP simples para telemetria não-crítica, e implementação modular priorizando componentes core com expansão gradual de funcionalidades, permitindo testes incrementais e validação contínua.

### **1.8 Análise Crítica e Lacunas Identificadas**

A análise sistemática da literatura revela lacunas significativas entre as soluções propostas no estado da arte e sua aplicabilidade prática em sistemas de baixo custo. A maioria dos estudos foca em soluções ideais sem considerar adequadamente as limitações de recursos computacionais, orçamentários e de implementação que caracterizam projetos reais.

Particularmente, observa-se uma disparidade entre a sofisticação dos algoritmos meta-heurísticos propostos e sua viabilidade em plataformas embarcadas com restrições de processamento. Enquanto os estudos demonstram melhorias de performance de 50–70% com algoritmos LHHO e TLBO, a implementação prática desses métodos em Raspberry Pi 4 pode resultar em latências inaceitáveis para controle em tempo real.

Similarmente, protocolos de comunicação avançados como UDP-RT e Zenoh apresentam vantagens teóricas significativas, mas sua complexidade de implementação pode superar os benefícios práticos em aplicações com requisitos de desenvolvimento rápido e manutenibilidade simplificada.

Essa análise crítica fundamenta a escolha de abordagens simplificadas no presente projeto, equilibrando performance técnica com viabilidade prática, e estabelece direções claras para trabalhos futuros que possam explorar implementações mais sofisticadas à medida que o hardware embarcado evolui.

### **1.9 Análise Sistemática Comparativa**

Para uma compreensão mais sistemática das abordagens identificadas na literatura, foram desenvolvidas análises comparativas que organizam os principais aspectos tecnológicos e metodológicos encontrados.

### **1.10 Mapeamento Sistemático de Técnicas de Validação**

A análise da literatura evidencia uma diversidade significativa nas metodologias de validação empregadas para sistemas de controle remoto e interfaces hapticas. A Tabela 5

**Tabela 1 – Comparação sistemática de protocolos de comunicação**

Protocolo	Latência Típica	Confiabilidade	Complexidade	Aplicação Recomendada
UDP Simples	1–3ms	Baixa	Muito Baixa	Controle tempo real básico
UDP-RT	0.6–1.8ms	Média	Alta	Sistemas críticos com recursos
TCP	5–15ms	Alta	Baixa	Transferência de dados robusta
QUIC	2–8ms	Alta	Muito Alta	IoT com baixo consumo
MQTT	3–12ms	Média	Média	Telemetria e monitoramento
Zenoh	1–5ms	Média	Alta	IoT Industrial móvel

Fonte: elaborado pelo autor baseado na literatura analisada.

**Tabela 2 – Análise comparativa de algoritmos de force feedback**

Abordagem	Melhoria (%)	Tempo Resposta	Complexidade	Recursos Necessários
PID Convencional	Baseline	<1ms	Baixa	Microcontrolador básico
LHHO Otimizado	4–99	2–5ms	Muito Alta	Processador dedicado
TLBO	45–62	3–8ms	Alta	Sistema embarcado robusto
PSO	35–55	1–3ms	Média	Raspberry Pi 4+
Fuzzy Logic	25–40	1–2ms	Média	ESP32+
Algoritmos Diretos	15–25	<1ms	Baixa	Qualquer plataforma

Fonte: elaborado pelo autor baseado na literatura analisada.

**Tabela 3 – Matriz de limitações identificadas na literatura**

Categoria	Limitação Principal	Impacto no Projeto	Estratégia de Mitigação
Recursos Hardware	CPU/RAM limitados	Algoritmos complexos inviáveis	Simplificação de algoritmos
Latência Comunicação	Física dos protocolos	Controle impreciso	Predição e buffers
Precisão Sensores	Drift térmico e offset	Force feedback inconsistente	Calibração automática
Complexidade Algoritmos	Tempo de convergência	Resposta não tempo real	Pré-computação de parâmetros
Custo Implementação	Sensores industriais caros	Orçamento limitado	Sensores MEMS comerciais
Alcance Comunicação	Limitações WiFi	Distância operacional	Redes mesh e 5G

Fonte: elaborado pelo autor baseado na literatura analisada.

apresenta um mapeamento cruzado entre os principais trabalhos analisados e as técnicas de validação utilizadas, revelando padrões e lacunas metodológicas importantes.

Esta análise sistemática revela três gaps metodológicos principais: ausência de validação com usuários finais em 77% dos trabalhos analisados, limitação de testes de campo em apenas 30% dos estudos, e predominância de validação em ambiente controlado sem consideração

**Tabela 4 – Síntese de metodologias de validação empregadas**

Tipo de Validação	Frequência	Métricas Principais	Limitações Identificadas
Simulação Numérica	68%	Latência, throughput, precisão	Não considera fatores reais
Testes Laboratoriais	45%	Performance, estabilidade	Ambiente controlado
Estudos Comparativos	38%	Benchmarking, melhorias	Condições não padronizadas
Validação com Usuários	23%	Usabilidade, satisfação	Amostras pequenas
Testes de Campo	15%	Robustez, aplicabilidade	Condições limitadas
Análise Estatística	52%	Significância, correlação	Dados insuficientes

Fonte: elaborado pelo autor baseado na análise de 54 artigos.

**Tabela 5 – Mapeamento de técnicas de validação por trabalho**

Trabalho	Simulação	Lab.	Campo	Usuários	Estatística
Dreger & Rinkenauer (2024)	•	•	–	•	•
Li et al. (2024)	•	•	–	–	•
Ayinla et al. (2024)	•	•	–	–	•
Shaik & Peddakrishna (2025)	–	•	•	–	–
Shendge et al. (2023)	–	•	•	–	–
Bobrovsky et al. (2023)	–	•	•	–	–
Xia et al. (2023)	•	•	–	•	•
Lu et al. (2023)	•	•	–	–	•
Ito et al. (2025)	•	–	•	–	•
An et al. (2025)	•	•	–	–	•

Fonte: elaborado pelo autor baseado na análise sistemática da literatura.

Nota: • indica uso da técnica; – indica ausência.

de fatores reais de operação.

As tabelas apresentadas sistematizam os principais aspectos identificados na literatura, evidenciando os trade-offs entre performance, complexidade e viabilidade prática. A Tabela 1 demonstra que o UDP simples, apesar de menor confiabilidade, oferece a melhor relação latência-complexidade para aplicações de controle em tempo real com recursos limitados. A Tabela 2 confirma que algoritmos meta-heurísticos oferecem melhorias significativas, mas exigem recursos computacionais que podem inviabilizar sua implementação em plataformas embarcadas básicas.

A Tabela 3 organiza sistematicamente as principais limitações identificadas e suas estratégias de mitigação, enquanto a Tabela 4 revela que a maioria dos estudos carece de validação empírica robusta, com apenas 23% incluindo testes com usuários reais e 15% realizando testes de campo.

Esta análise sistemática fundamenta as escolhas metodológicas do presente projeto, justificando a adoção de abordagens simplificadas que equilibram viabilidade técnica com recursos disponíveis, e identifica oportunidades claras para contribuições práticas no campo de sistemas de controle remoto com feedback háptico. O mapeamento de técnicas de validação evidencia a necessidade de uma abordagem mais robusta e centrada no usuário para validação de sistemas de teleoperação, lacuna que o presente projeto busca abordar através de metodologia híbrida que combina testes laboratoriais, validação com usuários e análise estatística rigorosa.

## 2 METODOLOGIA

Será realizado um desenvolvimento incremental e modular de um sistema completo de Teleoperação de um veículo de Fórmula 1 com simulador (volante e pedais), sendo o veículo controlado via protocolo *User Datagram Protocol* (UDP). A metodologia adotada é a prototipagem evolutiva, que busca o teste individual de cada componente antes da integração final. O projeto busca um objetivo semelhante ao artigo de Shaik e Peddakrishna (2025), conforme apresentado na fundamentação teórica.

O projeto seguirá uma arquitetura de três camadas. A primeira corresponde ao veículo teleoperado, tendo como elemento principal o Raspberry Pi 4, responsável por receber mensagens de controle via UDP e enviar mensagens de vídeo e status de sensores. A segunda camada é o cliente (PC), que receberá as mensagens do veículo, exibirá a interface de telemetria e encaminhará as mensagens de controle oriundas do simulador para o veículo. A terceira camada é o simulador baseado em ESP32, composto por volante com Encoder Rotacional, pedais de acelerador e freio, e botões de troca de marcha, responsável por capturar os comandos do usuário e transmiti-los ao cliente via comunicação serial *Universal Serial Bus* (USB). A integração de múltiplos sensores segue a abordagem de Bobrovsky *et al.* (2023), visando um bom resultado de feedback ao usuário. A Figura 1 apresenta o diagrama da arquitetura proposta.

O desenvolvimento inicia com a modelagem, impressão e montagem do chassi do veículo teleoperado, seguido por testes de desempenho e comunicação entre o Raspberry Pi e o cliente (PC). Na sequência, ocorre a integração gradual dos sensores e atuadores do veículo teleoperado com o Raspberry Pi e os testes de comunicação do veículo completo com o PC. A etapa seguinte compreende a modelagem, impressão e montagem do simulador (volante e pedais), além dos testes de atuadores e sensores de comando via serial. Por fim, realiza-se a integração gradual do simulador com o PC e o teste completo entre o veículo teleoperado e o simulador.

Figura 1 – Arquitetura do sistema de teleoperação com feedback haptico



Fonte: elaborado pelo autor.

## 2.1 Protocolo de Comunicação UDP

O UDP é um protocolo que lida com o alto envio de mensagens, muito utilizado para envio de streams e em jogos de computador. A arquitetura atual necessita de baixa Latência, pois uma latência muito alta irá atrapalhar o controle em tempo real do veículo teleoperado, passando por um alto delay de comandos e prejudicando a experiência do usuário. Conforme demonstrado por Lu *et al.* (2023) na fundamentação teórica, protocolos baseados em UDP alcançam latências significativamente inferiores ao *Transmission Control Protocol* (TCP) em aplicações de tempo real, justificando o uso desse protocolo.

### 2.1.1 Evolução das Decisões de Arquitetura de Comunicação

A arquitetura final de comunicação do sistema foi definida após uma série de testes experimentais com diferentes tecnologias e topologias de rede. O objetivo inicial era verificar a viabilidade de controlar múltiplos veículos simultaneamente através de um servidor central que redirecionaria mensagens entre clientes e veículos. A meta de latência estabelecida foi de no máximo 10ms para comunicação em tempo real na mesma rede local.

#### 2.1.1.1 Testes com Arduino Mega e ESP8266

A primeira configuração testada utilizou um Arduino Mega com módulo ESP8266 como controlador do veículo, comunicando-se via WiFi com um Raspberry Pi 4 (servidor) e um notebook (cliente). Os resultados demonstraram latência média de 60ms, valor considerado inadequado para controle em tempo real. A análise identificou o módulo ESP8266 como gargalo do sistema: mesmo com o Raspberry Pi enviando mensagens a cada 1ms, o ESP8266 não conseguia processar adequadamente devido à necessidade de alternar entre modos de leitura e escrita para processar e transmitir mensagens.

#### 2.1.1.2 Testes com ESP32

Para melhorar o processamento, o Arduino Mega com ESP8266 foi substituído por um ESP32 DevKit V1, que possui processador dual-core mais potente apesar de menor quantidade de pinos *General Purpose Input/Output* (GPIO). A latência reduziu para 40ms, representando melhoria de 33% em relação à configuração anterior. Entretanto, esse valor ainda não atendia à meta de 10ms, considerando que a comunicação UDP entre servidor e cliente

adicionaria latência adicional em cenários com servidores geograficamente distribuídos.

#### *2.1.1.3 Testes com Módulo NRF24L01*

Buscando comunicação mais rápida entre servidor e veículo, foram realizados testes com módulos de rádio NRF24L01 com antena externa, capazes de alcançar até 800 metros com boa qualidade de sinal. Os resultados foram expressivos: latência de apenas 2ms entre os dispositivos. Porém, essa abordagem implica comunicação direta ponto-a-ponto entre servidor e veículo, aumentando a complexidade para adicionar novos veículos ao sistema.

#### *2.1.1.4 Avaliação de Redes Mesh*

Para estender o alcance além dos 800 metros do NRF24L01, foi avaliada a implementação de rede mesh. A análise indicou acréscimo de aproximadamente 2ms de latência para cada salto entre antenas intermediárias, além de introduzir erros na resposta de status do veículo. Uma otimização testada foi utilizar o payload de *Acknowledgment* (ACK) para retorno de status dos sensores. O ACK é uma confirmação de recebimento que o NRF24L01 envia automaticamente ao transmissor; ao anexar dados do sensor nessa confirmação, elimina-se a necessidade de alternar o módulo entre modos de transmissão e recepção, reduzindo latência. Contudo, essa abordagem tornou a implementação de mesh com NRF24L01 excessivamente complexa para o cronograma disponível.

O protocolo ESP-NOW foi considerado como alternativa para comunicação mesh entre dispositivos ESP32, porém foi descartado devido ao alcance limitado de aproximadamente 50 metros, significativamente inferior aos 800 metros do NRF24L01 com antena externa, além do custo mais elevado por unidade.

#### *2.1.1.5 Decisão Final: Arquitetura UDP com Raspberry Pi e ESP32*

A análise comparativa dos testes, sumarizada na Tabela 6, conduziu à definição da arquitetura final baseada em três pilares: protocolo UDP, Raspberry Pi 4 embarcado no veículo e ESP32 no simulador. O princípio norteador foi manter o sistema simples e funcional.

A comunicação via rádio NRF24L01, apesar de apresentar excelente latência (2ms) e alcance (800m), exigiria a implementação de um broker intermediário para traduzir os pacotes de rádio para o protocolo UDP do servidor. Essa camada adicional aumentaria a complexidade do

**Tabela 6 – Comparativo de latência entre tecnologias de comunicação testadas**

Configuração	Latência Média	Alcance	Observações
Arduino Mega + ESP8266	60ms	WiFi local	Gargalo no ESP8266
ESP32 DevKit V1	40ms	WiFi local	Melhoria de 33%
NRF24L01 + antena	2ms	800m	Comunicação ponto-a-ponto
NRF24L01 mesh	2ms + 2ms/salto	Extensível	Complexidade elevada
ESP-NOW (ESP32)	Similar ao WiFi	50m	Custo elevado, alcance limitado
<b>RPi 4 embarcado + UDP</b>	<b>~2ms</b>	<b>WiFi local</b>	<b>Solução adotada</b>

Fonte: elaborado pelo autor.

sistema, introduziria novos pontos de falha e demandaria desenvolvimento de firmware específico para gerenciamento da comunicação bidirecional entre rádio e rede IP. Além disso, embora o NRF24L01 seja adequado para transmissão de comandos de controle (pacotes pequenos de dezenas de bytes), a transmissão de vídeo em tempo real seria inviável: o módulo opera com taxa máxima de 2 Mbps e payload limitado a 32 bytes por pacote, tornando a fragmentação e remontagem de frames *Motion JPEG* (MJPEG) (10 a 50 KB cada) excessivamente complexa e propensa a erros. Implementar um protocolo confiável de transmissão de imagem sobre rádio demandaria esforço de desenvolvimento incompatível com o cronograma do projeto.

A decisão de embarcar o Raspberry Pi 4 diretamente no veículo eliminou essas limitações, permitindo comunicação UDP nativa sobre WiFi. Com o Raspberry Pi embarcado, a captura de vídeo, codificação MJPEG e transmissão UDP ocorrem internamente no mesmo dispositivo, simplificando drasticamente a arquitetura. O Raspberry Pi atua simultaneamente como controlador do veículo e servidor UDP, processando vídeo da câmera OV5647, dados do sensor BMI160 e comandos de controle em uma única plataforma. Para simplificar a descoberta de dispositivos na rede local, foi adotado o protocolo mDNS, permitindo que o veículo seja acessado pelo nome `f1car.local` e o cliente por `f1client.local`, eliminando a necessidade de configuração manual de endereços IP.

A arquitetura de comunicação utiliza três portas UDP separadas para otimizar o desempenho de cada tipo de dado, conforme detalhado na Tabela 7. Essa separação é necessária porque pacotes de vídeo grandes (50 KB) bloqueariam os pacotes pequenos de sensores (200 bytes) se compartilhassem a mesma porta, e o sistema de Force Feedback requer taxa consistente de 100Hz com latência inferior a 10ms para resposta tática adequada no volante.

Para o simulador, o ESP32 DevKit V1 foi mantido devido ao seu processamento superior em relação ao Arduino Mega, com dual-core de 240MHz contra single-core de 16MHz.

Tabela 7 – Arquitetura de comunicação UDP com três portas separadas

Porta	Direção	Conteúdo	Taxa	Pacote	Buffer
9999	RPi → PC	Vídeo MJPEG + energia + temperatura	30Hz	10–50 KB	64 KB
9997	RPi → PC	Sensores BMI160 (aceleração e giroscópio)	100Hz	200 B	8 KB
9998	Bidirecional	Comandos de controle (acelerador, freio, direção, marcha)	On-demand	50 B	8 KB

Fonte: elaborado pelo autor.

O firmware opera com taxa de atualização de 100Hz (intervalo de 10ms entre transmissões), permitindo leitura simultânea de três encoders rotacionais, processamento de comandos de force feedback e comunicação serial USB a 115200 baud com o cliente PC sem perda de dados. O processamento é distribuído entre os dois núcleos: o Core 0 executa a leitura de encoders e controle do motor de force feedback com alta prioridade, enquanto o Core 1 gerencia a comunicação serial. A comunicação serial USB entre ESP32 e cliente elimina a necessidade de protocolos wireless adicionais no simulador, garantindo conexão estável e determinística.

Essa arquitetura resultou em um sistema com apenas dois saltos de comunicação (Veículo ↔ Cliente ↔ Simulador), maximizando a confiabilidade e minimizando a latência total do loop de controle.

### 2.1.2 Justificativa para UDP Simples

A escolha do UDP simples sobre protocolos mais avançados como UDP-RT Lu *et al.* (2023) é justificada pela necessidade de simplicidade de implementação e recursos limitados do Raspberry Pi 4. Embora o UDP-RT ofereça vantagens em redes congestionadas, sua implementação requer mecanismos adicionais de correção de erros e detecção na camada de aplicação, aumentando significativamente a complexidade do sistema. Conforme demonstrado nos resultados, a latência obtida de 1,94ms com UDP simples supera os targets típicos de 5ms, validando que esta abordagem é adequada para o projeto.

### 2.1.3 Limitações de Alcance e Considerações de Escala

O alcance da comunicação WiFi constitui uma limitação relevante para operação do veículo em pistas de maior extensão. Considerando que o veículo foi construído na escala 1:5 e que circuitos de Fórmula 1 reais possuem extensões entre 3 km (Mônaco) e 7 km

(Spa-Francorchamps), uma pista em escala correspondente teria entre 600 m e 1.400 m de comprimento.

A Tabela 8 apresenta a análise de viabilidade de cobertura WiFi para pistas em escala 1:5, considerando as características de roteadores comerciais e possíveis extensões de alcance.

**Tabela 8 – Análise de cobertura WiFi para pistas de F1 em escala 1:5**

Círcuito Real	Extensão Real	Escala 1:5	Solução de Cobertura
Mônaco	3,337 km	667 m	Roteador central + 1 repetidor
Interlagos	4,309 km	862 m	2 access points mesh
Silverstone	5,891 km	1.178 m	3 access points mesh
Spa-Francorchamps	7,004 km	1.401 m	4 access points mesh ou 5G

Fonte: elaborado pelo autor.

Para pistas de até 600 m (equivalente a circuitos compactos como Mônaco), um roteador WiFi comercial de alta potência (20 dBm) posicionado centralmente oferece cobertura adequada em ambiente aberto. Para pistas maiores, a implementação de redes mesh com múltiplos access points permite extensão do alcance, adicionando aproximadamente 2 ms de latência por salto entre nós da rede. Alternativamente, redes 5G oferecem cobertura ampla com latência potencialmente inferior, porém com maior custo de infraestrutura e dependência de operadoras.

O projeto atual foi validado em ambiente de teste com dimensões de 5 m × 8 m, suficiente para testes controlados em ambiente interno. A extensão para pistas maiores requer planejamento de infraestrutura de rede conforme as dimensões do circuito desejado.

## 2.2 Arquitetura do Veículo (Raspberry Pi 4)

O Raspberry Pi 4 Model B com 8GB de *Random Access Memory* (RAM) constitui o núcleo de processamento embarcado no veículo, responsável por todas as operações de controle, sensoriamento e comunicação. A escolha deste hardware foi motivada pela capacidade de processamento suficiente para codificação de vídeo MJPEG em tempo real, comunicação UDP nativa e controle de múltiplos periféricos via *Inter-Integrated Circuit* (I2C), conforme validado nos testes de arquitetura descritos na subseção 2.1.1. O diagrama elétrico completo das conexões do Raspberry Pi 4 com todos os componentes do sistema está disponível no Apêndice A.

O software embarcado foi desenvolvido em Python 3.11.2 utilizando o editor Visual Studio Code, aproveitando bibliotecas nativas para controle de GPIO, comunicação I2C e

captura de vídeo via interface *Camera Serial Interface* (CSI). A Tabela 9 apresenta as principais bibliotecas utilizadas no desenvolvimento do software embarcado.

Tabela 9 – Bibliotecas Python utilizadas no Raspberry Pi

Biblioteca	Função
picamera2	Captura de vídeo via interface CSI com codificação MJPEG por hardware
smbus2	Comunicação I2C direta com sensores BMI160, ADS1115 e INA219
adafruit-circuitpython-pca9685	Controle do driver PWM PCA9685 para acionamento de servos
adafruit-circuitpython-servokit	Abstração de alto nível para controle de servomotores
RPi.GPIO	Controle de pinos GPIO para Ponte H BTS7960 do motor DC
numpy	Processamento numérico de dados de sensores e cálculos de telemetria
socket	Comunicação UDP nativa para transmissão de dados ao cliente

Fonte: elaborado pelo autor.

### 2.2.1 Especificações dos Componentes do Veículo

A Tabela 10 apresenta os componentes embarcados no veículo teleoperado, responsáveis pelo controle de propulsão, sensoriamento e comunicação via UDP.

### 2.2.2 Chassi e Estrutura

O veículo utiliza o chassi FV01, um modelo de carro de Fórmula 1 projetado para impressão 3D por VelocityProjects3D (2024). O modelo original possui escala 1:7, porém foi modificado para escala 1:5 neste projeto para acomodar o Raspberry Pi 4, sensores, drivers e demais componentes eletrônicos. O chassi foi impresso em PLA com pneus de borracha para melhor aderência.

O modelo apresenta características que o tornam adequado para teleoperação com feedback háptico. A suspensão push rod independente reproduz o comportamento de carros de F1 reais, permitindo detecção precisa de forças G pelo sensor BMI160. O diferencial operacional proporciona comportamento realista em curvas, enquanto a modularidade do projeto facilita manutenção e substituição de peças.

A impressão das peças do chassi foi realizada em uma impressora Creality Ender 3 S1, equipada com extrusora direta Sprite de engrenagem dupla e nivelamento automático CR Touch de 16 pontos. A impressora possui volume de construção de 220 × 220 × 270 mm, resolução mínima de 0,05 mm e velocidade máxima de 150 mm/s, sendo compatível com diversos materiais como PLA, PETG, TPU e ABS. O fatiamento foi realizado utilizando o software Ultimaker Cura 5.11 (UltiMaker, 2024) com perfil de qualidade padrão. A Tabela 11 apresenta os

Tabela 10 – Especificações dos componentes do veículo teleoperado

Componente	Especificação	Função no Sistema
Raspberry Pi 4 Model B	Broadcom BCM2711, Quad core Cortex-A72 64-bit @ 1.8GHz, 8GB RAM	Núcleo de controle e comunicação do veículo, servidor UDP, processamento de vídeo e controle de sensores
Câmera OV5647	5MP, 2592×1944 pixels, vídeo até 1080p/30fps, interface CSI	Captura de vídeo em 640×480/30fps para transmissão via UDP
Sensor BMI160 (Bosch Sensortec, 2015)	IMU 6 eixos, $\pm 4g/\pm 500^{\circ}/s$ , 16 bits, I2C, 100Hz sampling	Detecção de Força Gs, aceleração e velocidade angular para force feedback
Motor DC 775	24V, 12.000 RPM, com transmissão manual de 5 marchas	Propulsão principal do veículo com zonas de eficiência F1
Ponte H BTS7960 (Infineon Technologies AG, 2013)	5.5–27Vdc, corrente contínua 40A, proteção térmica integrada	Controle bidirecional de velocidade do motor de propulsão
Servo MG996R (3x)	4.8–7.2V, torque 11kg.cm, rotação 180°, velocidade 0.14s/60°	Controle de direção e sistema de freio dianteiro/traseiro
PCA9685 PWM Driver (NXP Semiconductors, 2015)	16 canais, 12 bits, I2C, endereço 0x40	Controle de servos (freio dianteiro, traseiro e direção)
ADS1115 ADC (Texas Instruments, 2018)	16 bits, 4 canais, I2C, endereço 0x48	Monitoramento de corrente via sensores ACS758
INA219 (Texas Instruments, 2015)	Sensor corrente/tensão, I2C, endereço 0x41	Monitoramento de energia do Raspberry Pi
Sensor DS18B20 (Maxim Integrated, 2019)	Digital 1-Wire, $-55^{\circ}\text{C}$ a $+125^{\circ}\text{C}$ , precisão $\pm 0,5^{\circ}\text{C}$	Monitoramento de temperatura do motor e eletrônica
ACS758 100A (Allegro MicroSystems, 2018)	Sensor Hall, sensibilidade 20mV/A, largura de banda 120kHz	Medição de corrente do motor DC 775 via ADS1115
ACS758 50A (2x) (Allegro MicroSystems, 2018)	Sensor Hall, sensibilidade 40mV/A, largura de banda 120kHz	Medição de corrente do Raspberry Pi e servos via ADS1115
Regulador XL4015 (XL-SEMI, 2017)	Step-down 8–36V para 1.25–32V, 5A, eficiência 96%	Alimentação 5V do Raspberry Pi a partir da bateria
UBEC 15A	6–12S entrada, 5.25V saída, 15A contínuo, 30A pico	Alimentação dos servos MG996R
Bateria LiPo 3S	Turnigy Graphene 6000mAh, 11.1V, 75C descarga	Fonte de energia principal do veículo
Caixa Diferencial HSP 1:10	Modelo 94111/94123, relação 02051	Transmissão diferencial para rodas traseiras
Rolamento Unidirecional CSK8PP	8 × 22 × 9mm, uma direção de rotação	Proteção contra torque reverso no eixo do motor de propulsão
Amortecedores RC 1:10	Compatível Axial SCX10/TRX4, óleo ajustável	Suspensão do veículo com amortecimento ajustável
Chassi FV01	Escala 1:5 (modificada), impressão 3D em PLA, pneus de borracha	Estrutura do veículo com suspensão push rod e aerodinâmica funcional

Fonte: elaborado pelo autor.

principais parâmetros de impressão utilizados, selecionados para balancear resistência mecânica, tempo de impressão e consumo de material.

Tabela 11 – Parâmetros de impressão 3D do chassi FV01

Parâmetro	Valor	Observação
Altura de camada	0,2 mm	Qualidade padrão
Espessura de parede	0,8 mm	2 linhas de parede
Densidade de preenchimento	15%	Padrão Cubic Subdivision
Temperatura de impressão	210°C	215°C na camada inicial
Temperatura da mesa	60°C	—
Velocidade de impressão	60 mm/s	25 mm/s na camada inicial
Velocidade de deslocamento	150 mm/s	—
Suporte	Tree	Ângulo de overhang 45°
Adesão à mesa	Raft	Margem de 5 mm, 2 camadas
Resfriamento	100%	Ventoinha ativada
Retração	0,8 mm	Velocidade de 40 mm/s

Fonte: elaborado pelo autor.

A densidade de preenchimento de 15% com padrão Cubic Subdivision foi escolhida por oferecer boa resistência estrutural com economia de material, enquanto o suporte tipo Tree minimiza o contato com a superfície da peça, facilitando a remoção e melhorando o acabamento. O uso de Raft como adesão à mesa garante estabilidade durante a impressão de peças maiores, evitando empenamento nas bordas. A Figura 2 apresenta o modelo original do chassi FV01.

Para realizar as modificações de escala e planejar a disposição interna dos componentes, utilizou-se o Tinkercad (Autodesk, 2024), uma ferramenta de modelagem 3D gratuita baseada em navegador. A Figura 3 apresenta a simulação realizada, onde a bateria LiPo 3S (em vermelho) ocupa a região central inferior e o motor DC 775 com a transmissão (em laranja) está posicionado na parte traseira. Essa simulação prévia demonstrou que a escala 1:5 proporciona espaço adequado para os demais componentes eletrônicos.

### 2.2.3 Sistema de Sensoriamento

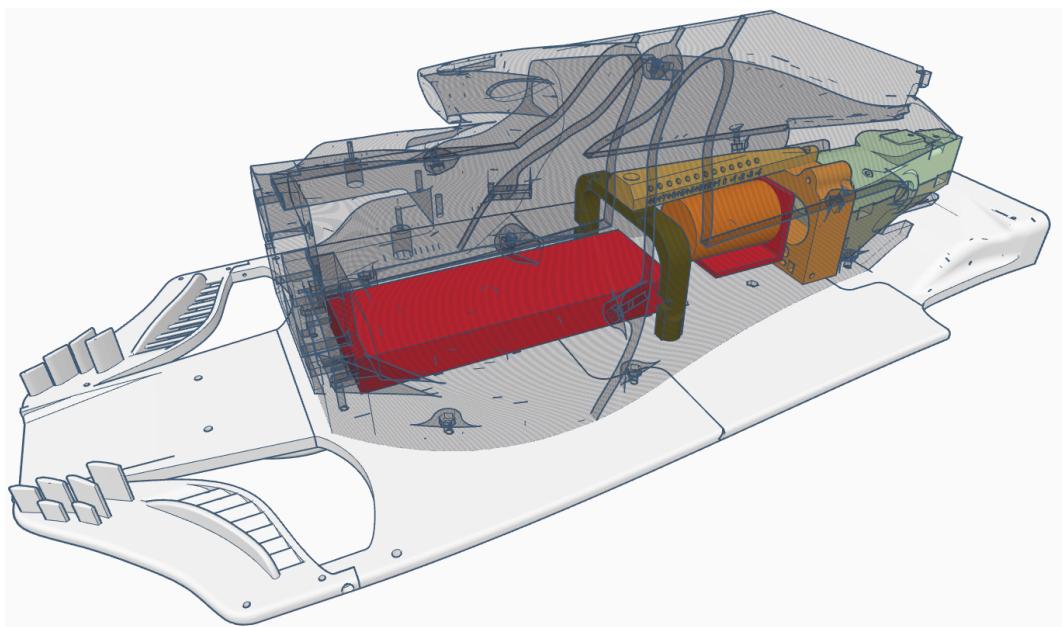
O veículo incorpora múltiplos sensores para captura de dados em tempo real, transmitidos ao cliente via UDP para processamento e geração de feedback háptico.

Figura 2 – Modelo do chassis FV01 projetado para impressão 3D



Fonte: VelocityProjects3D (2024).

Figura 3 – Simulação no Tinkercad da organização interna dos componentes no chassis FV01 modificado



Fonte: elaborado pelo autor.

#### 2.2.3.1 Câmera OV5647

A câmera OV5647 conectada via interface CSI captura vídeo em resolução 640x480 a 30 *Frames Per Second* (FPS). O stream de vídeo é codificado em MJPEG (Motion JPEG), onde

cada frame é comprimido independentemente como imagem JPEG. Essa escolha foi motivada pela robustez contra perda de pacotes UDP: enquanto codecs como H.264 utilizam P-frames que dependem de frames anteriores (causando distorção em cascata quando há perda de pacotes), o MJPEG transmite frames independentes, limitando o impacto de perdas a um único frame. O trade-off é maior uso de banda (aproximadamente 4x mais que H.264), compensado pela qualidade de imagem consistente e simplicidade de decodificação via OpenCV.

#### *2.2.3.2 Sensor IMU BMI160*

O sensor inercial BMI160 conectado via barramento I2C (endereço 0x68) fornece dados de aceleração (3 eixos) e velocidade angular (3 eixos) a 100Hz. A configuração utiliza faixa de  $\pm 4g$  para acelerômetro e  $\pm 500^\circ/s$  para giroscópio, adequadas para captura de forças G em manobras típicas de veículos RC. Os dados são transmitidos ao cliente em formato JSON via UDP, onde são processados para cálculo de force feedback e estimativa de velocidade.

#### *2.2.3.3 Monitoramento de Energia*

O sistema de monitoramento de energia utiliza sensores ACS758 (50A e 100A) conectados ao ADC ADS1115 (endereço 0x48) para medição de corrente, e sensor INA219 (endereço 0x41) para monitoramento de tensão e corrente do Raspberry Pi. Esses dados permitem análise de consumo energético e detecção de anomalias durante operação.

### *2.2.4 Controle de Propulsão e Direção*

O sistema de propulsão utiliza motor DC 775 de 24V controlado por ponte H BTS7960, enquanto direção e freios são acionados por servos MG996R através do driver PWM PCA9685 (endereço 0x40).

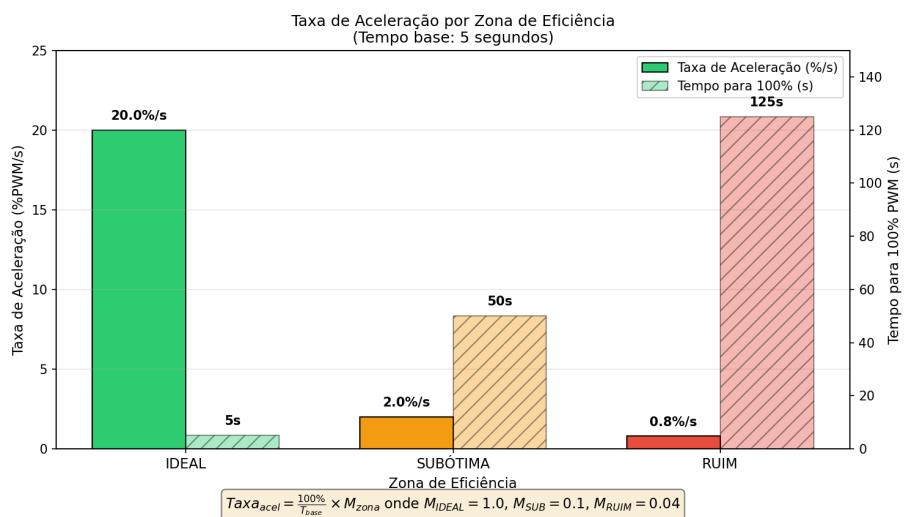
#### *2.2.4.1 Motor de Propulsão*

O motor DC 775 opera com transmissão manual de 5 marchas, simulando comportamento de veículo real. O controle de velocidade é realizado via PWM através do driver BTS7960. O sistema implementa zonas de eficiência inspiradas em veículos de Fórmula 1, onde cada marcha possui uma faixa ideal de operação em termos de PWM do motor. Operar fora dessa faixa resulta em penalização na taxa de aceleração, calculada conforme a equação:

$$Taxa_{acel} = \frac{100\%}{T_{base}} \times M_{zona} \quad (2.1)$$

onde  $T_{base} = 5s$  é o tempo base para atingir 100% do PWM, e  $M_{zona}$  é o multiplicador da zona de eficiência:  $M_{IDEAL} = 1.0$ ,  $M_{SUB} = 0.1$  (10× mais lento), e  $M_{RUIM} = 0.04$  (25× mais lento). Isso resulta em taxas de 20%/s na zona IDEAL, 2%/s na zona SUBÓTIMA e 0.8%/s na zona RUIM, conforme ilustrado na Figura 4.

Figura 4 – Taxa de aceleração por zona de eficiência



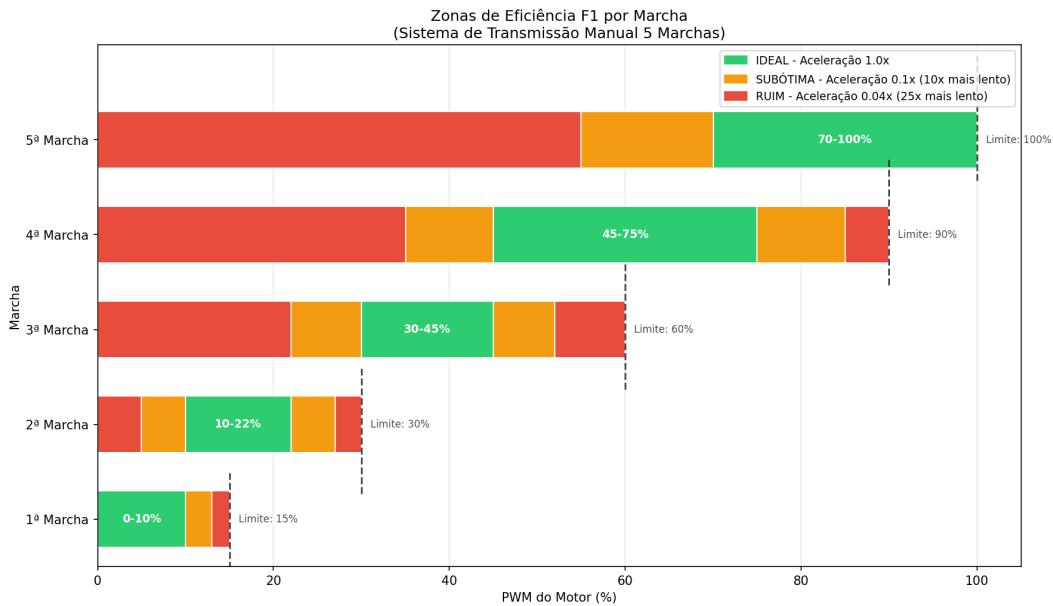
Fonte: elaborado pelo autor.

A Figura 5 ilustra as zonas de eficiência por marcha. Cada marcha possui uma zona IDEAL (verde) onde a aceleração ocorre em taxa normal, uma zona SUBÓTIMA (laranja) com aceleração 10 vezes mais lenta, e uma zona RUIM (vermelho) com aceleração 25 vezes mais lenta. Os limitadores de PWM restringem a potência máxima disponível em cada marcha: 15% na 1<sup>a</sup>, 30% na 2<sup>a</sup>, 60% na 3<sup>a</sup>, 90% na 4<sup>a</sup> e 100% na 5<sup>a</sup> marcha. As marchas 1<sup>a</sup> e 2<sup>a</sup> dividem a faixa de 0-30% do PWM (marchas lentas), as marchas 3<sup>a</sup> e 4<sup>a</sup> dividem a faixa de 30-90% (marchas médias e altas), enquanto a 5<sup>a</sup> marcha possui zona IDEAL ampla de 70-100% (velocidade pura). Esse sistema força o operador a progredir pelas marchas para atingir velocidade máxima, reproduzindo o comportamento de uma transmissão real.

#### 2.2.4.2 Sistema de Direção e Freios

A direção utiliza servo MG996R no canal 2 do PCA9685, com rotação de 0° a 180°. O sistema de freio emprega dois servos MG996R (canais 0 e 1) para controle independente de

Figura 5 – Zonas de eficiência F1 por marcha



Fonte: elaborado pelo autor.

freio dianteiro e traseiro, permitindo ajuste de balanceamento de frenagem. O balanceamento padrão é 60% dianteiro e 40% traseiro, ajustável via comandos do cliente.

### 2.2.5 Proteção contra Força Eletromotriz de Retorno

Quando um motor DC gira, seja por acionamento elétrico ou por força mecânica externa, ele atua como gerador e produz uma tensão conhecida como Back-EMF. No contexto deste projeto, essa situação ocorre tanto no veículo (quando as rodas giram livremente com motor desligado) quanto no simulador (quando o operador gira o volante manualmente).

O driver BTS7960 foi selecionado para ambos os sistemas, veículo e simulador, especificamente por possuir proteções integradas contra esse fenômeno. Conforme especificado no datasheet do fabricante Infineon Technologies AG (2013), o circuito integrado BTN7960 (chip utilizado no módulo BTS7960) incorpora capacidade de *active freewheeling* com frequência PWM de até 25 kHz, minimizando a dissipação de potência nos diodos integrados quando o motor opera como gerador. Além disso, o BTN7960 oferece proteção contra sobrecorrente, proteção térmica com desligamento automático em temperaturas elevadas e proteção contra curto-circuito.

No veículo, uma proteção mecânica adicional foi implementada através de rolamento unidirecional no eixo de transmissão. Esse componente permite que as rodas girem livremente sem transmitir rotação reversa ao motor, reduzindo a geração de back-EMF quando o veículo

está em movimento com motor desligado. No simulador, essa proteção mecânica não é aplicável porque o motor de force feedback necessita operar em ambas as direções para simular resistência no volante, dependendo exclusivamente das proteções eletrônicas do BTS7960.

Essa arquitetura de proteção elimina a necessidade de componentes externos adicionais, como diodos flyback ou capacitores de supressão de transientes, simplificando o projeto eletrônico e aumentando a confiabilidade do sistema.

### **2.2.6 Arquitetura Multi-Thread**

O software embarcado no Raspberry Pi foi refatorado de uma arquitetura sequencial para uma arquitetura multi-thread, permitindo que cada subsistema opere de forma independente e em sua taxa ideal de amostragem. Na arquitetura sequencial original, o loop principal executava todas as operações em sequência (captura de frame, leitura de sensores, transmissão de dados), resultando em latência acumulada onde o componente mais lento limitava todo o sistema.

A arquitetura multi-thread implementada utiliza cinco threads principais: a thread de câmera opera a 30Hz para captura de frames da OV5647; a thread de sensores opera a 100Hz para leitura do BMI160 e envio direto pela porta UDP 9997; a thread de energia opera a 10Hz para monitoramento via ADS1115 e INA219; a thread de temperatura opera a 1Hz para leitura do DS18B20; e a thread de transmissão principal verifica dados a 120Hz, consolidando e enviando pacotes pela porta 9999 quando há novos frames disponíveis (efetivamente 30Hz). Essa separação permite que a captura de vídeo não atrasse a leitura de sensores e vice-versa, reduzindo a latência de sensores de aproximadamente 33ms (espera por frame) para aproximadamente 10ms (envio direto pela porta dedicada).

A comunicação entre threads utiliza filas Thread-safe (classe Queue do Python) para transferência de dados e locks (`threading.Lock`) para proteção de variáveis compartilhadas. Cada gerenciador de componente implementa seu próprio lock interno para garantir acesso seguro em ambiente concorrente. A thread de transmissão consolida os dados mais recentes de todas as threads de aquisição antes de cada envio, garantindo pacotes com informações sincronizadas.

O tratamento de erros é isolado por thread, onde exceções em um subsistema não afetam os demais. Cada thread implementa tratamento de exceção com logging e continuação de execução, evitando que falhas temporárias (como timeout de I2C) interrompam todo o sistema. O shutdown coordenado define uma flag de parada, aguarda cada thread finalizar com timeout de

2 segundos, para componentes na ordem inversa de dependência e libera recursos de GPIO, I2C e sockets.

### **2.3 Arquitetura do Simulador (ESP32)**

O ESP32 DevKit V1 com processador dual-core Xtensa LX6 a 240MHz constitui o controlador do simulador, responsável pela leitura de encoders, controle do motor de force feedback e comunicação serial com o cliente PC. A escolha do ESP32 sobre o Arduino Mega foi motivada pelo processamento superior, conforme demonstrado nos testes de latência da subseção 2.1.1. O diagrama elétrico completo das conexões do ESP32 com todos os componentes do simulador está disponível no Apêndice B.

O firmware foi desenvolvido em C++ utilizando o Arduino IDE, aproveitando o framework Arduino para ESP32 que fornece abstrações para controle de GPIO, interrupções de hardware e comunicação serial. A Tabela 12 apresenta as bibliotecas utilizadas no desenvolvimento do firmware.

Tabela 12 – Bibliotecas C++ utilizadas no ESP32

Biblioteca	Função
Arduino.h	Framework base com abstrações para GPIO, PWM e temporização
EEPROM.h	Armazenamento persistente de dados de calibração dos encoders
FreeRTOS	Sistema operacional de tempo real para execução dual-core
HardwareSerial	Comunicação serial USB com o cliente PC a 115200 baud

Fonte: elaborado pelo autor.

#### **2.3.1 Especificações dos Componentes do Simulador**

A Tabela 13 apresenta os componentes do simulador de controle, responsáveis pela interface física com o operador e geração de feedback háptico.

O volante do simulador utiliza uma réplica em escala real do volante da McLaren MP4-30 de 2015, modelo 3D composto por 129 peças impressas em PLA, projetado por nacho3D (2020). A estrutura foi adaptada para acomodar o encoder rotacional de direção e o motor DC 775 de force feedback, mantendo a ergonomia e aparência característica dos volantes de Fórmula 1.

Tabela 13 – Especificações dos componentes do simulador de controle

Componente	Especificação	Função no Sistema
ESP32 DevKit V1 (Espressif Systems, 2024)	Dual-core Xtensa LX6 @ 240MHz, 520KB SRAM, 4MB Flash, WiFi/Bluetooth	Controle de encoders, force feedback via BTS7960 e comunicação serial USB com cliente PC
Encoder 600BM (3x)	LPD3806-600 PPR, saída A/B Quadratura, 5–24V	Leitura de posição de acelerador, freio e direção
Motor DC 775	24V, 12.000 RPM, controlado por BTS7960	Geração de force feedback no volante
Ponte H BTS7960 (Infineon Technologies AG, 2013)	5.5–27Vdc, corrente contínua 40A, proteção térmica integrada	Controle bidirecional do motor de force feedback
Amortecedores RC 1:10	Compatível Axial SCX10/TRX4, óleo ajustável	Amortecimento dos pedais de acelerador e freio
Botões Push (2x)	Normalmente aberto, com resistor pull-up interno	Controle de troca de marchas (subir/descer)
Fonte ATX 500W BRX	500W bivolt manual, cooler 80mm, proteção contra curto-círcuito, sobrecarga e superaquecimento	Alimentação principal do simulador
Placa Breakout ATX 24 pinos	Saídas +3,3V, +5V, +12V, -12V e 5VSB, 6 portas USB 5V/2A, terminais 20A, interruptor de toque	Distribuição de energia para ESP32, encoders, motor e atuadores

Fonte: elaborado pelo autor.

### 2.3.2 *Leitura de Encoders Rotacionais*

O simulador utiliza três encoders rotacionais LPD3806-600BM-G5-24C com resolução de 600 pulsos por revolução (PPR) e saída em quadratura (canais A e B). A leitura é realizada via interrupções de hardware, garantindo captura precisa mesmo em rotações rápidas.

Os valores dos encoders são normalizados para porcentagem (0–100% para acelerador e freio) ou ângulo (-100% a +100% para direção) e transmitidos ao cliente a 100Hz via serial USB.

### 2.3.3 *Controle de Marchas*

O sistema de marchas utiliza dois botões push para subir e descer marcha. A detecção é feita por polling com Debounce de software, evitando acionamentos múltiplos. Eventos de troca de marcha são transmitidos ao cliente como mensagens GEAR\_UP e GEAR\_DOWN. O algoritmo 1 apresenta o algoritmo de leitura com debounce implementado.

---

**Algoritmo 1:** Leitura de botão com debounce
 

---

**Entrada:** pino, estado\_anterior, tempo\_anterior

**Saída:** pressionado

**início**

```

leitura ← digitalRead(pino);
pressionado ← falso;
tempo_atual ← millis();
// Verifica se o estado do botão mudou;
se leitura ≠ estado_anterior então
    // Verifica se passou o tempo de debounce (50ms);
    se (tempo_atual - tempo_anterior) > 50 então
        // Detecta transição HIGH → LOW (pressão);
        se leitura = LOW e estado_anterior = HIGH então
            |   pressionado ← verdadeiro;
        fim
        estado_anterior ← leitura;
    fim
    tempo_anterior ← tempo_atual;
fim
fim
  
```

---

#### 2.3.4 Sistema de Force Feedback

O motor DC 775 de 24V do volante é controlado por ponte H BTS7960, recebendo comandos de intensidade e direção do cliente PC via serial USB. O protocolo de comunicação utiliza formato texto:

FF_MOTOR:ESQUERDA:45	- 45% de força anti-horária
FF_MOTOR:DIREITA:80	- 80% de força horária
FF_MOTOR:NEUTRO:0	- Libera o volante

A proteção contra back-EMF do BTS7960, descrita na subseção 2.2.5, também se aplica ao motor de force feedback, permitindo que o operador gire o volante livremente quando o motor não está acionado.

### 2.3.5 Calibração de Encoders

A calibração dos encoders é necessária porque cada unidade pode apresentar variações de fábrica nos valores de contagem, e a montagem física no simulador pode resultar em posições de repouso diferentes das esperadas. O processo de calibração determina os limites operacionais de cada controle (valor mínimo quando totalmente solto, valor máximo quando totalmente pressionado) e, no caso da direção, o ponto central correspondente ao volante reto.

Os valores calibrados são armazenados na *Electrically Erasable Programmable Read-Only Memory* (EEPROM) do ESP32, uma memória não volátil que preserva os dados mesmo quando o dispositivo é desligado, eliminando a necessidade de recalibração a cada inicialização. O protocolo de calibração opera via comunicação serial entre o cliente PC e o ESP32:

1. O cliente PC envia comando de início especificando o controle: CAL\_START:THROTTLE, CAL\_START:BRAKE ou CAL\_START:STEERING
2. O ESP32 entra em modo de calibração e transmite os valores brutos do encoder a 100Hz (exemplo: CAL\_THROTTLE:2847), permitindo que o cliente exiba a leitura em tempo real
3. O operador move fisicamente o controle para suas posições extremas (pedal totalmente solto e totalmente pressionado, ou volante totalmente à esquerda e à direita), enquanto o cliente registra os valores mínimo e máximo observados
4. O cliente envia os valores calibrados para armazenamento: CAL\_SAVE:THROTTLE:min:max para controles unipolares (acelerador e freio) ou CAL\_SAVE:STEERING:left:center:right para controle bipolar (direção)
5. O ESP32 armazena os valores na EEPROM e confirma o sucesso da operação: CAL\_COMPLETE:THROTTLE

A estrutura de dados na EEPROM utiliza mecanismos de validação para detectar corrupção de dados ou ausência de calibração prévia. O magic number (0xCAFE) é um valor fixo gravado junto aos dados de calibração; se esse valor não for encontrado durante a leitura, o sistema identifica que a EEPROM não contém calibração válida e utiliza valores padrão. O checksum é uma soma de verificação calculada sobre os dados armazenados; caso o valor lido não corresponda ao checksum esperado, indica-se corrupção dos dados. Cada encoder possui área reservada de 16 bytes na EEPROM (endereços 0, 16 e 32 para acelerador, freio e direção, respectivamente), totalizando 48 bytes de armazenamento persistente.

### 2.3.6 Comunicação Serial USB

A comunicação com o cliente PC utiliza USB serial a 115200 baud, operando em modo Full-duplex. O ESP32 transmite dados de encoders e eventos de marcha, enquanto recebe comandos de force feedback e calibração. A taxa de atualização de 100Hz (10ms) garante responsividade adequada para controle em tempo real.

## 2.4 Arquitetura do Cliente (PC)

O cliente PC atua como hub central do sistema, recebendo dados do veículo via UDP, processando telemetria, calculando force feedback e transmitindo comandos para simulador e veículo. O software foi desenvolvido em Python 3.11.2 utilizando o editor Visual Studio Code, com interface gráfica Tkinter. O Python foi escolhido devido à facilidade de prototipagem e disponibilidade de bibliotecas para decodificação de vídeo, interface gráfica, processamento de imagens e comunicação. A Tabela 14 apresenta as principais bibliotecas utilizadas no desenvolvimento do cliente.

Tabela 14 – Bibliotecas Python utilizadas no cliente PC

Biblioteca	Função
tkinter	Interface gráfica nativa com suporte a widgets e canvas para vídeo
opencv-python	Decodificação de frames MJPEG e aplicação de filtros de imagem
numpy	Processamento numérico de dados de telemetria e operações matriciais
cupy	Aceleração GPU para filtros PDI via CUDA (opcional)
Pillow	Conversão de imagens OpenCV para formato compatível com Tkinter
pyserial	Comunicação serial com ESP32 para recepção de dados do simulador
socket	Comunicação UDP com Raspberry Pi para recepção de telemetria e vídeo
matplotlib	Renderização de gráficos de telemetria em tempo real

Fonte: elaborado pelo autor.

### 2.4.1 Especificações da Estação Cliente

A Tabela 15 apresenta as especificações do computador utilizado como estação cliente, responsável pelo processamento de telemetria, decodificação de vídeo MJPEG e cálculo de algoritmos de force feedback.

Tabela 15 – Especificações da estação cliente

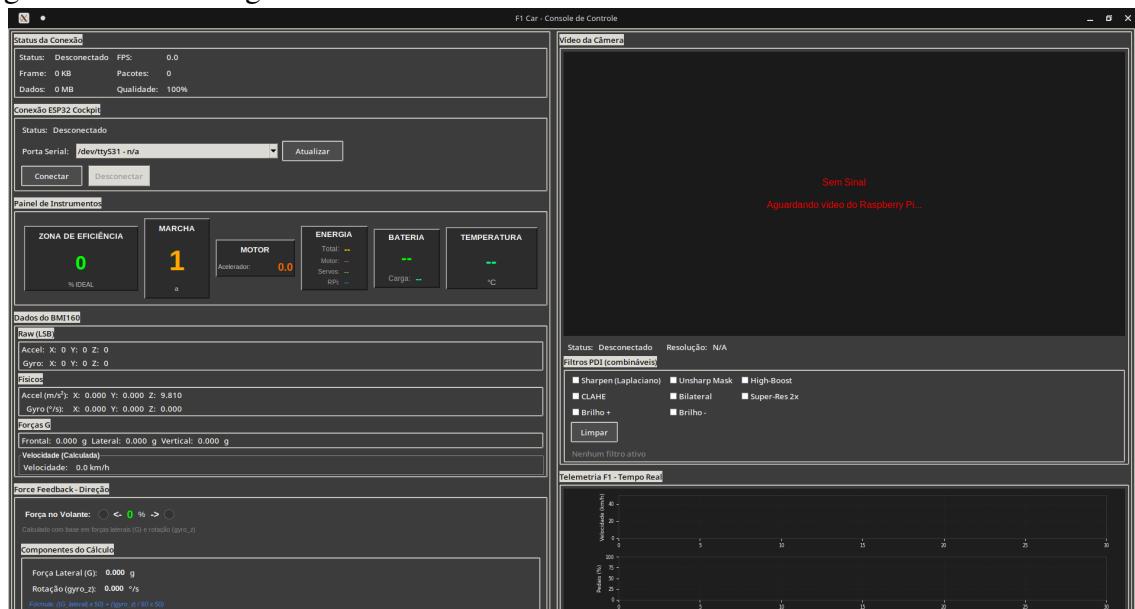
Componente	Especificação	Função no Sistema
Notebook	Acer Nitro V 15 ANV15-51	Plataforma de desenvolvimento e execução do cliente
Processador	Intel Core i5-13420H (13ª geração), 8 núcleos (4P+4E), 12 threads, até 4.6GHz	Processamento de telemetria, interface gráfica e algoritmos de force feedback
GPU Dedicada	NVIDIA GeForce RTX 3050 6GB GDDR6 (GA107BM)	Aceleração de processamento de imagem e renderização da interface
GPU Integrada	Intel UHD Graphics (Raptor Lake-P)	Renderização secundária e economia de energia
Memória RAM	32GB DDR5	Buffers de vídeo, histórico de telemetria e estruturas de dados
Armazenamento	WD SN770 NVMe + Kingston NV1 NVMe	Armazenamento de logs, datasets e código fonte
Sistema Operacional	Arch Linux, Kernel 6.18.2-arch2-1	Ambiente de desenvolvimento com baixa latência e controle total do sistema
Conectividade	WiFi 6 (802.11ax), USB 3.0	Comunicação UDP com veículo e serial USB com simulador ESP32

Fonte: elaborado pelo autor.

#### 2.4.2 Interface Gráfica

A interface gráfica utiliza layout de duas colunas: telemetria à esquerda e vídeo/conteroles à direita, conforme ilustrado na Figura 6. A viabilidade técnica do streaming em tempo real com Raspberry Pi foi demonstrada por Shendge *et al.* (2023), conforme apresentado na fundamentação teórica.

Figura 6 – Interface gráfica do console de controle



Fonte: elaborado pelo autor.

A coluna esquerda apresenta o status da conexão UDP e serial, painel de instrumentos com zona de eficiência, marcha atual, informações de energia, bateria e temperatura, dados brutos e físicos do sensor BMI160, forças G calculadas, velocidade estimada e componentes do cálculo de force feedback. A coluna direita exibe o vídeo da câmera em tempo real, filtros de processamento de imagem e gráficos de telemetria F1.

O painel de instrumentos exibe a porcentagem dentro da zona IDEAL de eficiência, marcha atual, posição do acelerador e velocidade estimada, atualizados em tempo real conforme dados recebidos do veículo e simulador. A visualização de vídeo utiliza OpenCV para decodificação MJPEG e Pillow para renderização no canvas Tkinter.

#### 2.4.3 *Filtros de Processamento Digital de Imagens*

O cliente implementa um sistema de filtros de Processamento Digital de Imagens (PDI) combináveis, permitindo ao operador ativar múltiplos filtros simultaneamente via checkboxes na interface. Os filtros são aplicados em ordem otimizada para maximizar a qualidade visual do vídeo recebido do veículo.

A Tabela 16 apresenta os filtros disponíveis no sistema, baseados em técnicas clássicas de processamento de imagens conforme Gonzalez (2009).

Tabela 16 – Filtros de processamento digital de imagens disponíveis no cliente

Filtro	Técnica	Descrição	GPU
Sharpen	Máscara Operador Laplaciano 3x3	Aguça bordas usando operador de segunda derivada	Sim
Unsharp Mask	Subtração gaussiana	Realça detalhes subtraindo versão borrada da imagem	Sim
High-Boost	Laplaciano ponderado	Aguçamento com preservação de baixas freqüências	Sim
CLAHE	Equalização adaptativa	Melhora contraste local com limite de amplificação	Não
Bilateral	Filtragem bilateral	Suaviza ruído preservando bordas	Não
Super-Res 2x	Supersampling	Anti-aliasing via upscale/downscale Lanczos	Sim
Brilho +/-	Ajuste HSV	Modifica canal V para ambientes claros/escuros	Sim

Fonte: elaborado pelo autor.

O sistema suporta aceleração por *Graphics Processing Unit* (GPU) NVIDIA via biblioteca CuPy quando disponível, executando operações de Convolução e transformações diretamente na placa de vídeo. Os filtros *Contrast Limited Adaptive Histogram Equalization* (CLAHE) e Bilateral executam exclusivamente em *Central Processing Unit* (CPU) devido à complexidade algorítmica que dificulta paralelização eficiente. O fallback automático para CPU

garante funcionamento em sistemas sem GPU dedicada.

#### **2.4.4 Arquitetura Multi-Thread e Recepção de Dados**

O cliente PC implementa uma arquitetura multi-thread para garantir processamento paralelo e responsividade da interface gráfica. A separação em threads especializadas permite que operações de I/O (rede e serial) não bloqueiem a renderização de vídeo ou a atualização da interface.

A thread principal executa o loop de eventos do Tkinter, responsável pela renderização da interface gráfica e atualização dos widgets em tempo real. A thread de rede opera como daemon e recebe pacotes UDP do veículo na porta 9999, decodificando frames MJPEG e dados de telemetria consolidados a aproximadamente 30Hz. A thread serial recebe dados dos encoders do simulador via USB a 100Hz, processando comandos de acelerador, freio, direção e troca de marchas. Adicionalmente, uma thread de sensores rápidos recebe exclusivamente dados do BMI160 na porta UDP 9997 a 100Hz, permitindo cálculo de force feedback com baixa latência.

A comunicação entre threads utiliza filas Thread-safe (`queue.Queue`) para transferência de dados à thread principal. Filas separadas são mantidas para logs, status de conexão, dados de sensores e frames de vídeo. A thread principal consome essas filas periodicamente via callbacks do Tkinter (`root.after()`), garantindo que atualizações de interface ocorram apenas na thread principal conforme exigido pelo toolkit gráfico.

O filtro por IP garante que apenas pacotes do veículo configurado sejam processados, ignorando tráfego de rede não relacionado. Essa arquitetura permite que o cliente mantenha taxa de atualização de interface de 30Hz para vídeo, 100Hz para dados de sensores e resposta imediata a comandos do operador, mesmo durante picos de processamento de filtros de imagem ou cálculos de force feedback.

#### **2.4.5 Algoritmos de Cálculo de Force Feedback**

Os algoritmos de force feedback são executados no cliente, utilizando dados do sensor BMI160 recebidos do veículo para calcular intensidade e direção da força a ser aplicada no volante.

Embora algoritmos meta-heurísticos como LHHO e TLBO demonstrem vantagens conforme Ayinla *et al.* (2024), optou-se por implementação direta dos algoritmos de force feedback devido às limitações computacionais e à necessidade de resposta em tempo real inferior

a 5ms.

#### 2.4.5.1 Cálculo das Forças G

O algoritmo para cálculo das forças G considera os valores de aceleração linear e velocidade angular obtidos do sensor BMI160:

$$G_{frontal} = \frac{aceleração_{linear\_X}}{9.81} \quad (2.2)$$

$$G_{lateral} = \frac{aceleração_{linear\_Y}}{9.81} \quad (2.3)$$

#### 2.4.5.2 Algoritmos de Controle de Force Feedback

O sistema utiliza os dados de aceleração linear do sensor BMI160 para calcular as forças G experimentadas pelo veículo. O acelerômetro fornece diretamente os valores de aceleração nos três eixos, permitindo cálculo preciso das forças G frontal (eixo X) e lateral (eixo Y). Essa abordagem segue o trabalho de Dreger e Rinkenauer (2024), que demonstraram que feedback em tempo real melhora significativamente a precisão do operador.

O algoritmo 2 apresenta a implementação do cálculo de forças G. A Figura 7 ilustra graficamente a relação entre aceleração e força G calculada.

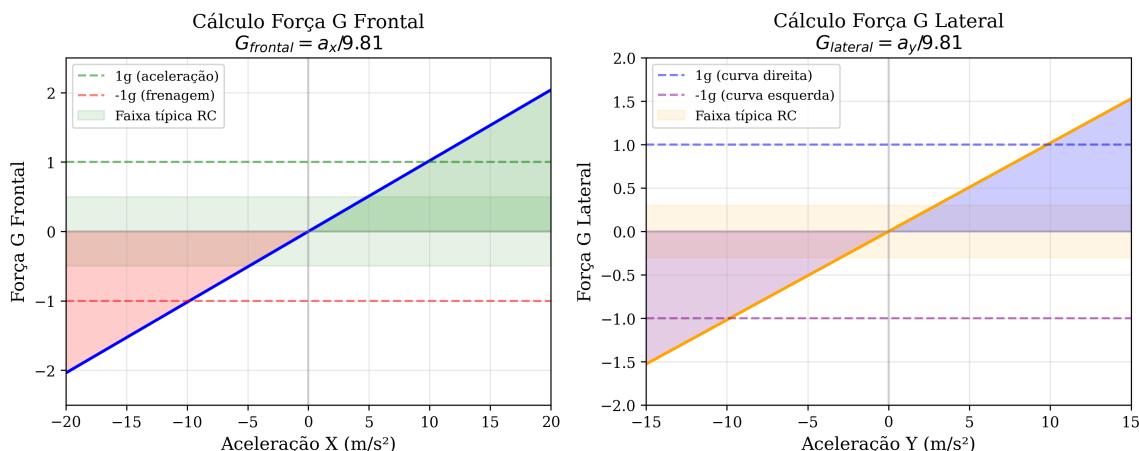


Figura 7 – Cálculo das forças G frontal e lateral a partir dos dados do acelerômetro

---

**Algoritmo 2:** Cálculo das forças G

---

**Entrada:** accel\_x, accel\_y, accel\_z

**Saída:** g\_frontal, g\_lateral, g\_vertical

**início**

```
// Leitura dos sensores (valores em m/s2);
accel_x ← ler_acelerometro_x();
accel_y ← ler_acelerometro_y();
accel_z ← ler_acelerometro_z();

// Cálculo das forças G;
g_frontal ← accel_x / 9.81;
g_lateral ← accel_y / 9.81;
g_vertical ← (accel_z - 9.81) / 9.81;
```

**fim**

---

#### 2.4.5.3 Componentes do Force Feedback da Direção

O sistema de force feedback da direção combina três componentes principais, conforme ilustrado na Figura 8: força lateral (curvas), rotação yaw e mola de centralização. A força base é calculada como a soma dos três componentes, limitada a 100%.

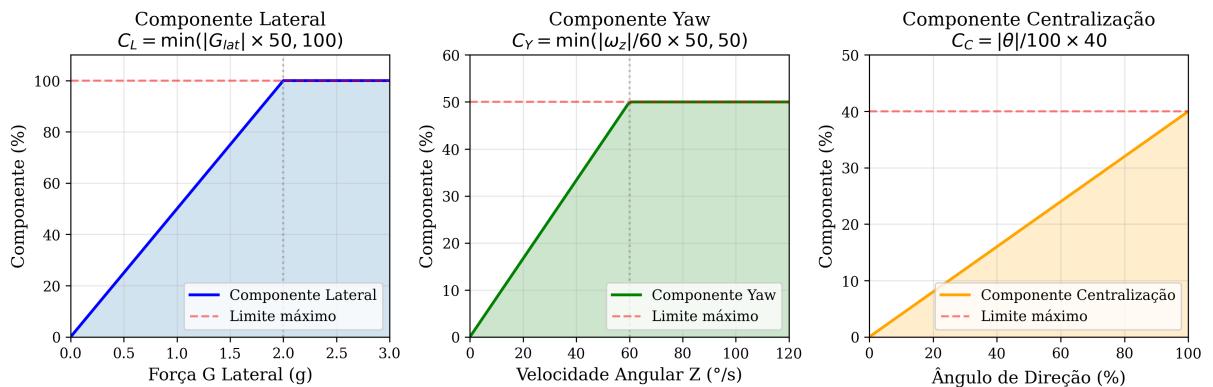


Figura 8 – Componentes do force feedback da direção: lateral, yaw e centralização

O componente lateral representa a força experimentada pelo volante durante curvas, proporcional à força G lateral. O componente yaw captura a rotação do veículo em torno do eixo vertical, detectada pelo giroscópio. O componente de centralização simula a mola de retorno do sistema de direção, aplicando força proporcional ao ângulo de esterçamento para retornar o volante à posição central.

A Figura 9 apresenta a contribuição de cada componente em cenários típicos de condução, desde reta em velocidade constante até curvas no limite de aderência.

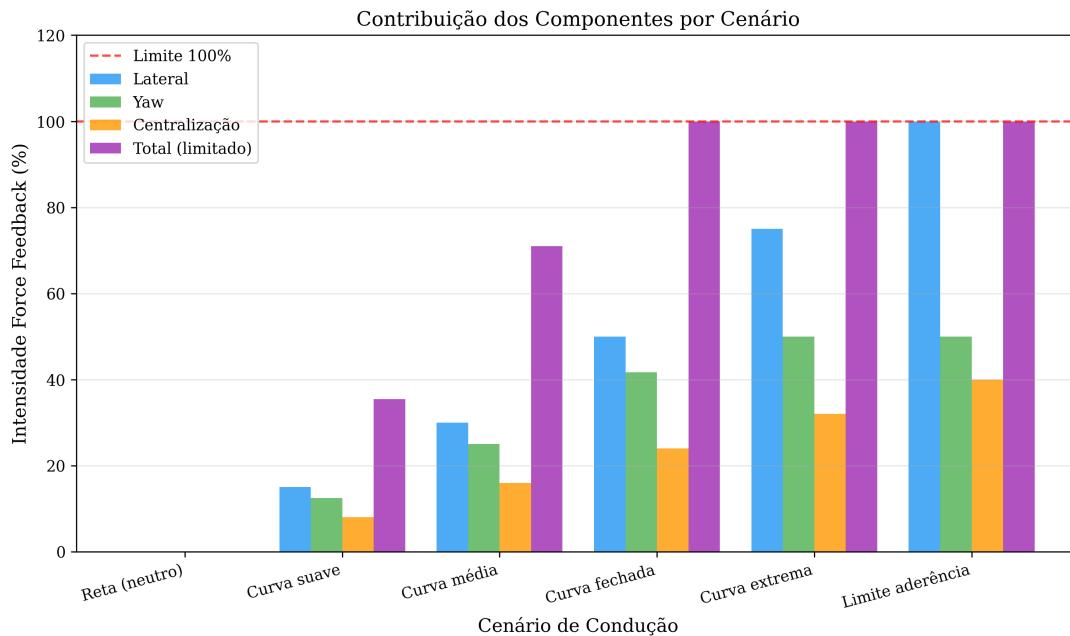


Figura 9 – Contribuição dos componentes de force feedback por cenário de condução

O algoritmo é executado no cliente PC e envia comandos para o motor DC 775 do simulador via ESP32:

#### 2.4.5.4 Parâmetros Ajustáveis de Force Feedback

O sistema permite ajuste em tempo real de quatro parâmetros que modificam a resposta do force feedback, conforme ilustrado na Figura 10. A sensibilidade (padrão 75%) escala a intensidade geral da força. A fricção (padrão 30%) adiciona resistência proporcional à velocidade angular do veículo, simulando o atrito dos pneus. O filtro (padrão 40%) aplica suavização exponencial para eliminar oscilações de alta frequência. O damping (padrão 50%) implementa média móvel para simular inércia mecânica do sistema de direção.

#### 2.4.5.5 Determinação da Direção do Force Feedback

A direção do force feedback é calculada combinando três componentes direcionais: centralização (força contrária ao ângulo de esterçamento), força lateral (proporcional à força G) e rotação yaw (velocidade angular). O valor resultante determina se a força será aplicada para esquerda, direita ou neutro, conforme ilustrado na Figura 11.

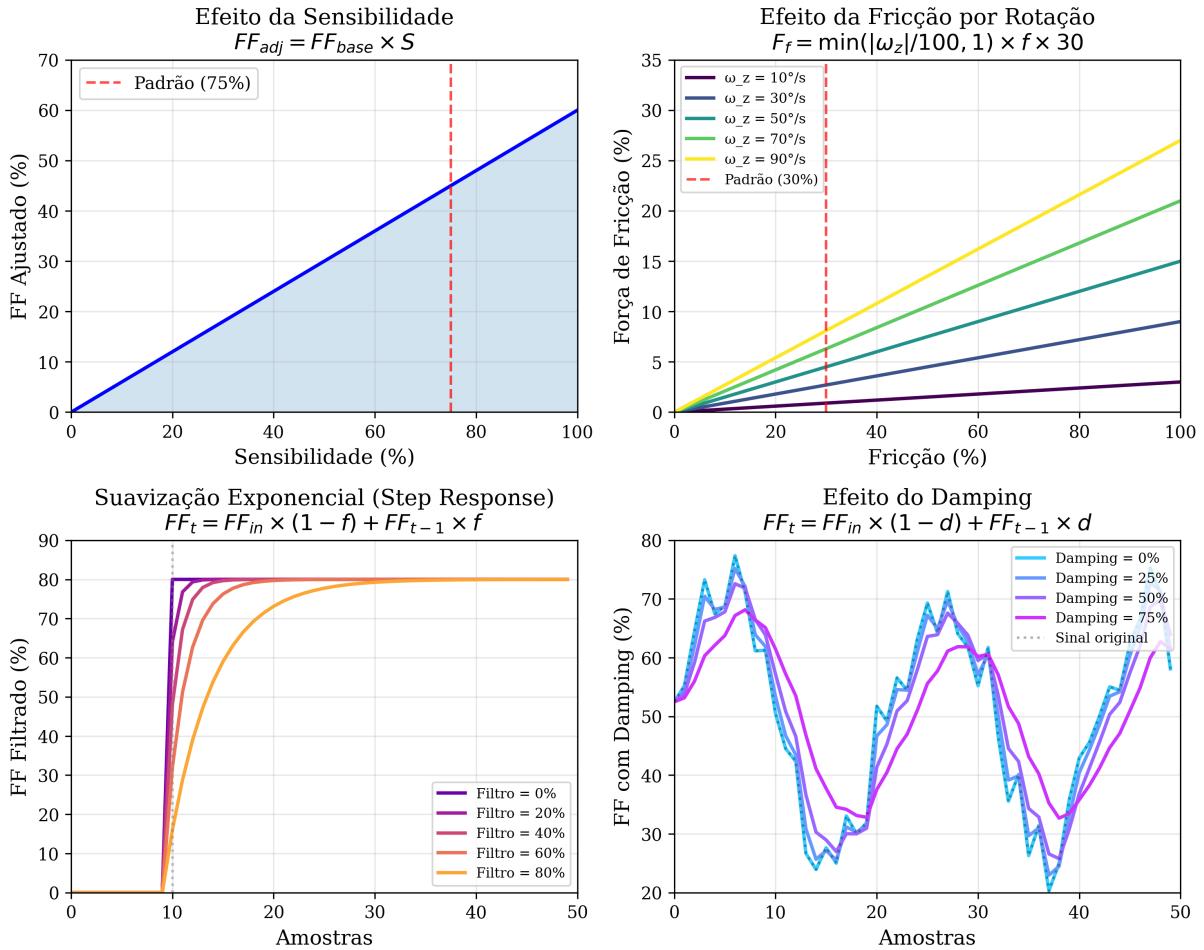


Figura 10 – Efeito dos parâmetros ajustáveis no force feedback

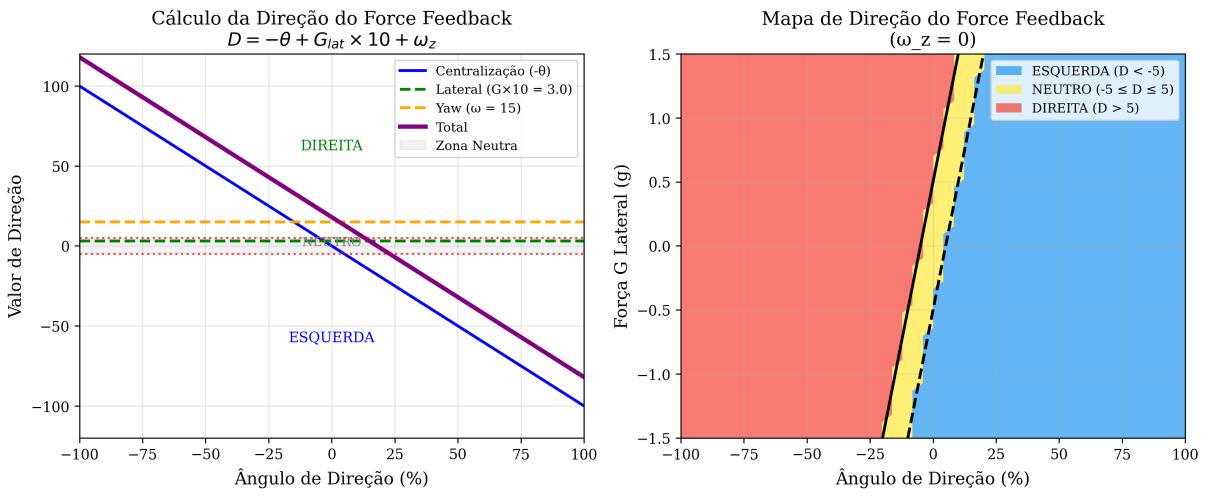


Figura 11 – Cálculo da direção do force feedback e mapa de zonas

#### 2.4.5.6 Mapeamento PWM do Motor

O comando de intensidade calculado (0-100%) é transmitido via USB serial para o ESP32, que converte o valor para ciclo de trabalho PWM de 8 bits (0-255) através de mapeamento linear. A ponte H BTS7960 recebe o sinal PWM e aciona o motor DC 775 na direção correspondente.

$$PWM = \frac{\text{intensidade} \times 255}{100} \quad (2.4)$$

#### 2.4.5.7 Cálculo de Velocidade por Integração

A velocidade do veículo é estimada através da integração numérica da aceleração medida pelo sensor BMI160. Um fator de decaimento é aplicado para simular atrito e resistência do ar, conforme ilustrado na Figura 12:

$$v(t) = v(t - 1) + a \times \Delta t \quad (2.5)$$

$$v_{final} = v(t) \times fator_{decaimento} \quad (2.6)$$

#### 2.4.6 Sistema de Telemetria em Tempo Real

O sistema de telemetria exibe gráficos em tempo real com visual inspirado nas interfaces de engenharia de equipes de Fórmula 1, utilizando a biblioteca Matplotlib integrada ao framework gráfico Tkinter. Os dados de sensores são armazenados em Buffer Circulars implementados com estruturas *deque* do Python, com capacidade máxima de 500 pontos. Essa capacidade representa aproximadamente 50 segundos de histórico considerando a taxa de atualização de 10Hz, permitindo ao operador visualizar tendências recentes sem consumo excessivo de memória. O algoritmo 5 apresenta o processo de atualização dos gráficos.

#### 2.4.7 Sistema de Auto-Save e Exportação de Dados

O sistema implementa salvamento automático periódico para garantir a preservação dos dados coletados durante as sessões de operação. A cada 20 segundos, o sistema verifica se

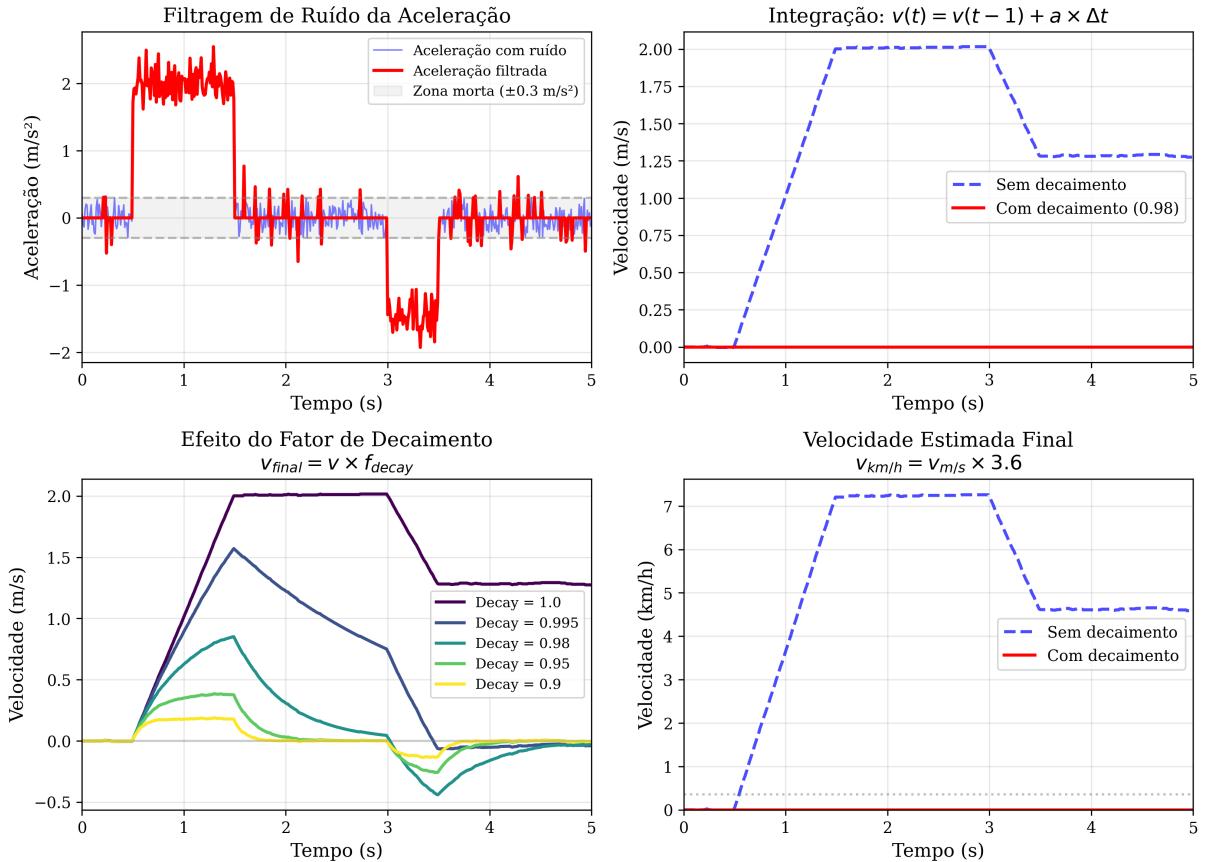


Figura 12 – Processo de integração de aceleração para estimativa de velocidade

há quantidade suficiente de dados novos e, em caso positivo, exporta logs, dados de sensores e telemetria para arquivos no diretório `exports/auto/`. O formato Pickle (binário nativo do Python) foi escolhido por oferecer performance 5 a 10 vezes superior ao CSV para Serialização de estruturas de dados complexas. Após cada salvamento, os buffers são limpos para evitar duplicação de dados em exportações subsequentes. O algoritmo 6 apresenta a lógica de verificação e salvamento.

## 2.5 Armazenamento de Logs e Validação

O sistema de armazenamento de dados implementa salvamento automático periódico para garantir a preservação de informações de telemetria durante as sessões de operação. Os dados armazenados permitem validação das telemetrias entre corridas e análise posterior, de forma similar às equipes de Fórmula 1 que utilizam dados de sessões para otimização de desempenho.

### 2.5.1 Formatos de Armazenamento

A seleção dos formatos de armazenamento considerou velocidade de serialização, tamanho dos arquivos e facilidade de processamento posterior. A Tabela 17 apresenta os formatos utilizados para cada tipo de dado.

O formato Pickle foi escolhido para dados numéricos por oferecer serialização binária nativa do Python, preservando estruturas complexas contendo listas e dicionários aninhados. Embora o Pickle não seja legível por humanos, a análise dos dados é realizada através de scripts Python que reconstruem as estruturas originais. Os arquivos de log mantêm formato texto para permitir inspeção manual rápida durante depuração.

### 2.5.2 Estrutura dos Dados Armazenados

Os arquivos de sensores contêm dicionários Python com listas de valores para cada canal do sensor BMI160 e do sistema de monitoramento de energia: `timestamp`, `bmi160_accel_x`, `bmi160_accel_y`, `bmi160_accel_z`, `bmi160_gyro_x`, `bmi160_gyro_y`, `bmi160_gyro_z`, `g_force_lateral`, `g_force_frontal`, `current_motor`, `current_servos`, `current_rpi`, `power_total` e `voltage_rpi`. Cada lista contém valores correspondentes ao mesmo índice temporal, permitindo correlação entre diferentes grandezas.

Os arquivos de telemetria armazenam dados processados para os gráficos da interface: `time`, `speed`, `throttle`, `brake`, `g_lateral`, `g_frontal` e metadados como `start_time`, `points_count` e `export_time`. Esses dados representam valores já filtrados e normalizados, prontos para visualização.

### 2.5.3 Script de Análise de Sessões

O script `analyze_session.py` foi desenvolvido para processar os arquivos gerados pelo sistema de auto-save, oferecendo análise estatística e visualização gráfica das sessões de operação. O script suporta três modos de operação: análise combinada de todos os arquivos do diretório (padrão), análise apenas dos arquivos mais recentes e análise de timestamp específico.

O processamento combina múltiplos arquivos Pickle em estruturas unificadas, concatenando as listas de cada canal para formar séries temporais contínuas. As estatísticas calculadas incluem duração total da sessão, velocidade máxima, média e mínima, tempo em aceleração total (*full throttle*), tempo em frenagem, forças G laterais e frontais máximas, acelerações e

velocidades angulares extremas, e consumo de energia quando disponível.

A visualização gráfica utiliza a biblioteca Matplotlib com estilo visual inspirado nas telemetrias de Fórmula 1, apresentando gráficos de velocidade ao longo do tempo, utilização de pedais, forças G, distribuição de velocidade em histograma e diagrama G-G (*scatter plot* de força lateral versus frontal). A análise de logs identifica padrões de erros, avisos, conexões e desconexões através de expressões regulares.

O script também suporta exportação de relatório HTML formatado contendo todas as estatísticas e metadados da sessão, facilitando documentação e compartilhamento dos resultados experimentais. A algoritmo 7 apresenta o fluxo principal do processamento.

A leitura dos arquivos Pickle para análise externa pode ser realizada através do seguinte código Python:

```
import pickle

with open("sensors_20241216_143000.pkl", "rb") as f:
    dados = pickle.load(f)

# dados contém dicionário com listas de cada sensor
print(dados.keys()) # ['timestamp', 'bmi160_accel_x', ...]
print(len(dados['timestamp'])) # Quantidade de amostras
```

#### **2.5.4 Métricas de Validação Extraídas**

Os dados armazenados pelo sistema de auto-save permitem a extração de métricas específicas para validação do desempenho do sistema. A Tabela 18 apresenta as métricas extraídas de cada categoria de dados, organizadas por subsistema avaliado.

O script `analyze_session.py` calcula automaticamente as estatísticas principais a partir dos dados armazenados. Para métricas de comunicação, o intervalo entre timestamps consecutivos permite estimar a latência efetiva do sistema, enquanto o desvio padrão desses intervalos caracteriza o jitter. A análise de logs por expressões regulares identifica eventos de erro, permitindo cálculo da taxa de perda de pacotes.

Para validação estatística dos resultados, o volume de dados coletado pelo sistema de auto-save garante robustez. Considerando taxa de amostragem de 100Hz para sensores e sessões típicas de 15 minutos, cada sessão gera aproximadamente 90.000 pontos de telemetria. Esse volume supera significativamente o mínimo de 384 amostras requerido para população infinita

com 95% de confiança e margem de erro de 5%, proporcionando poder estatístico superior a 99% para detectar diferenças com tamanho de efeito médio nas comparações com o estado da arte.

## **2.6 Reprodutibilidade e Transparência Experimental**

A garantia de reproduzibilidade científica constitui um pilar fundamental para a validação e evolução do conhecimento em engenharia de sistemas embarcados. Segundo as diretrizes estabelecidas por Graf *et al.* (2024) para avaliação de desempenho de sistemas wireless, este trabalho adota protocolos rigorosos de documentação e disponibilização de recursos para permitir a replicação completa dos experimentos realizados.

### **2.6.1 Disponibilização de Código Fonte e Datasets**

Todo o código fonte desenvolvido para este projeto está disponibilizado publicamente no repositório GitHub sob licença MIT, acessível em <<https://github.com/inacio-dev/tcc>>. O repositório contém a estrutura completa do projeto, organizada conforme a Tabela 19.

Conforme demonstrado por Bobrovsky *et al.* (2023), a transparência no desenvolvimento de sistemas embarcados facilita a reprodução e melhoria contínua das soluções propostas. Os datasets coletados durante as sessões experimentais, totalizando mais de 90.000 pontos de telemetria, estão disponíveis no diretório `exports/auto/` do repositório em formato Pickle, permitindo análise direta com o script `analyze_session.py` incluso no projeto.

### **2.6.2 Protocolo Detalhado de Replicação**

O protocolo de replicação experimental documentado contempla todos os aspectos críticos para reprodução fidedigna dos resultados obtidos. As especificações de hardware incluem números de modelo exatos, versões de firmware e configurações específicas de cada componente utilizado no sistema. Para o Raspberry Pi 4, documenta-se a versão do Raspberry Pi OS (Bullseye 64-bit), kernel utilizado (5.15.84-v8+) e configurações específicas do arquivo `config.txt` para otimização da câmera OV5647.

As configurações de rede wireless seguem padrões reproduzíveis, especificando canal WiFi (2.4GHz canal 6), potência de transmissão (20dBm), tipo de roteador utilizado (TP-Link Archer C6) e posicionamento físico dos equipamentos. As condições ambientais durante os experimentos são registradas detalhadamente, incluindo temperatura ambiente ( $22\pm2^{\circ}\text{C}$ ),

umidade relativa ( $45\pm5\%$ ), interferências eletromagnéticas medidas e layout físico do ambiente de teste.

### **2.6.3 Documentação de Configurações Ambientais**

As configurações ambientais experimentais são registradas sistematicamente para garantir reproduzibilidade das condições de teste. O ambiente de rede é caracterizado através de medições de potência de sinal WiFi utilizando ferramentas como iwconfig e wavemon, documentando valores de *Received Signal Strength Indicator* (RSSI) em múltiplos pontos do ambiente experimental.

As características do ambiente físico incluem dimensões precisas do local de teste ( $5m \times 8m$ ), materiais de construção das paredes (alvenaria com reboco), presença de obstáculos metálicos e fontes potenciais de interferência eletromagnética. O posicionamento relativo entre veículo teleoperado e estação de controle é documentado com coordenadas precisas, utilizando sistema de referência baseado em marcos físicos permanentes.

As condições de iluminação para os testes de câmera são padronizadas utilizando iluminação artificial controlada (lâmpadas LED 6500K, 1000 lúmens), minimizando variações devido à iluminação natural. Os padrões de teste visual incluem alvos de calibração com dimensões conhecidas, permitindo validação da qualidade de captura de vídeo independente das condições específicas do ambiente.

O controle de temperatura ambiente utiliza sistema de climatização para manter estabilidade térmica durante as sessões experimentais, evitando drift térmico excessivo nos sensores. Registros contínuos de temperatura e umidade são mantidos através de datalogger dedicado, correlacionando variações ambientais com performance do sistema.

---

**Algoritmo 3:** Cálculo do force feedback da direção

---

**Entrada:** g\_lateral, gyro\_z, angulo\_direcao, sensibilidade, friccao, filtro, damping**Saída:** intensidade\_ff, direcao\_ff**início**

```

// Componente 1: Força lateral (curvas) - máximo 100%;
componente_lateral ← minimo(abs(g_lateral) × 50, 100);
// Componente 2: Rotação yaw - máximo 50%;
componente_yaw ← minimo(abs(gyro_z) / 60.0 × 50, 50);
// Componente 3: Mola de centralização - máximo 40%;
razao_angulo ← abs(angulo_direcao) / 100.0;
componente_centragem ← razao_angulo × 40;
// Força base combinada (0-100%);
ff_base ← minimo(componente_lateral + componente_yaw +
componente_centragem, 100);
// Aplica sensibilidade;
ff_ajustado ← ff_base × sensibilidade;
// Aplica fricção baseada na rotação;
forca_friccao ← minimo(abs(gyro_z) / 100.0, 1.0) × friccao × 30;
ff_ajustado ← minimo(ff_ajustado + forca_friccao, 100.0);
// Aplica filtro (suavização exponencial);
ff_ajustado ← ff_ajustado × (1.0 - filtro) + ff_filtrado_anterior × filtro;
// Aplica damping (média móvel);
intensidade_ff ← ff_ajustado × (1.0 - damping) + ff_anterior × damping;
// Determina direção do force feedback;
valor_direcao ← (-angulo_direcao) + (g_lateral × 10) + gyro_z;
se valor_direcao > 5 então
    | direcao_ff ← "DIREITA";
fim
senão se valor_direcao < -5 então
    | direcao_ff ← "ESQUERDA";
fim
senão
    | direcao_ff ← "NEUTRO";
fim
fim

```

---

---

**Algoritmo 4:** Cálculo de velocidade por integração
 

---

**Entrada:** accel\_x, accel\_y, tempo\_atual

**Saída:** velocidade\_total\_kmh

**início**

```

LIMIAR_ACCEL ← 0.3;
FATOR_DECAIMENTO ← 0.98;
LIMIAR_VELOCIDADE ← 0.1;
// Calcula delta time;
dt ← tempo_atual - tempo_anterior;
tempo_anterior ← tempo_atual;
// Filtra ruído da aceleração;
se abs(accel_x) < LIMIAR_ACCEL então
  |   accel_x ← 0;
fim
se abs(accel_y) < LIMIAR_ACCEL então
  |   accel_y ← 0;
fim
// Integração: v = v0 + a × dt;
velocidade_x ← velocidade_x + accel_x × dt;
velocidade_y ← velocidade_y + accel_y × dt;
// Aplica decaimento (simula atrito);
velocidade_x ← velocidade_x × FATOR_DECAIMENTO;
velocidade_y ← velocidade_y × FATOR_DECAIMENTO;
// Zera velocidades muito pequenas;
se abs(velocidade_x) < LIMIAR_VELOCIDADE então
  |   velocidade_x ← 0;
fim
se abs(velocidade_y) < LIMIAR_VELOCIDADE então
  |   velocidade_y ← 0;
fim
// Calcula magnitude e converte para km/h;
velocidade_ms ← raiz(velocidade_x2 + velocidade_y2);
velocidade_total_kmh ← velocidade_ms × 3.6;

```

**fim**

---

---

**Algoritmo 5:** Atualização dos gráficos de telemetria
 

---

**Entrada:** dados\_sensores

**Saída:** graficos\_atualizados

**início**

```

    MAX_PONTOS ← 500;
    JANELA_TEMPO ← 30;
    // Calcula tempo relativo desde o início;
    tempo_atual ← tempo_sistema() - tempo_inicio;
    // Adiciona dados aos buffers circulares;
    buffer_tempo.adicionar(tempo_atual);
    buffer_velocidade.adicionar(dados_sensores["velocidade"]);
    buffer_acelerador.adicionar(dados_sensores["acelerador"]);
    buffer_freio.adicionar(dados_sensores["freio"]);
    buffer_g_lateral.adicionar(dados_sensores["g_lateral"]);
    buffer_g_frontal.adicionar(dados_sensores["g_frontal"]);
    // Ajusta janela de visualização (últimos 30 segundos);
    x_min ← maximo(0, tempo_atual - JANELA_TEMPO);
    x_max ← tempo_atual + 1;
    // Atualiza linhas dos gráficos;
    linha_velocidade.atualizar(buffer_tempo, buffer_velocidade);
    linha_acelerador.atualizar(buffer_tempo, buffer_acelerador);
    linha_freio.atualizar(buffer_tempo, buffer_freio);
    linha_g_lateral.atualizar(buffer_tempo, buffer_g_lateral);
    linha_g_frontal.atualizar(buffer_tempo, buffer_g_frontal);
    // Redesenha canvas;
    canvas.redesenhar();
  
```

**fim**

---

---

**Algoritmo 6:** Auto-save periódico de dados
 

---

**Entrada:** console, intervalo\_ms

**Saída:** arquivos\_salvos

**início**

```

    MIN_LOGS ← 100;
    MIN_SENORES ← 1000;
    MIN_TELEMETRIA ← 100;
    DIRETORIO ← "exports/auto/";
    // Verifica se há dados novos suficientes;
    qtd_logs ← contar_linhas(console.log_text);
    qtd_sensores ← tamanho(sensor_display.historico);
    qtd_telemetria ← tamanho(telemetria.buffer_tempo);
    // Só salva se atingir mínimo E houver dados novos;
    se ( $qtd\_logs \geq MIN\_LOGS$ ) OU ( $qtd\_sensores \geq MIN\_SENORES$ ) OU
      ( $qtd\_telemetria \geq MIN\_TELEMETRIA$ ) então
        timestamp ← formato_data("YYYYMMDD_HHMMSS");
        // Salva logs em texto;
        se  $qtd\_logs \geq MIN\_LOGS$  então
          salvar_arquivo(DIRETORIO + "logs_" + timestamp + ".txt",
            console.log_text);
        fim
        // Salva sensores em Pickle (binário);
        se  $qtd\_sensores \geq MIN\_SENORES$  então
          pickle.salvar(DIRETORIO + "sensors_" + timestamp + ".pkl",
            sensor_display.historico);
        fim
        // Salva telemetria em Pickle;
        se  $qtd\_telemetria \geq MIN\_TELEMETRIA$  então
          pickle.salvar(DIRETORIO + "telemetry_" + timestamp + ".pkl",
            telemetria.dados);
        fim
        // Limpa buffers após salvar;
        console.log_text.limpar();
        sensor_display.resetar();
        telemetria.resetar();
    fim
    // Reagenda próximo auto-save;
    agendar(intervalo_ms, auto_save);
  fim

```

---

Tabela 17 – Formatos de armazenamento utilizados no sistema de auto-save

Tipo de Dado	Formato	Nomenclatura	Justificativa
Logs do console	Texto (.txt)	logs_YYYYMMDD_HHMMSS.txt	Legibilidade humana, facilidade de inspeção
Sensores brutos	Pickle (.pkl)	sensors_YYYYMMDD_HHMMSS.pkl	Serialização binária eficiente
Telemetria	Pickle (.pkl)	telemetry_YYYYMMDD_HHMMSS.pkl	Preservação de estruturas Python complexas

Fonte: elaborado pelo autor.

---

### Algoritmo 7: Análise de sessão de telemetria

---

**Entrada:** diretório de dados, modo de operação

**Saída:** estatísticas, gráficos, relatório

**início**

```

// Localiza arquivos por padrão de nomenclatura;
arquivos_telemetria ← glob("telemetry_*.pkl");
arquivos_sensores ← glob("sensors_*.pkl");
arquivos_logs ← glob("logs_*.txt");
// Carrega e combina dados de múltiplos arquivos;
dados_combinados ← dicionário vazio;
para cada arquivo em arquivos_telemetria faz
    dados ← pickle.load(arquivo);
    para cada chave, valores em dados faz
        | dados_combinados[chave].estender(valores);
    fim
fim

// Calcula estatísticas com NumPy;
velocidade ← array(dados_combinados["speed"]);
estatisticas.velocidade_max ← max(velocidade);
estatisticas.velocidade_media ← media(velocidade);
// Gera gráficos estilo F1;
plotar_telemetria(dados_combinados);
plotar_sensores(dados_sensores);
// Exporta relatório HTML;
exportar_html(estatisticas, analise_logs);
fim

```

---

**Tabela 18 – Métricas de validação extraídas dos dados armazenados**

Subsistema	Métrica	Fonte de Dados	Cálculo
Comunicação	Latência média	timestamp	Diferença entre envio e receção
	Jitter	timestamp	Desvio padrão dos intervalos
	Packet loss	logs_*.txt	Contagem de erros de receção
Vídeo	FPS efetivo	telemetry_*.pkl	Frames por segundo médio
	Estabilidade	telemetry_*.pkl	Variância do intervalo entre frames
	Throughput	sensors_*.pkl	Bytes por segundo
Sensores	Taxa de amostragem	sensors_*.pkl	Amostras por segundo
	Drift Térmico	temperature	Variação ao longo do tempo
	Ruído	bmi160_accel_*	Desvio padrão em repouso
Force Feedback	Tempo de resposta	timestamp	Latência comando-atuação
	Precisão	g_force_*	Correlação sensor-feedback
	Estabilidade	gyro_z	Variância em regime permanente
Energia	Consumo médio	power_total	Média de potência
	Pico de corrente	current_motor	Valor máximo registrado

Fonte: elaborado pelo autor.

**Tabela 19 – Estrutura do repositório de código fonte**

Diretório	Conteúdo
raspberry/	Scripts Python para controle do Raspberry Pi 4: captura de vídeo, leitura de sensores BMI160, controle de motor e servos, comunicação UDP
client/	Interface gráfica em Python/Tkinter, recepção UDP, processamento de telemetria, sistema de auto-save e script de análise de sessões
esp32/	Firmware C++ para ESP32: leitura de encoders, controle de force feedback, comunicação serial USB
exports/auto/	Datasets de telemetria em formato Pickle (.pkl) e logs de sessões (.txt)
scripts/	Rotinas de análise estatística e geração de gráficos
docs/	Documentação técnica das decisões de projeto
monografia/	Código fonte L <sup>A</sup> T <sub>E</sub> X deste documento

Fonte: elaborado pelo autor.

## REFERÊNCIAS

- AJAYI, O. K.; DU, S.; NAHRI, S. N. F. A review of haptic technologies for hardware-in-the-loop development. **Sensors and Actuators Reports**, Elsevier, 2025.
- Allegro MicroSystems. **ACS758 – Thermally Enhanced, Fully Integrated, Hall-Effect-Based Linear Current Sensor IC**. [S.I.], 2018. Datasheet.
- AN, D.; JOO, H.; KIM, H. Enabling low-latency digital twins for large-scale uav networks using mqtt-based communication framework. **ICT Express**, Elsevier, 2025.
- Autodesk. **Tinkercad – From mind to design in minutes**. 2024. Software de modelagem 3D online. Disponível em: <<https://www.tinkercad.com/>>.
- AYINLA, S. L.; AMOSA, T. I.; IBRAHIM, O.; RAHMAN, M. S.; BAHASHWAN, A. A.; MOSTAFA, M. G.; YUSUF, A. O. Optimal control of dc motor using leader-based harris hawks optimization algorithm. **Franklin Open**, Elsevier, 2024.
- BARÓN, M.; DIEZ, L.; ZVEREV, M.; JUÁREZ, J. R.; AGÜERO, R. On the performance of zenoh in industrial iot scenarios. **Ad Hoc Networks**, Elsevier, 2025.
- BOBROVSKY, A. V.; DROBCHENKO, A. E.; ZOTOV, A. V.; GOROKHOVA, D. A.; CHIZHATKINA, E. D. Development of a universal module for connecting sensors to the can-bus for the formula student electric car. **Transportation Research Procedia**, Elsevier, 2023.
- Bosch Sensortec. **BMI160 – Small, Low Power Inertial Measurement Unit**. [S.I.], 2015. Datasheet BST-BMI160-DS000-09.
- CAÑADAS-ARÁNEGA, F.; MORENO, J. C.; BLANCO-CLARACO, J. L. A pid-based control architecture for mobile robot path planning in greenhouses. **IFAC PapersOnLine**, Elsevier / IFAC, 2024.
- CHEN, S.; NOGUCHI, N. Remote safety system for a robot tractor using a monocular camera and a yolo-based method. **Computers and Electronics in Agriculture**, Elsevier, 2023.
- DENG, Z.; ZHANG, S.; GUO, Y.; JIANG, H.; ZHENG, X.; HE, B. Assisted teleoperation control of robotic endoscope with visual feedback for nasotracheal intubation. **Robotics and Autonomous Systems**, Elsevier, 2024.
- DREGER, F. A.; RINKENAUER, G. Evaluation of different feedback designs for target guidance in human controlled robotic cranes: A comparison between high and low performance groups. **Applied Ergonomics**, Elsevier, 2024.
- Espressif Systems. **ESP32 Series Datasheet**. [S.I.], 2024. Datasheet V4.6.
- GONZALEZ, R. E. W. R. C. **Processamento Digital de Imagens**. 3. ed. [S.I.]: Pearson Prentice Hall, 2009.
- GRAF, F.; WATTEYNÉ, T.; VILLNOW, M. Monitoring performance metrics in low-power wireless systems. **ICT Express**, Elsevier, 2024.
- GöKÇE, C. O.; İPEK, M. E.; DAYIÖĞLU, M.; ÜNAL, R. Parameter estimation and speed control of real dc motor with low resolution encoder. **Results in Control and Optimization**, Elsevier, 2025.

HUO, F.; WANG, T.; FANG, F.; SUN, C. The influence of tactile feedback in in-vehicle central control interfaces on driver emotions: A comparative study of touchscreens and physical buttons. **International Journal of Industrial Ergonomics**, Elsevier, 2024.

Infineon Technologies AG. **BTN7960/BTN7960B – High Current PN Half Bridge**. [S.I.], 2013. Datasheet Rev. 1.0.

IQBAL, F.; GOHAR, M.; KARAMTI, H.; KARAMTI, W.; KOH, S.-J.; CHOI, J.-G. Use of quic for amqp in iot networks. **Computer Networks**, Elsevier, 2023.

ITO, K.; NAKAZATO, J.; FONTUGNE, R.; TSUKADA, M.; HIROSHI, E. A multipath redundancy communication framework for enhancing 5g mobile communication quality. **Computer Communications**, Elsevier, 2025.

JI, Q.; JANSSON, J.; SJÖBERG, M.; WANG, X. V.; WANG, L.; FENG, L. Design and calibration of 3d printed soft deformation sensors for soft actuator control. **Mechatronics**, Elsevier, 2023.

KHODAMPOUR, G.; KHORASHADIZADEH, S.; FARSHAD, M. Adaptive formation control of leader-follower mobile robots using reinforcement learning and the fourier series expansion. **ISA Transactions**, Elsevier, 2023.

KUMARI, M.; SINGH, M. P.; SINGH, A. K. A latency sensitive and agile iiot architecture with optimized edge node selection and task scheduling. **Digital Communications and Networks**, Elsevier, 2025.

LI, W.; HUANG, F.; CHEN, Z.; CHEN, Z. Automatic-switching-based teleoperation framework for mobile manipulator with asymmetrical mapping and force feedback. **Mechatronics**, Elsevier, 2024.

LU, Q.; LI, J.; YUAN, K.; LIU, K.; NI, M.; LUO, J. Udp-rt: A udp-based reliable transmission scheme for power waps. **Computer Networks**, Elsevier, 2023.

MANUEL, N. L.; INANÇ, N.; LÜY, M. Control and performance analyses of a dc motor using optimized pids and fuzzy logic controller. **Results in Control and Optimization**, Elsevier, 2023.

Maxim Integrated. **DS18B20 – Programmable Resolution 1-Wire Digital Thermometer**. [S.I.], 2019. Datasheet 19-7487.

nacho3D. **F1 McLaren MP4-30 2015 Steering Wheel**. 2020. Printables. Modelo 3D. Disponível em: <<https://www.printables.com/model/46901-f1-mclaren-mp4-30-2015-steering-wheel>>.

NXP Semiconductors. **PCA9685 – 16-channel, 12-bit PWM Fm+ I2C-bus LED controller**. [S.I.], 2015. Datasheet Rev. 4.

RODRÍGUEZ-ARELLANO, J. A.; MIRANDA-COLORADO, R.; VILLAFUERTE-SEGURA, R.; AGUILAR, L. T. Experimental observer-based delayed control of wheeled mobile robots. **Applied Mathematical Modelling**, Elsevier, 2025.

SANTOS, V. S.; ERAS, J. J. C.; ULLOA, M. J. C. Evaluation of the energy saving potential in electric motors applying a load-based voltage control method. **Energy**, Elsevier, 2024.

SHAIK, Z. B.; PEDDAKRISHNA, S. Design and implementation of electric vehicle with autonomous motion and steering control system using single board computer and sensors. **Results in Engineering**, Elsevier, 2025.

SHENDGE, A.; SINGH, R.; ANSARI, K. I.; PAKHRANI, K. Development of an unmanned aerial vehicle for remote live streaming on web dashboard. **Materials Today: Proceedings**, Elsevier, 2023.

SHUKLA, A.; ROSS, R.; BHATTACHARYA, B.; STUMPF, A. Autonomous water sampling and quality monitoring in remote locations: A novel approach using a remote-controlled boat. **HardwareX**, Elsevier, 2025.

Texas Instruments. **INA219 – Zero-Drift, Bidirectional Current/Power Monitor with I2C Interface**. [S.l.], 2015. Datasheet SBOS448G.

Texas Instruments. **ADS1115 – Ultra-Small, Low-Power, 16-Bit Analog-to-Digital Converter with Internal Reference**. [S.l.], 2018. Datasheet SBAS444C.

UEMURA, R.; ASAKURA, T. Cross-modal feedback of tactile and auditory stimuli for cyclists in noisy environments. **Sensors and Actuators: A. Physical**, Elsevier, 2024.

UltiMaker. **UltiMaker Cura – Trusted by millions of users worldwide**. 2024. Software fatiador para impressão 3D. Disponível em: <<https://ultimaker.com/software/ultimaker-cura/>>.

VASHISHT, A.; GANDHI, G. C.; KALRA, S.; SAINI, D. K. Hybrid robot navigation: Integrating monocular depth estimation and visual odometry for efficient navigation on low-resource hardware. **Computers and Electrical Engineering**, Elsevier, 2025.

VelocityProjects3D. **FV01 – The Most Advanced 3D Printed RC Formula 1 Car**. 2024. Printables. Modelo 3D. Disponível em: <<https://www.printables.com/model/964784-fv01-the-most-advanced-3d-printed-rc-formula-1-car>>.

XIA, P.; YOU, H.; DU, J. Visual-haptic feedback for rov subsea navigation control. **Automation in Construction**, Elsevier, 2023.

XLSEMI. **XL4015 – 5A 180KHz 36V Buck DC to DC Converter**. [S.l.], 2017. Datasheet V2.5.

## GLOSSÁRIO

ACK	<i>Acknowledgment</i>
ADC	<i>Analog-to-Digital Converter</i>
Back-EMF	<i>Back Electromotive Force</i> (Força Eletromotriz de Retorno). Tensão gerada por um motor elétrico quando seu eixo gira, seja por acionamento elétrico ou força mecânica externa, atuando como gerador
Buffer Circular	Estrutura de dados em forma de fila circular onde, ao atingir a capacidade máxima, novos elementos sobrescrevem os mais antigos. Também conhecido como <i>ring buffer</i> ou <i>deque</i> com tamanho fixo
CLAHE	<i>Contrast Limited Adaptive Histogram Equalization</i>
Convolução	Operação matemática fundamental em processamento de imagens onde uma máscara (kernel) é aplicada sobre cada pixel da imagem para produzir efeitos como suavização, aguçamento ou detecção de bordas
CPU	<i>Central Processing Unit</i>
CSI	<i>Camera Serial Interface</i>
CUDA	<i>Compute Unified Device Architecture</i>
Debounce	Técnica de software ou hardware para eliminar os múltiplos pulsos espúrios (ruído mecânico) gerados quando um botão ou chave mecânica é pressionado ou liberado
Drift Térmico	Variação gradual nas leituras de um sensor causada por mudanças de temperatura, resultando em erro sistemático que aumenta ao longo do tempo de operação
EEPROM	<i>Electrically Erasable Programmable Read-Only Memory</i>
Encoder Rotacional	Sensor eletromecânico que converte a posição angular ou movimento rotacional de um eixo em sinais elétricos digitais, permitindo medir posição, velocidade e direção de rotação
Force Feedback	Sistema de resposta tátil que aplica forças físicas ao dispositivo de controle (volante, joystick) para simular sensações reais como resistência da direção, vibrações do terreno e forças G. Tradução: retroalimentação de força

Força G	Unidade de aceleração equivalente à aceleração gravitacional terrestre ( $9,81 \text{ m/s}^2$ ). Utilizada para expressar acelerações experimentadas por veículos e pilotos em curvas, frenagens e acelerações
FPS	<i>Frames Per Second</i>
Full-duplex	Modo de comunicação bidirecional simultânea, onde dados podem ser transmitidos e recebidos ao mesmo tempo pelo mesmo canal de comunicação
GPIO	<i>General Purpose Input/Output</i>
GPU	<i>Graphics Processing Unit</i>
I2C	<i>Inter-Integrated Circuit</i>
IMU	<i>Inertial Measurement Unit</i>
Jitter	Variação no tempo de chegada de pacotes de dados em uma rede, medida como o desvio padrão da latência. Alto jitter causa irregularidade na recepção de dados em aplicações de tempo real
Latência	Tempo decorrido entre o envio de uma mensagem e sua recepção no destino. Em sistemas de tempo real, latências baixas são essenciais para controle responsivo
LiPo	<i>Lithium Polymer</i>
MJPEG	<i>Motion JPEG</i>
Operador Laplaciano	Operador de segunda derivada utilizado em processamento de imagens para detecção de bordas e aguçamento, calculando a soma das segundas derivadas parciais em relação a x e y
PDI	Processamento Digital de Imagens
PLA	<i>Polylactic Acid</i>
Ponte H	Círcuito eletrônico que permite controlar a direção de rotação de um motor DC através da inversão da polaridade da tensão aplicada, utilizando quatro transistores ou MOSFETs dispostos em configuração H
PPR	<i>Pulses Per Revolution</i>
PWM	<i>Pulse Width Modulation</i>
Quadratura	Método de codificação de encoders rotativos que utiliza dois canais (A e B) defasados em $90^\circ$ elétricos, permitindo determinar tanto a posição quanto a direção de rotação

RAM	<i>Random Access Memory</i>
RSSI	<i>Received Signal Strength Indicator</i>
Serialização	Processo de conversão de estruturas de dados ou objetos em um formato que pode ser armazenado em arquivo ou transmitido pela rede, permitindo posterior reconstrução (dessa serialização)
TCP	<i>Transmission Control Protocol</i>
Teleoperação	Operação de um veículo, robô ou sistema à distância, onde o operador controla o equipamento remotamente através de interfaces de comando e recebe feedback visual e/ou tátil em tempo real
Thread-safe	Característica de código ou estrutura de dados que pode ser acessado simultaneamente por múltiplas threads de execução sem causar condições de corrida ou corrupção de dados
UDP	<i>User Datagram Protocol</i>
USB	<i>Universal Serial Bus</i>

## APÊNDICE A – DIAGRAMA ELÉTRICO DO RASPBERRY PI

Este apêndice apresenta o diagrama elétrico completo das conexões do Raspberry Pi 4 com os demais componentes do sistema: sensor BMI160, driver PCA9685, ADC ADS1115, sensor de corrente INA219, ponte H BTS7960 e demais periféricos. O diagrama está dividido em cinco partes para melhor visualização.

### Pinagem do Raspberry Pi 4

A Tabela 21 apresenta a pinagem completa do Raspberry Pi 4 utilizada no veículo teleoperado.

Tabela 21 – Mapeamento de pinos GPIO do Raspberry Pi 4 no veículo

GPIO	Componente	Sinal	Observação
2	Barramento I2C	SDA	Compartilhado: BMI160, PCA9685, ADS1115, INA219
3	Barramento I2C	SCL	Compartilhado: BMI160, PCA9685, ADS1115, INA219
4	DS18B20	DATA	1-Wire com pull-up 4,7kΩ
18	BTS7960	RPWM	PWM motor horário
27	BTS7960	LPWM	PWM motor anti-horário
22	BTS7960	R_EN	Enable lado direito
23	BTS7960	L_EN	Enable lado esquerdo
CSI	Câmera OV5647	Flat cable	Interface Camera Serial Interface

Fonte: elaborado pelo autor.

A Tabela 22 apresenta os endereços I2C dos dispositivos conectados ao barramento compartilhado.

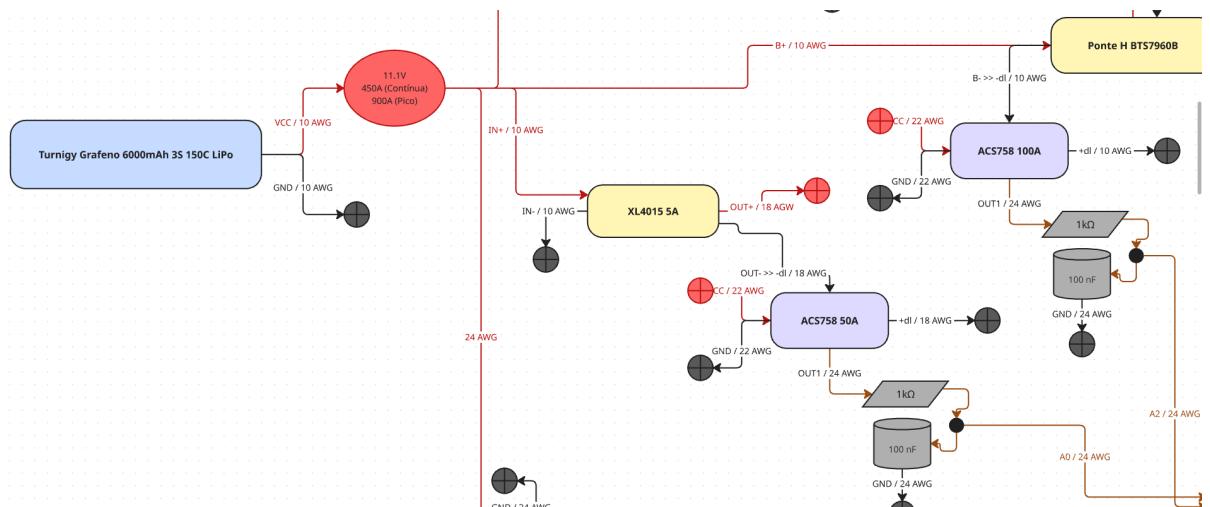
Tabela 22 – Endereços I2C dos dispositivos no veículo

Endereço	Dispositivo	Função
0x40	PCA9685	Driver PWM 16 canais para servos
0x41	INA219	Sensor de corrente/tensão do Raspberry Pi (A0=VCC)
0x48	ADS1115	ADC 16 bits para sensores ACS758
0x68	BMI160	IMU 6 eixos (acelerômetro/giroscópio)

Fonte: elaborado pelo autor.

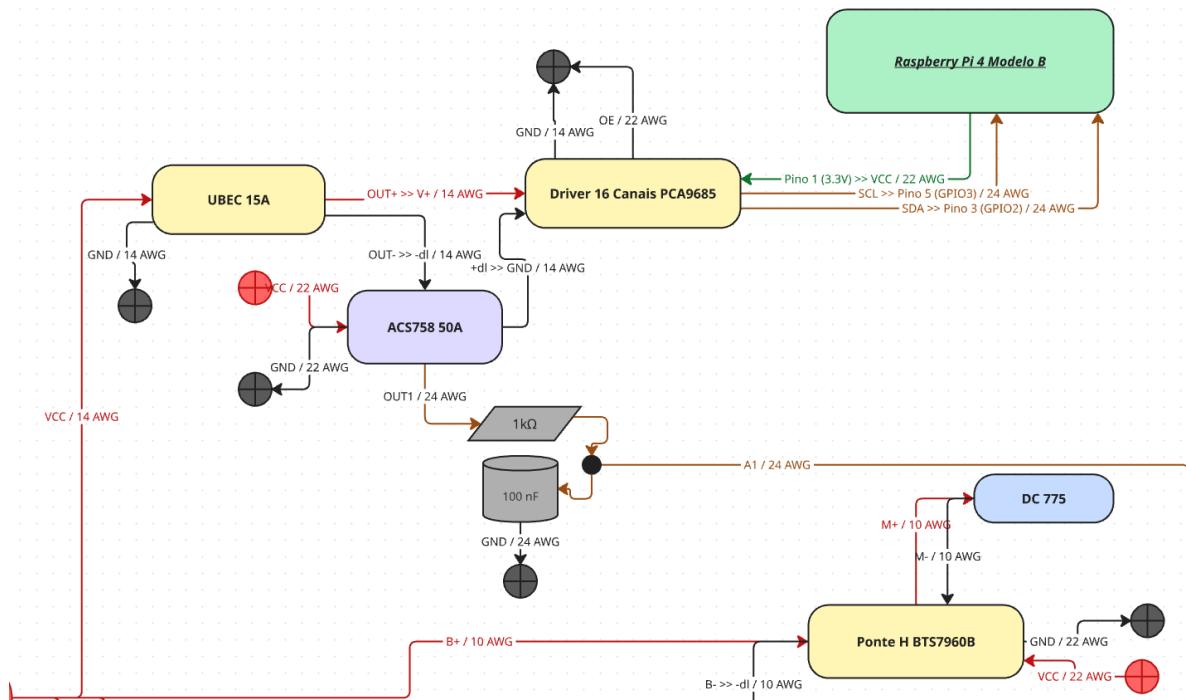
## **Parte 1 - Alimentação Principal e Sensores de Corrente**

Seção do diagrama mostrando a bateria LiPo Turnigy Grafeno 6000mAh 3S 150C, regulador step-down XL4015 5A, sensores de corrente ACS758 (50A e 100A) com filtros RC (1kΩ e 100nF), e conexão com a ponte H BTS7960B.



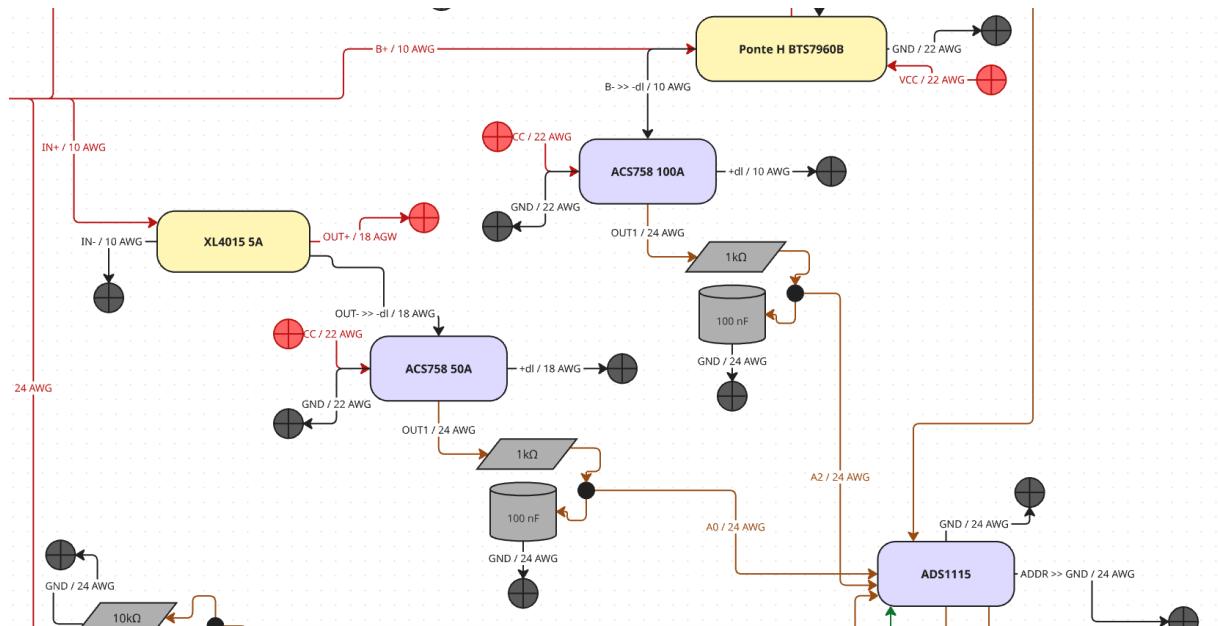
## Parte 2 - UBEC, Driver PWM e Motor de Propulsão

Seção do diagrama mostrando o UBEC 15A para alimentação dos servos, driver PWM PCA9685 de 16 canais conectado ao Raspberry Pi 4 via I2C (GPIO2/GPIO3), sensor de corrente ACS758 50A para monitoramento dos servos, motor DC 775 e ponte H BTS7960B.



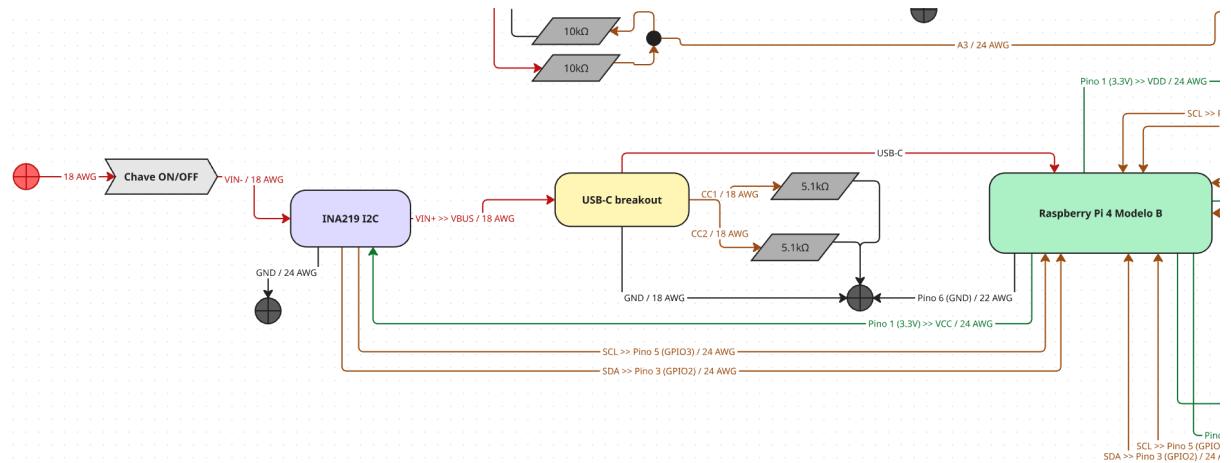
### Parte 3 - ADC e Monitoramento de Corrente

Seção do diagrama mostrando a ponte H BTS7960B, sensores de corrente ACS758 (100A para motor e 50A para Raspberry Pi), regulador XL4015, ADC ADS1115 de 16 bits conectado via I2C, e filtros RC para condicionamento dos sinais analógicos.



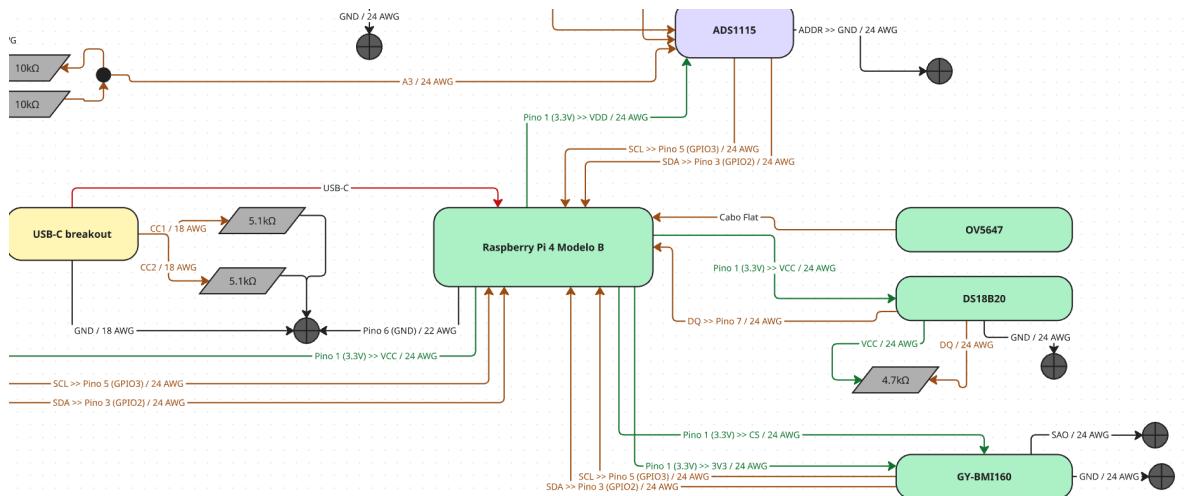
## Parte 4 - Alimentação USB-C e Sensor INA219

Seção do diagrama mostrando a chave ON/OFF, sensor de corrente e tensão INA219 I2C, módulo USB-C breakout com resistores de  $5.1\text{k}\Omega$  para negociação de energia, divisores de tensão com resistores de  $10\text{k}\Omega$ , e conexão de alimentação com o Raspberry Pi 4.



## Parte 5 - Raspberry Pi e Sensores

Seção do diagrama mostrando o Raspberry Pi 4 Modelo B como elemento central, ADC ADS1115 conectado via I2C, câmera OV5647 conectada via cabo flat CSI, sensor de temperatura DS18B20 com resistor pull-up de  $4,7\text{k}\Omega$ , sensor IMU GY-BMI160 conectado via I2C, e módulo USB-C breakout.



## APÊNDICE B – DIAGRAMA ELÉTRICO DO SIMULADOR (ESP32)

Este apêndice apresenta o diagrama elétrico completo das conexões do ESP32 DevKit V1 com os demais componentes do simulador: encoders rotacionais LPD3806-600BM, ponte H BTS7960 para force feedback, botões de troca de marcha, e sistema de alimentação via fonte ATX com placa breakout.

### Pinagem do ESP32 DevKit V1

A Tabela 23 apresenta a pinagem completa do ESP32 DevKit V1 utilizada no simulador.

Tabela 23 – Mapeamento de pinos GPIO do ESP32 no simulador

GPIO	Componente	Sinal	Observação
25	Encoder Acelerador	CLK	Pinos invertidos para correção de direção
26	Encoder Acelerador	DT	—
27	Encoder Freio	CLK	—
14	Encoder Freio	DT	—
12	Encoder Direção	CLK	Pinos invertidos para correção de direção
13	Encoder Direção	DT	—
32	Botão Marcha	UP	Pull-up interno ativado
33	Botão Marcha	DOWN	Pull-up interno ativado
16	BTS7960	RPWM	PWM force feedback horário
17	BTS7960	LPWM	PWM force feedback anti-horário
18	BTS7960	R_EN	Enable lado direito
19	BTS7960	L_EN	Enable lado esquerdo

Fonte: elaborado pelo autor.

### Diagrama de Conexões