INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE MINAS GERAIS - *CAMPUS* BETIM
BACHARELADO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Mariana Gomes Brandão Vidotti

DEFINIÇÃO DE TESTES AUTOMÁTICOS PARA A DETECÇÃO DE PROBLEMAS DE "TELA PRETA" NA VALIDAÇÃO DE *HEAD UNITS*

Betim
2026

Mariana Gomes Brandão Vidotti

DEFINIÇÃO DE TESTES AUTOMÁTICOS PARA A DETECÇÃO DE PROBLEMAS DE "TELA PRETA" NA VALIDAÇÃO DE *HEAD UNITS*

Trabalho de Conclusão de Curso apresentado à banca examinadora do curso de Engenharia de Controle e Automação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais, Campus Betim, como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Orientador: Prof. Arthur Hermano Rezende

Betim
2026

FICHA CATALOGRÁFICA

V654d Vidotti, Mariana Gomes Brandão

Definição de testes automáticos para a detecção de problemas de "tela preta" na validação de head units / Mariana Gomes Brandão Vidotti – 2026.

63 f. : il.

Trabalho de conclusão de curso (Bacharelado em Engenharia de Controle e Automação) - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais, Câmpus Betim, 2026.

Orientação: Prof. Dr. Arthur Hermano Rezende Rosa

1. Sistemas automotivos. 2. Visão computacional. 3. Localização de falhas (Engenharia). 4. Arduino (Controlador programável) I. 5. Engenharia de Controle e Automação. I. Vidotti, Mariana Gomes Brandão. II. Título.

CDU: 681.51

Ficha catalográfica elaborada pelo Bibliotecário Denísio Pereira Marcos CRB-6/3142

INSTITUTO FEDERAL MINAS GERAIS
Campus Betim - Código INEP: 31000576
Rua Itamarati, nº. 140, Bairro São Caetano, CEP: 32677-564, Betim/MG
CNPJ: 10.626.896/0008-49 - Telefone: (31) 3597-6360

ATA DE DEFESA DE RELATÓRIO DE PROJETO

Na presente data realizou-se a sessão pública de defesa do Relatório de Projeto intitulada **Definição de testes automáticos para a detecção de problemas de "tela preta" na validação de head units** apresentada pela aluna **Mariana Gomes Brandão Vidotti (0056235)** do Curso **Bacharelado em Engenharia de Controle e Automação (Campus Betim)**. Os trabalhos foram iniciados às **18:00** do dia **29/01/2026** pelo Professor presidente da banca examinadora, constituída pelos seguintes membros:

- **Arthur Hermano Rezende Rosa** (Orientador)
- **Gustavo Lobato Campos** (Examinador Interno)
- **Marcus Paulo Silva de Abreu** (Examinador Interno)

A banca examinadora, tendo terminado a apresentação do conteúdo do Relatório de Projeto, passou à argüição da candidata. Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelo aluno, tendo sido atribuído o seguinte resultado:

[X] Aprovado [ ] Reprovado Nota (quando exigido): _____

Observação / Apreciações:

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu **Arthur Hermano Rezende Rosa** lavrei a presente ata que assino juntamente com os demais membros da banca examinadora.

BETIM / MG, 29/01/2026

gov.br Documento assinado digitalmente MARCUS PAULO SILVA DE ABREU Data: 30/01/2026 20:13:45-0300 Verifique em https://validar.iti.gov.br

gov.br Documento assinado digitalmente ARTHUR HERMANO REZENDE ROSA Data: 30/01/2026 17:27:09-0300 Verifique em https://validar.iti.gov.br

Marcus Paulo Silva de Abreu

Arthur Hermano Rezende Rosa

Gustavo Lobato Campos

gov.br Documento assinado digitalmente GUSTAVO LOBATO CAMPOS Data: 30/01/2026 20:46:47-0300 Verifique em https://validar.iti.gov.br

AGRADECIMENTOS

Gostaria de agradecer, primeiramente, a Deus, pois creio que, se não fosse sua infinita bondade e misericórdia, eu não estaria onde estou hoje.

Aos meus pais, Leidiane Gomes e Vinicius Vidotti, que sempre foram minha base e meu suporte durante toda a minha jornada acadêmica e ao longo da vida. Obrigada por investirem na minha educação, por todo apoio, paciência e por nunca me deixarem desistir, mesmo diante das dificuldades. Vocês sempre tiveram palavras sábias para me consolar e me encorajar a seguir em frente.

Ao meu noivo, Saulo Carvalho, por toda paciência nos finais de semana em que precisei estudar e por estar ao meu lado em todos os momentos. Obrigada por me apoiar, me incentivar e até mesmo por se sentar ao meu lado para me ajudar a estudar matérias que você ainda lembrava da sua época de faculdade. Você é meu companheiro de vida, e seu apoio aos meus estudos é uma grande prova de amor. Sei que você deseja que eu tenha um futuro brilhante, e isso me fortalece todos os dias.

A todos os professores que tive o imenso prazer de ser aluna e aprender ao longo dessa trajetória, levo comigo o nome de cada um com profunda gratidão.

Por fim, aos meus amigos, que tornaram esse período mais leve e descontraído. Obrigada pelas horas de estudo, pelas conversas nos intervalos e por compartilharem comigo essa caminhada.

RESUMO

Este trabalho apresenta uma abordagem para a automação de testes voltada à validação de sistemas de *infotainment* automotivos, com foco na detecção da falha crítica conhecida como “tela preta” em centrais multimídia (*Head Units*). A proposta busca substituir processos manuais, tradicionalmente onerosos e suscetíveis a erros, por soluções automatizadas que garantam maior robustez, repetibilidade e escalabilidade no ciclo de validação. Para isso, foram desenvolvidos três cenários complementares: **(i)** verificação automática da imagem da câmera de ré, utilizando técnicas de visão computacional com Python e OpenCV integradas ao ambiente CANalyzer via CAPL; **(ii)** monitoramento da inicialização da Head Unit por meio de ciclos simulados de ignição, empregando Arduino, relés e sensores de luminosidade para detecção precisa de falhas intermitentes; e **(iii)** validação avançada com desligamento automatizado, incorporando o braço robótico Dobot Magician para interação física com a interface da central. A metodologia inclui a construção de uma bancada de testes modular, integrando hardware e software, capaz de executar milhares de ciclos contínuos sem intervenção humana. Os resultados demonstraram alta eficiência na detecção de falhas visuais, redução expressiva do tempo de validação e aumento da confiabilidade do processo, evidenciando o potencial da automação como ferramenta estratégica para elevar os padrões de qualidade em sistemas embarcados automotivos.

Palavras-chave: validação automotiva; *Head Unit*; tela preta; sistema de *infotainment*; automação de testes; bancada de testes; Arduino IDE; OpenCV; CAPL; Dobot Magician.

ABSTRACT

This study presents an innovative approach to automated testing aimed at validating automotive infotainment systems, focusing on detecting the critical failure known as the “black screen” in multimedia Head Units. The proposed solution replaces manual processes, traditionally costly and error-prone, with automated methods that ensure greater robustness, repeatability, and scalability throughout the validation cycle. Three complementary scenarios were developed: ***(i)*** *automatic verification of the rear camera image using computer vision techniques with Python and OpenCV integrated into the CANalyzer environment via CAPL;* ***(ii)*** *monitoring Head Unit startup through simulated ignition cycles, employing Arduino, relays, and light sensors for accurate detection of intermittent failures; and* ***(iii)*** *advanced validation with automated shutdown, incorporating the Dobot Magician robotic arm for physical interaction with the system interface. The methodology includes building a modular test bench that integrates hardware and software, capable of executing thousands of continuous cycles without human intervention. Results demonstrated high efficiency in detecting visual failures, significant reduction in validation time, and improved process reliability, highlighting the potential of automation as a strategic tool to raise quality standards in automotive embedded systems.*

Keywords: *automotive validation; Head Unit; black screen; infotainment system; test automation; test bench; Arduino IDE; OpenCV; CAPL; Dobot Magician.*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A complexidade das funções embarcadas nos veículos modernos tem evoluído de maneira expressiva, impulsionada pelo avanço contínuo dos sistemas eletrônicos e de conectividade. Com a crescente adoção de veículos elétricos e autônomos, os automóveis passaram a incorporar arquiteturas eletrônicas centralizadas e robustas, capazes de suportar funções críticas como atualizações remotas *Over-The-Air* (OTA), assistência avançada à condução e integração com plataformas digitais em nuvem. Nesse cenário, os sistemas de *infotainment* assumem papel de destaque como principal interface entre motorista e veículo, agregando funcionalidades como navegação, espelhamento de smartphones, controle de mídia e monitoramento por câmeras. Essa transformação tecnológica, além de ampliar a conveniência e segurança, impõe novos desafios à confiabilidade e à validação desses sistemas complexos (BANDUR et al., 2021; EBERT; FAVARO, 2017).

Para acompanhar essa evolução, muitas montadoras e fornecedores têm defendido a tendência de arquiteturas veiculares centralizadas, impulsionada pela necessidade de maior eficiência, integração de sistemas avançados e redução da complexidade, especialmente nos custos de manutenção (BANDUR et al., 2021).

Dentro das arquiteturas centralizadas ou descentralizadas, existem as Unidades de Controle Eletrônicas (ECUs), que são os módulos responsáveis pelo controle das funções dos veículos, como as centrais eletrônicas de chassis, *Body*, *infotainment*, *Advanced Driver Assistance Systems* (ADAS), Conectividade e Telemática. Essas ECUs podem operar de maneira independente e descentralizada, onde cada unidade possui sua própria plataforma de hardware e software, ou de maneira centralizada, onde múltiplas ECUs compartilham a mesma plataforma de hardware e software (MODY et al., 2018).

O desenvolvimento de software nessas centrais eletrônicas é uma atividade bastante complexa que segue o modelo V de desenvolvimento, aderindo às boas práticas automotivas, tais como, análise de requisitos, desenvolvimento, testes para verificação, validação e manutenção do software (TIERNO et al., 2016).

Dessa forma, como estabelecido pelo modelo V, uma das etapas cruciais no desenvolvimento de software são os testes, pois estes visam garantir que o sistema funcione corretamente, de forma segura, eficiente e com qualidade. Embora testar sistematicamente o software seja importante para garantir a qualidade, testes manuais

são extremamente caros, suscetíveis a erros e demorados. Por sua vez, testes automáticos possuem maior eficiência, precisão e abrangência no processo de validação, maior cobertura de testes, detecção precoce de defeitos e melhor custo-efetividade (TIERNO et al., 2016).

Assim, a automação de testes tem como objetivo aprimorar a qualidade, robustez e confiabilidade das ECUs, acelerando o processo de desenvolvimento. Isso permite que montadoras e fornecedores de automóveis entreguem produtos mais seguros e inovadores de forma ágil ao mercado, atendendo às crescentes demandas por eficiência e inovação no setor automotivo.

1.1 Justificativa

Para mitigar o impacto de lançar produtos com maior robustez e menos suscetíveis a falhas, as grandes montadoras têm investido em tecnologias e metodologias para tornar o processo de validação mais robusto, visando melhorar a qualidade e confiabilidade do software embarcado em suas ECUs (Unidades de Controle Eletrônico).

Uma das formas de tornar mais robusto o processo de validação é através da realização de testes automáticos. Este trabalho visa propor testes automáticos a serem utilizados na validação de uma central multimídia, garantindo uma detecção antecipada de falhas e reduzindo o tempo de desenvolvimento e os custos associados a correções tardias.

A automação dos testes possibilita uma cobertura mais ampla e consistente, assegurando que diferentes cenários e condições de uso sejam avaliados de forma sistemática. Essa abordagem é crucial para assegurar que a implementação de novas funcionalidades não introduza regressões ou falhas em partes do sistema já validadas, promovendo a execução eficiente e contínua dos testes de validação.

Em complemento, a utilização de testes automáticos contribui para a padronização do processo de validação, permitindo que diferentes equipes e projetos sigam os mesmos critérios de qualidade. Isso melhora a comunicação e a colaboração entre as equipes, assegurando que os produtos finais atendam aos mesmos padrões elevados de qualidade e confiabilidade.

Por fim, a adoção de testes automáticos é uma resposta às demandas do mercado por produtos mais seguros e confiáveis. Em um setor altamente competitivo como o automotivo, a capacidade de entregar produtos de alta qualidade pode ser um

diferencial significativo, aumentando a satisfação do cliente e a reputação da marca.

1.2 Problema

Com o avanço tecnológico, aumentaram também os requisitos de desempenho e confiabilidade desses sistemas. As centrais multimídia, conhecidas como *Head Units*, passaram a representar um elemento crítico para a usabilidade e a satisfação do cliente. No entanto, falhas recorrentes, como travamentos e o fenômeno da "tela preta", têm se mostrado persistentes. Essa falha, que impede a exibição de qualquer conteúdo na tela do sistema, pode comprometer funções essenciais, como a visualização da câmera de ré, gerando insegurança, insatisfação e, em casos extremos, levando até à recompra do veículo pela montadora (J.D. POWER, 2023).

Além dos impactos diretos na experiência do usuário, essas falhas afetam diretamente os indicadores de qualidade e garantia dos fabricantes, exigindo respostas mais eficientes. Para enfrentar tais desafios, a indústria automotiva tem aderido cada vez mais a normas internacionais rigorosas, como a ISO 26262, que estabelece diretrizes para garantir a segurança funcional de sistemas eletrônicos veiculares. Essas normas demandam que os testes sejam realizados de forma estruturada, padronizada e repetível — características que podem ser alcançadas com a automação de testes (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2018).

Nesse contexto, a automação do processo de validação surge como uma alternativa eficiente para detectar falhas de forma antecipada e sistemática. Diversas abordagens têm demonstrado que é possível construir soluções eficazes utilizando ferramentas acessíveis, como Arduino, sensores de luminosidade, câmeras, e bibliotecas de visão computacional como a OpenCV, integradas a ambientes de simulação veicular como o CANalyzer. Essas soluções são capazes de simular interações reais do usuário e capturar comportamentos anômalos do sistema, como a tela preta, com elevado grau de precisão (DA SILVA et al., 2020).

Portanto, como estabelecido pelo modelo em V, uma das etapas cruciais no desenvolvimento de software são os testes, pois estes visam garantir que o sistema funcione corretamente, de forma segura, eficiente e com qualidade. Embora testar sistematicamente o software seja importante para garantir a qualidade, testes manuais são frequentemente onerosos, suscetíveis a erros humanos e demandam tempo excessivo. Por sua vez, testes automáticos possuem maior eficiência, precisão e

abrangência no processo de teste, maior cobertura de testes, detecção precoce de defeitos e melhor custo-efetividade.

1.3 Objetivo Geral

O presente trabalho teve como objetivo principal o desenvolvimento de soluções automatizadas para a detecção de falhas do tipo “tela preta” durante o processo de validação de *Head Units*. Mediante a construção de uma bancada de testes composta por sensores, relés, Arduino e integração com ferramentas de programação como Python e CAPL, foi possível propor, implementar e validar três cenários distintos, os quais apresentaram variação no grau de automação e complexidade.

1.4 Objetivos específicos

Como objetivos específicos, temos:

- Identificar cenários de falha: mapear e documentar cenários e condições que podem levar ao problema “tela preta”;
- Implementar testes automáticos: desenvolver e implementar testes automatizados que visem capturar problemas de “tela preta”;
- Validar o teste: por meio do teste proposto, simular condições de falha para avaliar a eficácia do método desenvolvido.

1.5 Organização do trabalho

O Capítulo 1, Introdução, consiste na delimitação do tema, contextualização do problema, justificativa e objetivos. Neste capítulo será apresentada uma visão geral do tema abordado, destacando a relevância do problema e justificando a realização desse estudo. Além disso, serão definidos os objetivos gerais e específicos que guiarão a pesquisa.

No Capítulo 2, Fundamentação Teórica, será realizada uma análise detalhada de estudos anteriores que abordam problemas semelhantes. A revisão da literatura permitirá identificar as melhores abordagens, métodos e tecnologias disponíveis para resolver o problema em questão. Esta seção apresenta a base teórica que fundamenta o desenvolvimento do trabalho.

No Capítulo 3, Metodologia, serão descritos os recursos utilizados e os

procedimentos adotados na elaboração de testes automáticos com o objetivo de detectar o problema de “Tela Preta” durante a inicialização da *Head Unit*. Serão detalhadas as etapas do processo, incluindo a seleção de ferramentas, o desenvolvimento de códigos, procedimento automatizados, bem como a configuração do ambiente de testes. A metodologia adotada garantirá a replicabilidade e a validade dos resultados obtidos.

O Capítulo 4, Resultado e Discussão, apresentará os resultados obtidos e uma análise crítica dos dados apresentados. Serão discutidos os resultados do estudo, comparando-os com a literatura existente e avaliando a eficácia dos testes automáticos implementados. A discussão permitirá identificar as implicações práticas e teóricas dos resultados.

Por fim, no Capítulo 5, Conclusão, serão relatadas as principais conclusões do estudo e potenciais contribuições futuras. Serão destacadas as implicações dos resultados para a prática e a teoria, bem como sugestões para pesquisas futuras que aprofundem o conhecimento sobre o tema.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Projetos similares

A validação de sistemas de *infotainment* (centrais multimídia) por meio de testes automatizados tem sido tema recorrente em pesquisas acadêmicas e iniciativas industriais. Isso se deve à crescente complexidade e importância desses sistemas nos veículos modernos. A automação tem se mostrado crucial para aumentar a confiabilidade, rastreabilidade e repetibilidade nos processos de validação.

Na literatura, é possível encontrar alguns trabalhos que propõem o desenvolvimento de um sistema de teste automatizado para sistemas de infotainment. Segundo Huang et al. (2010, p. 235), os testes automatizados são essenciais para garantir a qualidade e robustez desses sistemas. Além disso, em seu trabalho, os autores utilizam exemplos com HIL (*Hardware in the Loop*) para simular sinais da rede veicular CAN e comunicação serial com outros componentes, como câmeras. Eles também utilizam outras ferramentas, como o simulador dSPACE, para simular determinadas frequências na saída de áudio da central multimídia, bem como algoritmos de reconhecimento de imagem para fazer a detecção de cores na interface HMI do sistema de infotainment.

Lunetta (2021), em sua dissertação de mestrado, descreveu uma abordagem prática que utiliza braços robóticos e reconhecimento de imagem para automatizar testes em sistemas de infotainment. O estudo também integra técnicas de *machine learning*, mostrando como algoritmos inteligentes podem contribuir para a identificação de falhas visuais e operacionais durante o uso da interface da *Head Unit*.

TSAI, Pu-Sheng et al. (2021) em seu artigo descrevem um teste automático baseado em um braço robótico integrado por um Raspberry Pi programado em Python para determinar os pixels de contorno de um objeto utilizando uma câmera para capturar a imagem.

Sini et al. (2018) propõem uma estratégia avançada de automação baseada em HIL *(Hardware in the Loop*) para executar mais de 1.200 casos de teste em apenas duas horas. Em contrapartida, a realização manual dos mesmos testes demandaria cerca de 45 horas. O modelo apresentado integra sinais dinâmicos simulados com ferramentas de análise, permitindo a avaliação do comportamento de sistemas críticos do veículo.

Complementando essa linha de pesquisa, Solimene (2020) desenvolveu um

sistema de automação de baixo custo voltado à validação de painéis de instrumentos (*Intrument Panel Cluster* - IPC). A arquitetura proposta pelo autor utiliza câmeras, CAPL, Arduino e ambientes de programação como PyCharm, o autor propôs uma estrutura eficiente para detectar falhas visuais recorrentes em displays automotivos. A confiabilidade obtida com o reconhecimento de imagem foi um dos destaques da abordagem apresentada.

Estudos mais recentes, como o de DA SILVA et al. (2020), reforçam o potencial de *frameworks* baseados em plataformas abertas. Nesse caso, foram utilizados sensores, Arduino e a biblioteca OpenCV para detectar falhas visuais, como a ausência de imagem esperada durante o acionamento da câmera de ré, validando o funcionamento da *Head Unit* por comparação com imagens de referência.

Em paralelo ao desenvolvimento acadêmico, empresas do setor automotivo também têm investido em soluções de validação automatizada. A Bosch, por exemplo, desenvolveu uma bancada de testes industriais que integra câmeras, sensores de toque e *scripts* automatizados para simular ciclos de ignição e detectar falhas visuais ou funcionais em *Head Units*. Além disso, a coleta automática de vídeos, logs e dados em tempo real permite análises profundas sobre o comportamento do sistema (BOSCH, 2022).

Outro aspecto importante a se considerar é o alinhamento com modelos normativos na indústria. O padrão Automotive SPICE (ASPICE), amplamente adotado por fabricantes automotivos, define níveis de maturidade para processos de desenvolvimento de software embarcado. A automação de testes é um requisito essencial para alcançar níveis avançados de qualidade e rastreabilidade, especialmente em projetos com requisitos de segurança funcional (AUTOMOTIVE SPICE, 2017).

Por fim, Mahfoudhi et al. (2021) destacam os principais desafios enfrentados na automação de testes em ambientes com interfaces gráficas, como é o caso das *Head Units*. O estudo aponta que falhas intermitentes e visuais, como a “tela preta”, não são facilmente detectáveis por testes unitários ou baseados apenas em software. Assim, abordagens híbridas, combinando sensores físicos, reconhecimento visual e controle automatizado, são mais eficazes para identificar esse tipo de problema.

Os trabalhos acima são exemplos de como testes automatizados podem tornar mais robusto a validação de componentes em especial aqueles que fazem parte de sistemas de *infotainment.* Portanto, este trabalho visa propor uma arquitetura de testes

automatizados para a detecção do fenômeno de "Tela Preta" no display de *Head Units*, usando recursos similares, tais como: CANcase, Arduino, câmera, braço robótico, além de ambientes para a programação como Arduino IDE, PyCharm para Python e CAPL.

2.2 Modelo V

O universo da tecnologia automotiva e TI têm mudado rapidamente. Sistemas de *infotainmen*t e conectividade têm transformado os veículos em sistemas distribuídos de TI com acesso a nuvem, atualizações de software *over-the-air*, alta largura de banda para serviços conectados e navegação embarcada, etc. Como mostrado na Figura 1, os sistemas baseados em software estão em constante evolução e inovação (EBERT; FAVARO, 2017).

Figura 1: Evolução dos sistemas automotivos ao longo dos anos.

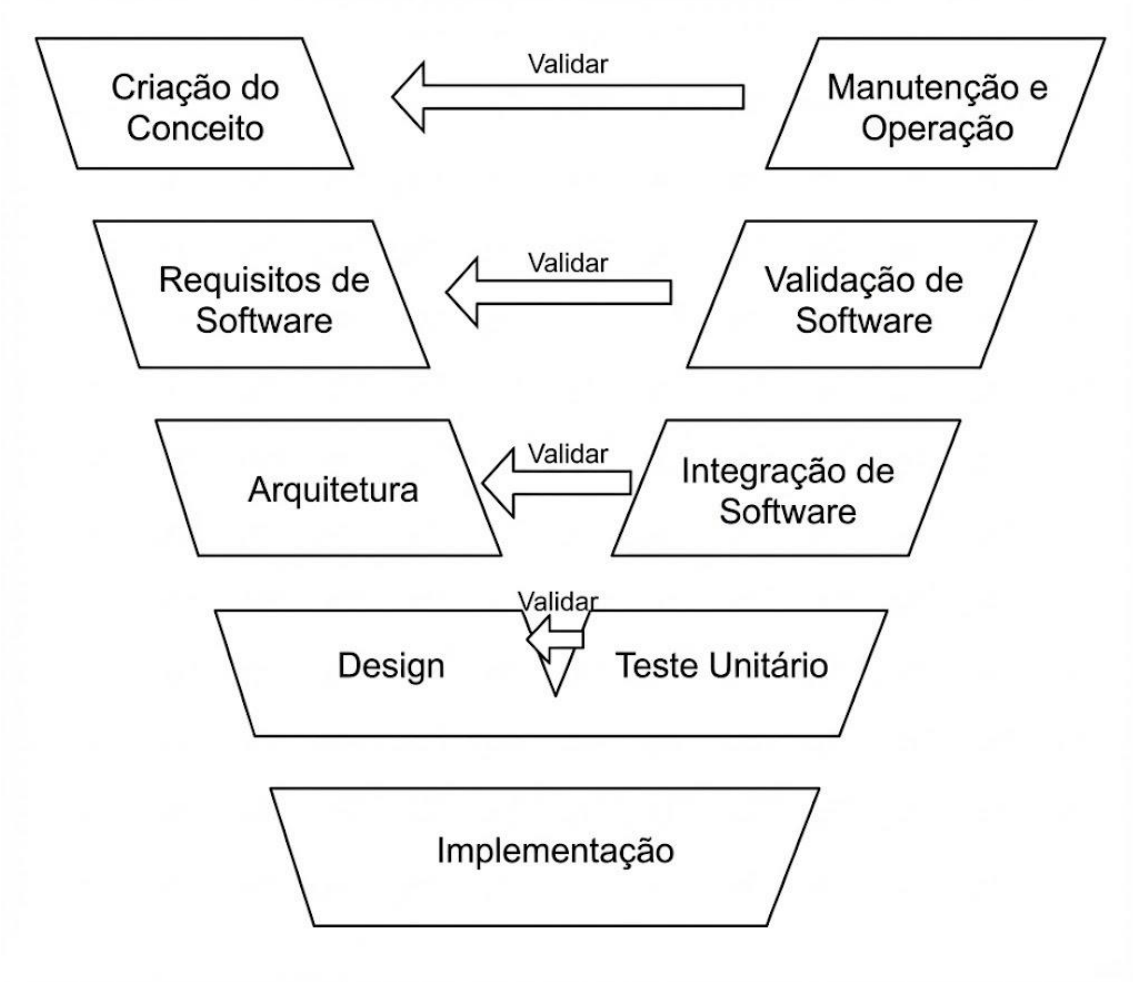
Fonte: TECHNOLOGIES (2021).

Essa rápida evolução é o que permitiu que o desenvolvimento clássico de sistemas embarcados automotivos evoluísse para sistemas distribuídos de TI. Para que os sistemas evoluam de maneira contínua, o modelo V é utilizado para padronizar o desenvolvimento de software no mundo automotivo (EBERT; FAVARO, 2017).

O modelo V representa um processo de desenvolvimento sequencial com validação paralela. O processo inicia-se com a criação do conceito, a definição dos requisitos funcionais e não funcionais, a arquitetura e o design. Em seguida, são realizadas a

implementação, os testes unitários, a integração do software, a validação e a manutenção, conforme mostrado na Figura 2 (NICOLAESCU et al., 2017).

Figura 2: Modelo V de desenvolvimento de software.



Fonte: NICOLAESCU et al. (2017).

A estrutura do Modelo V parte da definição dos requisitos do sistema, passando pelo design e pela codificação, até chegar à verificação e validação final. No braço direito do “V”, estão os testes de unidade, integração e validação, fundamentais para garantir que o sistema atenda aos requisitos funcionais e de segurança. No entanto, a validação em sistemas modernos, como as *Head Units*, envolve também a análise de falhas visuais e de interface, algo que o modelo tradicional não prevê diretamente.

Com o avanço de paradigmas como *Continuous Integration/Delivery* (CI/CD) e metodologias ágeis/DevOps, o modelo V tem sido adaptado, incorporando ciclos iterativos e testes automatizados em estágios intermediários. Isso permite conciliar a rigidez da rastreabilidade do modelo com a flexibilidade exigida pelas atualizações *over-the-air* e pelos ciclos rápidos de desenvolvimento do setor automotivo moderno

(PRETSCHNER et al., 2020).

Além disso, normas como a ISO 26262, voltada à segurança funcional de sistemas elétricos e eletrônicos em veículos rodoviários, reforçam a necessidade de validação formal dos requisitos desde as fases iniciais do projeto, adotando o modelo V como estrutura de referência. Essa norma define categorias de criticidade, denominadas *Automotive Safety Integrity Levels* (ASILs), que exigem níveis crescentes de evidência formal de teste, contemplados na fase de validação do modelo V (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2018).

Por fim, embora o modelo V ofereça solidez em termos de estrutura e rastreabilidade, autores como Mahfoudhi et al. (2021) destacam que sua aplicação rígida pode ser limitante em sistemas altamente iterativos e orientados à interface gráfica, como os de *infotainment*. Por isso, há uma tendência crescente de utilização de modelos híbridos, que mantêm a base conceitual do modelo V, mas incorporam automações de testes, testes guiados por sensores físicos e ferramentas de validação inteligente para lidar com falhas complexas, como a tela preta.

A etapa de validação dos requisitos é uma das mais críticas no desenvolvimento de software, conforme o modelo em V, pois assegura que o sistema implementado esteja de acordo com o que foi originalmente especificado. Nesse contexto, o presente trabalho tem como objetivo aprimorar essa etapa por meio da definição e execução automática de casos de teste, contribuindo para uma validação mais eficiente, confiável e repetível para o problema tela preta.

2.3 Arduino

O Arduino é uma plataforma de prototipagem eletrônica de codigo aberto. Ele consiste em um hardware, um software e uma placa de microcontrolador com a finalidade de permitir ao usuário o gerenciamento de dispositivos como sensores, motores, sistemas de supervisão, sistemas de iluminação, entre outros componentes que simplificam a conexão e programação com outros circuitos (BITTENCOURT, 2017).

A vantagem de se utilizar o Arduino é que se trata de um ambiente que pode ser executado em Windows, iOS e Linux, tem por base a IDE de programação Processing, pode ser programado utilizando um cabo USB, não precisando de porta serial, possui hardware e software de fonte aberta, conta com uma comunidade muito ativa de usuários e foi desenvolvido em um ambiente educacional, o que o faz ideal

para iniciantes (BANZI; SHILOH, 2011).

No contexto da engenharia automotiva, o Arduino tem sido amplamente utilizado em aplicações de validação de sistemas embarcados. Sua flexibilidade permite a criação de dispositivos de testes personalizados, como simuladores de botões físicos, controle de LEDs, leitores de sensores analógicos ou digitais e módulos para monitoramento de tensão e luminosidade. Em testes de *Head Units*, por exemplo, é possível integrar o Arduino a sensores de luminosidade (LDR) para detectar falhas visuais como a “tela preta”, além de utilizá-lo para acionar comandos automáticos em interfaces físicas com apoio de relés ou braços robóticos.

Estudos recentes destacam o uso do Arduino em conjunto com bibliotecas como OpenCV para validar a resposta visual de sistemas de interface homem-máquina (IHM) automotivos de forma automatizada, reforçando seu papel como ferramenta de baixo custo e alta eficiência em bancadas experimentais e validações funcionais (DA SILVA et al., 2020; CHO; KIM, 2023). Sua capacidade de integração com ambientes como CAPL e Python permite ampliar o escopo dos testes, simulando condições reais de uso com elevada confiabilidade.

2.4 Robotic Arm – Dobot Magician

O Dobot Magician é um braço robótico versátil com foco em necessidades de educação, pesquisa e aplicações na industria. Ele possui uma interface bem amigável e suporte para várias linguagens de programação, por exemplo: Python e CC++. Precisão e repitibilidade são funcionalidades que esse braço robótico oferece com com excelência (DOBOT, 2024).

Sua versatilidade faz com que ele se adapte facilmente às necessidades específicas de cada ambiente. Dessa forma, o braço robótico torna-se uma ferramenta bastante promissora para automatizar tarefas repetitivas com alta precisão e eficiência (DOBOT, 2024).

No contexto da indústria automotiva, o Dobot Magician tem sido utilizado como solução de baixo custo para automação de testes em sistemas de *infotainment* e painéis de instrumentos. Em estudos recentes, o braço robótico foi integrado a sensores visuais e scripts em Python para simular interações humanas com *displays das Head Units*, validando respostas a comandos físicos e detectando falhas intermitentes, como travamentos ou ausência de imagem (“tela preta”) (TSAI et al., 2021; SOLIMENE, 2020).

2.5 Programação em Python

A linguagem de programação Python é conhecida por sua simplicidade e clareza, sendo uma linguagem poderosa que pode ser usada em grandes projetos. Trata-se de um software livre, o que permite o seu uso gratuito em plataformas como Windows, Linux, MacOs (MENEZES, 2010).

Projetos de teste automatizado em *Head Units* têm utilizado Python em conjunto com câmeras e bibliotecas de visão computacional (como OpenCV) para capturar e interpretar imagens da interface HMI, identificar falhas visuais como a “tela preta” e executar testes comparativos de estados visuais. Essa abordagem permite criar *scripts* modulares, facilmente adaptáveis a diferentes tipos de *displays*, resoluções e padrões de falhas (CHEN et al., 2022).

2.6 Programação em CAPL

O CAPL (Communication Access Programming Language) é uma linguagem de programação semelhante à linguagem C, desenvolvida pela Vector e disponível nos softwares CANoe e CANalyzer. Com CAPL, é possível criar desde casos simples, como a geração e transmissão de estímulos em redes de comunicação, até casos complexos, como análises, simulações e testes de ECUs. A linguagem permite acesso a objetos em bancos de dados (dbc, arxml, fibex, ldf), arquivos de descrição de diagnóstico (CDD/ODX), variáveis de sistema definidas no CANoe/CANalyzer e outros recursos computacionais (VECTOR, 2024).

2.7 Síntese da Fundamentação Teórica

A revisão da literatura realizada neste trabalho permitiu identificar diferentes abordagens para a automação de testes em sistemas automotivos, especialmente voltados para *Head Units* e interfaces gráficas. A seguir, apresenta-se uma síntese das principais contribuições encontradas:

BOSCH (2022) desenvolveu uma bancada industrial para validação automatizada de sistemas de *infotainment,* utilizando câmeras, sensores de toque e scripts automatizados. Essa solução possibilita a simulação de ciclos de ignição e a coleta automática de dados, garantindo maior rastreabilidade e eficiência nos testes;

DA SILVA et al. (2020) propuseram um framework baseado em plataformas abertas, integrando Arduino, sensores e a biblioteca OpenCV para validação visual

por comparação com imagens de referência. A abordagem destaca-se pela simplicidade e baixo custo, sendo aplicável em ambientes experimentais;

HUANG et al. (2010) apresentaram um sistema automatizado de testes para *infotainment* utilizando HIL (*Hardware-in-the-Loop*), dSPACE, CAN e algoritmos de reconhecimento de imagem. Essa solução assegura robustez e qualidade, permitindo simulação de sinais e detecção visual de falhas;

LUNETTA (2021) introduziu a automação com braço robótico e técnicas de *machine learning* para testes em *Head Units*. A proposta possibilita identificar falhas visuais e operacionais, aumentando a confiabilidade do processo de validação;

MAHFOUDHI et al. (2021**)** revisaram os principais desafios na automação de testes com interfaces gráficas, sugerindo abordagens híbridas que combinam sensores físicos, reconhecimento visual e controle automatizado, tornando a detecção de falhas mais eficaz;

SINI et al. (2018) apresentaram uma estratégia avançada baseada em HIL para executar mais de 1.200 casos de teste em apenas duas horas, em contraste com 45 horas necessárias para testes manuais. Essa abordagem contribui para alta eficiência e redução significativa do tempo de validação;

SOLIMENE (2020) desenvolveu um sistema para automação de testes em painéis de instrumentos, utilizando CAPL, Arduino, PyCharm e câmeras. A proposta destaca-se pela detecção eficiente de falhas visuais recorrentes em displays automotivos;

TSAI et al. (2021) criaram um teste automatizado com braço robótico e Raspberry Pi, programado em Python, para análise de pixels e automação física confiável, reforçando a integração entre hardware e software.

Por fim, a proposta desenvolvida neste trabalho apresenta propostas para a automação de testes voltados à detecção do fenômeno conhecido como tela preta em sistemas de *infotainment* automotivos (*Head Units*). A solução integra recursos de hardware e software, combinando plataformas abertas, sensores, e algoritmos utilizando Arduino IDE, CAPL e Python.

Essa proposta amplia a cobertura dos testes, aumenta a robustez no processo de validação, complementado as práticas identificadas na literatura.

A figura 3, representa um resumo a respeito das referências bibliográficas mencionadas.

Figura 3: Referências bibliográficas da fundamentação teórica

AUTOR(ES)	ANO	TEMA/FOCO	TECNOLOGIAS E FERRAMENTAS	PRINCIPAIS RESULTADOS E CONTRIBUIÇÕES
Huang et al.	2010	Qualidade e robustez via HIL e HMI	HIL, dSPACE, Simulação CAN, Reconhecimento de Imagem	Simulação de sinais de rede e frequências de aúdio, detecção de cores na interface.
Automotive SPICE	2017	Modelo Normativo	Processos de desenvolvimento (ASPICE)	Define niveis de maturidade e requisitos de rastroabilidade/segurança funcional.
Sini et al.	2018	Eficiência e velocidade de teste	Estratégia HIL, Ferramentas de análise	Execução de 1200 testes em 2h (vs. 45h manuais); avaliação de sistemas críticos.
Solimene	2020	Automação de baixo custo para IPC	Câmaras, CAPL, Arduino, PyCharm (Python)	Detecção eficiente de falhas visuais recorrentes em painéis de instrumentos.
Da Silva et al.	2020	Frameworks em plataformas abertas	Arduino, OpenCV, Sensores	Validação por comparação com imagem de referência (ex: câmera de ré).
Lunetta	2021	Abordagem prática com Inteligência Artificial	Braços Robóticos, Reconhecimento de Imagem, Machine Learning	Identificação de falhas visuals e operacionais na Head Unit.
TSAI, Pu-Sheng et al.	2021	Determinação de contorno de objetos	Braço Robótico, Raspberry Pi, Python, Câmara	Uso de vieto computacional para determinar pixels de contorno.
Mahfoudhi et al.	2021	Desafios em interfaces gráficas (GUI)	Estudo de abordagens hibridas	Aponta a dificuldade de detectar "tela preta" via software puro, sugere sensoos fisicos + visão.
Bosch	2022	Solução Industrial	Câmaras, Sensoros de toque, Scripts, Logs/Vídeo	Plataforma robusta para simular ciclos de iginção e analzar comportamento profundo do sistema.
Proposta do Trabalho	Atual	Detecção do fenômeno "Tela Preta"	CANcase, Arduino, Cámara, Braço Robótico, CAPL, Python	Proposta de testes robustos combinando hardware e software para falhas intormitentes.

Fonte: Elaborado pelo autor (2025).

3 METODOLOGIA

Esse capítulo descreve a metodologia adotada para o desenvolvimento de testes automatizados para detectar e diagnosticar problemas relacionados à tela preta que podem ser encontrados durante a validação de *Head Units* em sistemas de *infotainment*. A metodologia será dividida em etapas sequenciais, abrangendo desde a identificação de cenários da falha mencionada, recursos, cenários de falha, até a implementação e validação dos testes automáticos propostos.

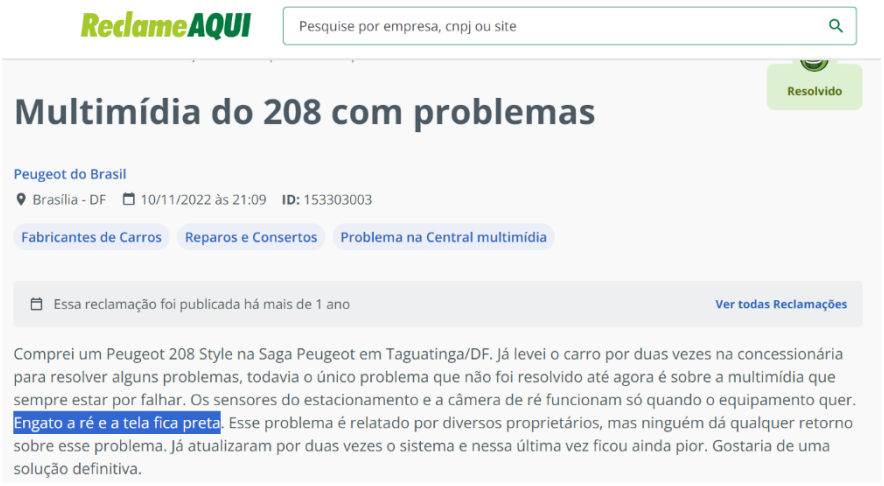
3.1 Definição do problema e condições

A primeira etapa realizada foi uma análise para entender e documentar as principais reclamações de clientes em sistemas de *infotainment* já em produção e lançados ao mercado. Essa etapa envolveu:

- Análise e registro de falhas em *Head Units* informadas por clientes;
- Consulta à equipe de Qualidade e Validação da Empresa S sobre os principais problemas que afetam os seus índices de garantia e satisfação do cliente.

Segundo a Mobiauto (2024) e a Autoesporte (2024), a central multimídia do Peugeot 208 é uma das principais fontes de problemas, com as principais queixas incluindo travamento, tela cinza, tela congelada e tela preta na câmera de ré. A Figura 4, mostra a manifestação da reclamação de um cliente que enfrentou o problema de tela preta ao engatar a marcha ré no veículo.

Figura 4: Verbalização do cliente.



Fonte: PEUGEOT (2022).

Na consulta realizada com o time de Qualidade e Validação da Empresa S, essas reclamações se confirmam. A tela preta é considerada um problema de alta severidade, com impacto direto nos indicadores de confiabilidade e satisfação do cliente, sendo classificada como um problema de estabilidade. A empresa possui um controle para evitar que esse tipo de problema chegue ao consumidor final, mas reconhece que seu processo de validação não é totalmente à prova de falhas; problemas de difícil reprodução ainda escapam às validações tradicionais. Por isso, maneiras de tornar esse processo mais robusto são sempre bem-vindas.

Portanto, com base nos dados acima, ficou definido que o problema a ser tratado neste trabalho seria o de tela preta. Com o fenômeno definido, foi questionado à equipe de Qualidade e Validação sobre algumas condições nas quais o problema se manifestava. O resultado foi que as principais queixas ocorriam no momento da inicialização da central multimídia, quando o veículo é ligado. Outra condição bastante severa ocorre durante o engate da marcha ré, momento em que a tela da *Head Unit* deve exibir a imagem da câmera de ré, caso isso não ocorra, é considerado um problema de alta severidade, por se tratar de uma funcionalidade de segurança. Problemas desse tipo certamente levam o cliente de volta à concessionária para registrar uma reclamação.

Dessa maneira, foram elaborados três cenários de testes automáticos: dois deles voltados para a fase de inicialização da *Head Unit* e um direcionado ao engate da marcha ré, com o objetivo de verificar a ocorrência do evento de “Tela Preta” no display da central multimídia durante essa manobra.

3.2 Recursos

Definidos os cenários de teste acima, a próxima etapa consiste em de detalhar os recursos e procedimentos que compõem os testes automáticos propostos.

3.2.1 Arduino

O Arduino, plataforma de código aberto, foi escolhido devido a sua flexibilidade de uso para simular interações com a *Head Unit*. Essa flexibilidade se dá na liguagem de programação utilizada, baseada em C/C++. Além disso, o seu ambiente de desenvolvimento integrado (IDE) simplifica o uso dessas linguagens, oferecendo uma série de funções especificas, o que torna a programação mais acessível, em especial

para iniciantes.

No teste proposto o Arduino será utilizado para processar a execução do teste através da lógica programada em seu microcontrolador, como ilustrado na Figura 5. Ele será utilizado para controlar as informações que são exibidas no display *shield*, acionar o modulo relé para controlar o botão de partida do veículo, bem como a parametrização e leitura do brilho da tela do display da *head unit* através de um sensor de cor.

Figura 5: : Placa Arduino

Fonte: Wikimedia (2023).

3.2.2 Dobot Magician

O Dobot Magician é um braço robótico de alta precisão, que pode ser utilizado para automatizar interações físicas com a *Head Unit* que seriam difíceis de simular eletronicamente, por exemplo, toques na tela, simulando a seleção de ícones, navegação pelos menus e outras interações táteis na interface da central multimídia. A Figura 6 ilustra o braço robótico utilizado.

Figura 6: Dobot Magician.

Fonte: DOBOT (2023).

Em um dos testes automatizados propostos o braço robótico será utilizado para clicar no botão “Sim” que aparece na *Head Unit.* Essa ação é necessária quando a central multimídia inicia o processo de desligamento, respondendo uma mensagem de confirmação exibida na tela.

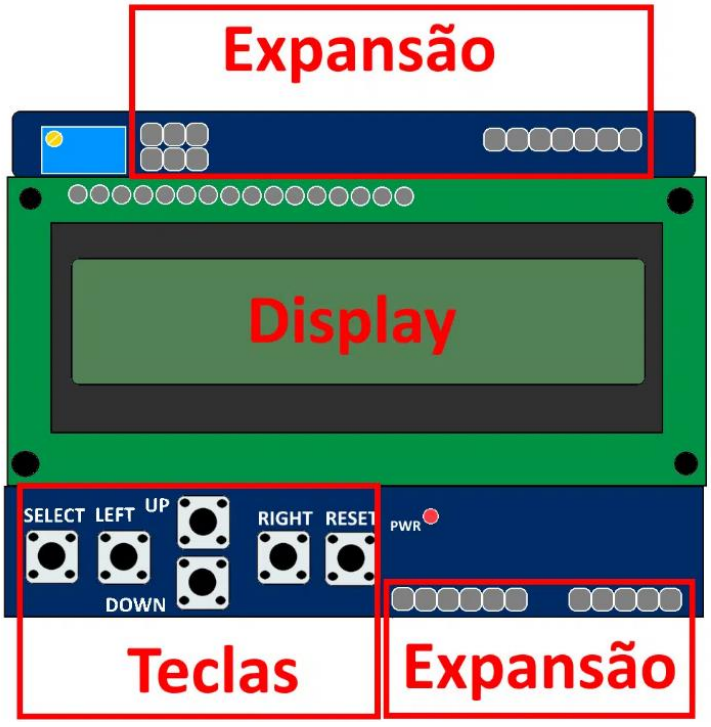
3.2.3 Arduíno *Shield* e periféricos

Os *Shields* para Arduino são placas eletrônicas que podem ser utilizadas junto ao microcontrolador, permitindo ampliar sua capacidade de funções de maneira simples e confiável. Esses *Shields* possuem uma série de aplicações, tais como: sensores de temperatura, umidade, pressão, controle de motores, controle de LEDs, Wi-Fi, GPS, display, entre outros (MAKERHERO, 2024).

O Display *Shield* é uma opção para incluir uma interface homem-máquina em projetos com Arduino. Os conectores de expansão do *shield* dão acesso aos demais pinos do Arduino, além de permitir a ligação de sensores e atuadores (MAKERHERO, 2024).

O hardware do Display *Shield* é basicamente composto por três blocos: display, teclado e conectores de expansão, conforme ilustrado na Figura 7.

Figura 7: Arduino Display Shield.



Fonte: MAKERHERO (2024).

Sendo o display alfanumérico de 2 linhas por 16 colunas, com *backlight* azul, o

shield utiliza 4 pinos do Arduino para operar em modo de 4 bits, conforme representado na tabela 1.

Tabela 1: Equivalencia pinos Display e Arduíno.

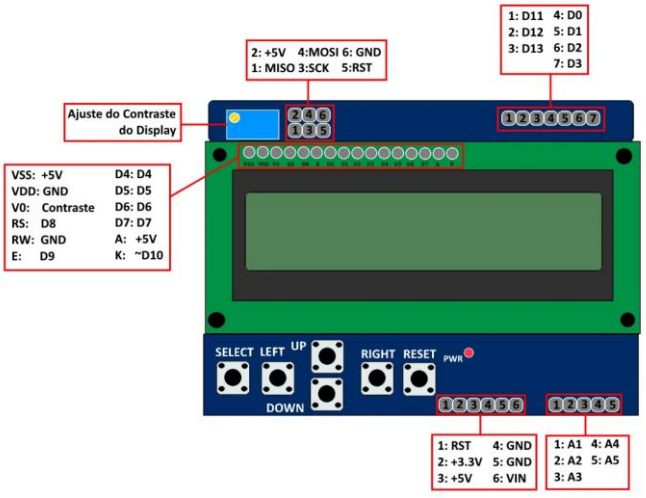
Sinal do Display	Pino do Arduíno
D4	D4
D5	D5
D6	D6
D7	D7
RS	D8
E	D9
R/W	GND (só escrita)

Fonte: MAKERHERO (2024).

Os botões possuem interação com operador, sendo ligados por uma entrada analógica utilizando uma combinação de resistores para realizar diferentes funções, como: direita, esquerda, baixo, cima, etc.

Por fim, os conectores de expansão permitem ligar outros dispositivos aos pinos não utilizados do Arduíno, conforme ilustrado na Figura 8.

Figura 8: Conectores de expansão Display Shield.



Fonte: (MAKERHERO, 2024).

Neste trabalho, o Display *Shield* será utilizado com o objetivo de apresentar o status atual do teste realizado, bem como registrar o número de interações e contabilizar o tempo em que o teste permanece em *key-off,* simulando a condição de chave do veículo desligada. Além disso, outros *shields* serão empregados com diferentes propósitos para reproduzir o ambiente de um veículo, tais como: módulo relé para o acionamento de botões e sensores de cor para capturar o brilho da tela do display da *Head Unit*, a fim de identificar o fenômeno da tela preta.

Os periféricos também são componentes importantes para a montagem da configuração de teste, sendo eles:

- Botões para o acionamento de relés e para simular a partida do veículo;
- Fonte de energia para alimentar todo o ambiente de teste, incluindo a *Head Unit* a ser avaliada;
- Placa divisora de tensão para realizar a interface de alimentação entre o Arduino e o braço robótico;
- Câmera para capturar as imagens da *Head Unit*, com o objetivo de identificar o fenômeno da tela preta.

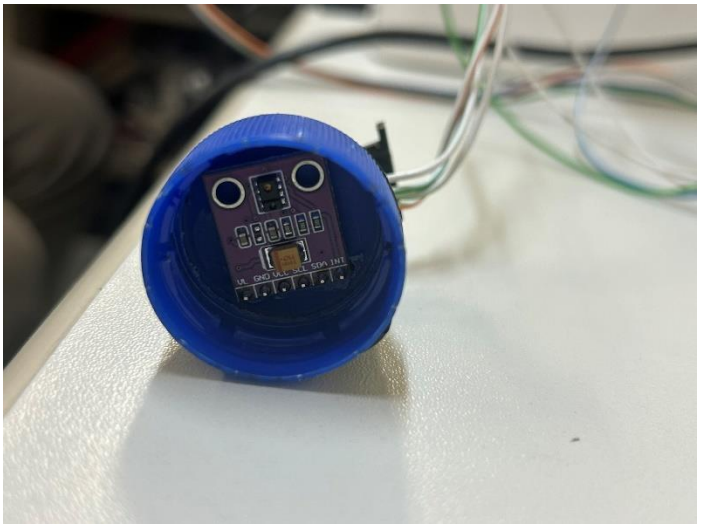
3.2.4 Módulo APDS-9960

O módulo APDS-9960, produzido pela Broadcom, integra um sensor de cor RGB, um sensor de gestos e um sensor de luminosidade em um único encapsulamento. Este módulo é notável por sua aplicação em dispositivos como o Galaxy S5 da Samsung, onde é utilizado para interfaces de movimento e controle de luminosidade. Para facilitar a integração com microcontroladores como o Arduino, o APDS-9960 utiliza uma interface de comunicação I2C, além de possuir pinos dedicados para interrupção (INT) e infravermelho (VL) (ELETROGATE, 2024).

O sensor de cor do módulo segue o padrão RGBC (Vermelho, Verde, Azul e Claro), permitindo a detecção precisa da intensidade da luz em diversas condições de luminosidade. Essa precisão é possível graças aos materiais de atenuação e ao filtro de bloqueio UV-IR integrados no sensor, que melhoram a medição da luz ambiente e das cores. O APDS-9960 é amplamente utilizado em dispositivos móveis para controle de luminosidade e funções de movimento, como, desligar a tela ao aproximar o telefone do ouvido. Além disso, sua capacidade de detecção de gestos com alta precisão e velocidade o torna útil em aplicações de robótica e como interruptor óptico em diferentes contextos de uso (ELETROGATE, 2024).

Na programação do APDS-9960, a captura de luz é configurada através de comandos específicos que ajustam os fotodiodos e LEDs infravermelhos. Utilizando bibliotecas como a Adafruit_APDS9960 para Arduino, é possível inicializar o sensor e habilitar suas funcionalidades com comandos simplificados. A leitura dos dados capturados é feita através de funções que retornam os valores digitais correspondentes à intensidade da luz ou à distância medida, facilitando a integração do sensor em diversos projetos. A Figura 9 ilustra o sensor utilizado.

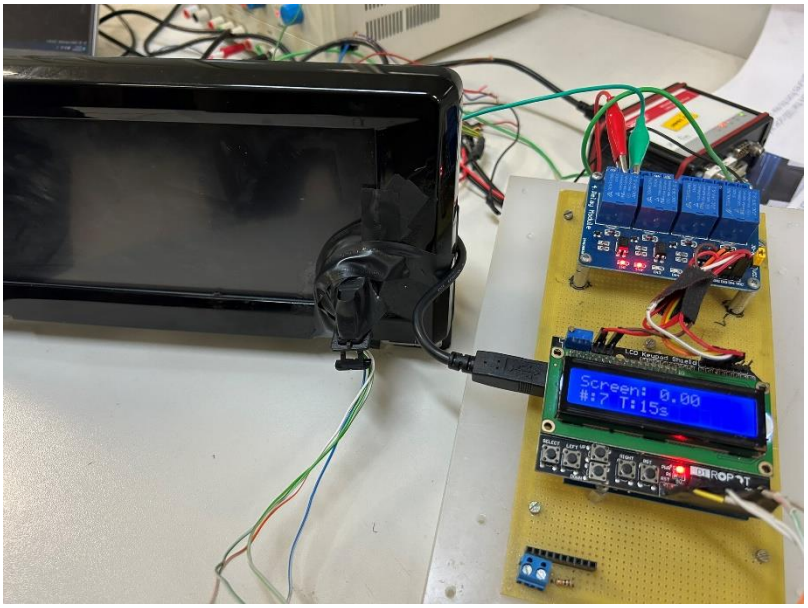
Figura 9: Módulo APDS-9960 utilizado como sensor de luz.



Fonte: Elaborado pelo autor (2025).

A Figura 10 ilustra o sensor de luz instalado no display da *Head Unit* com o objetivo de realizar as medições de brilho. Quando o brilho medido está abaixo de um limiar predefinido, infere-se que o display da central apresenta um problema de “tela preta”.

Figura 10: Sensor de luz instalado no display da Head Unit.



Fonte: Elaborado pelo autor (2025).

3.2.5 Vector CANCase e programação em CAPL

A *Vector CANcase* é uma interface de hardware que facilita a comunicação entre um computador e uma rede CAN, sendo amplamente utilizada no desenvolvimento e teste de sistemas automotivos. Conectada via USB, a CANcase suporta diferentes versões do protocolo CAN, incluindo o CAN FD, e é compatível com diversas aplicações, que vão desde o desenvolvimento de software até o diagnóstico de falhas em veículos. Ferramentas de software complementares, como o *CANoe* e o *CANalyzer*, permitem a simulação, análise e teste de redes CAN (VECTOR, 2024). A Figura 11, apresentada a seguir, ilustra uma *CANcase* da Vector.

Figura 11: Vector CANCase.

Fonte: VECTOR (2024).

A *CAPL* (*CAN Access Programming Language*) é uma linguagem de programação desenvolvida pela Vector para a criação de scripts que interagem com redes CAN, sendo utilizada principalmente nas ferramentas *CANoe* e *CANalyzer*. Semelhante à linguagem C, a CAPL é baseada em eventos e permite a definição de funções, variáveis e bibliotecas padrão para operações comuns. Esse recurso facilita a implementação de simulações, testes automáticos e diagnósticos em redes CAN, tornando o desenvolvimento mais modular e eficiente (VECTOR, 2024).

Em conjunto, a *Vector CANcase* e a programação em *CAPL* oferecem uma solução robusta e flexível para o desenvolvimento e teste de sistemas automotivos. A *CANcase* proporciona uma interface eficiente para a comunicação com a rede CAN, enquanto a *CAPL* possibilita a criação de scripts personalizados para simulação, teste e diagnóstico. Essas ferramentas são fundamentais para assegurar a eficiência e a confiabilidade dos sistemas automotivos modernos.

Neste trabalho, as ferramentas *Vector CANcase* e a programação em *CAPL* serão utilizadas para automatizar mensagens na rede veicular CAN, simulando alguns sinais, tais como: partida do veículo, posição da chave de ignição e sinais do câmbio para identificar a marcha em ré.

3.2.6 PyCharm e programação em Python

O PyCharm é um Ambiente de Desenvolvimento Integrado (IDE) desenvolvido pela JetBrains, projetado para facilitar a programação em Python. Ele oferece funcionalidades como autocompletar, realce de sintaxe, verificação de erros em tempo real e um depurador integrado. Além disso, o PyCharm suporta diversos *frameworks*, ferramentas de controle de versão e ambientes virtuais, sendo amplamente utilizado tanto para desenvolvimento *web* quanto para ciência de dados (JETBRAINS, 2024).

A programação em Python envolve o uso de uma linguagem de programação de alto nível, conhecida por sua sintaxe clara e legível, o que facilita tanto o aprendizado quanto a escrita de código eficiente. Trata-se de uma linguagem de programação amplamente utilizada e altamente adaptável, destacando-se por sua facilidade de uso e clareza. Sua vasta coleção de bibliotecas disponíveis permite que os desenvolvedores construam desde aplicações básicas até projetos de grande complexidade. É amplamente utilizada em diversas áreas, como desenvolvimento *web*, análise de dados, automação de tarefas, inteligência artificial e *machine learning*, devido à sua simplicidade, robustez e a uma comunidade ativa que continuamente desenvolve e compartilha novos módulos e *frameworks* (HOSTINGER, 2024).

A programação em Python será utilizada para fazer o reconhecimento de imagem em tempo real, utilizando uma câmera posicionada de frente ao display da *Head Unit* comparando a imagem com um padrão previamente armazenada a fim de identificar a falha de “Tela Preta” ao engatar a ré. Da mesma maneira, o Python automatizará a execução do script em CAPL para simular o sinal de ré periodicamente, ativando e desativando o sinal da câmera de ré do veículo.

3.2.7 Programação no Arduino IDE

A programação no Arduino é realizada por meio do Arduino IDE (Integrated Development Environment). Sua instalação pode ser feita mediante download no site oficial do Arduino. Após a instalação, basta conectar a placa via USB a um computador, e o IDE reconhecerá automaticamente o dispositivo. A interface do Arduino IDE é simples e intuitiva, composta por uma área de edição de código, uma barra de ferramentas com botões para compilar e carregar o programa, além de uma área de mensagens que exibe erros e avisos.

Quando o código é escrito na linguagem de programação C/C++, o próximo passo consiste na verificação e compilação. Estando livre de erros, o programa pode então ser carregado na placa Arduino por meio do botão *Upload*. Nesse processo, o código é transferido para a placa, onde passa a ser executado imediatamente, permitindo que ajustes sejam realizados em tempo real, conforme a necessidade.

Toda a lógica de implementação, bem como os mecanismos de controle necessários para a execução dos testes automatizados destinados à identificação do problema de tela preta" durante a inicialização da *Head Unit*, serão desenvolvidos e gerenciados no ambiente Arduino IDE. Esse recurso oferece uma plataforma integrada e simplificada para programação, compilação e carregamento de código, permitindo que os testes sejam conduzidos de forma estruturada, modular e eficiente. Dessa maneira, garante-se maior confiabilidade na simulação das condições reais de operação, além de possibilitar ajustes rápidos e precisos conforme as necessidades do experimento.

3.3 Definição do teste e lógica do algorítimo

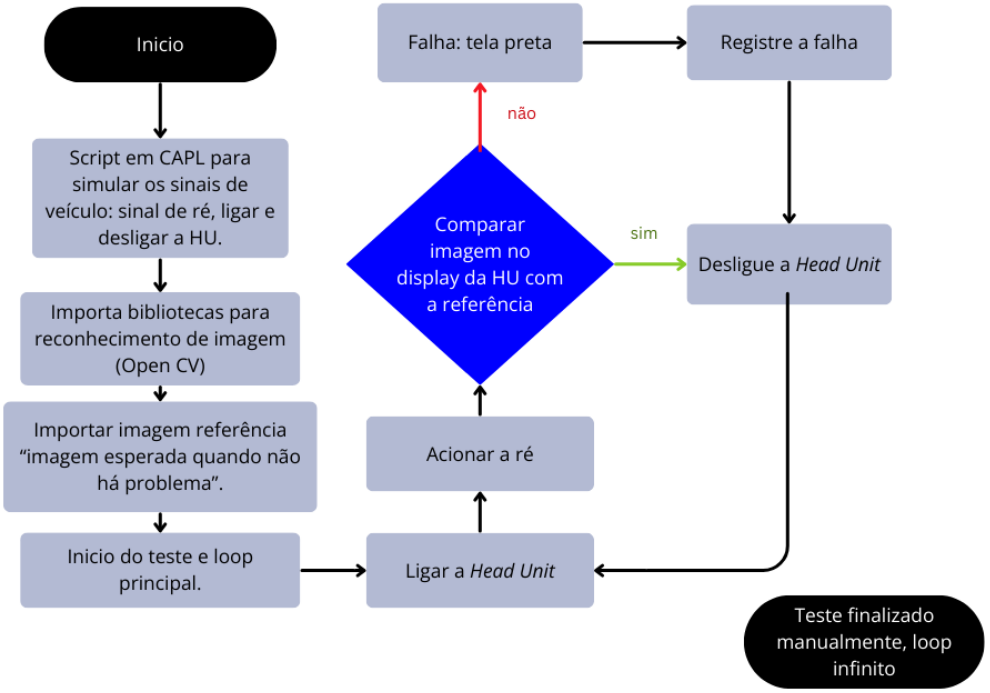
Uma vez realizado o *setup* de teste, o próximo passo é definir a lógica do algoritmo a ser implementado para a detecção do problema de tela preta na *Head Unit*, considerando os seguintes cenários:

- **Cenário 1**: Tela Preta na *Head Unit* ao engatar a ré;
- **Cenário 2:** Tela Preta na inicialização da *Head Unit*;
- **Cenário 3:** Tela preta na inicialização da *Head Unit*, utilizando um braço robótico para desligar o dispositivo.

3.3.1 Cenário 1:Tela Preta na *Head Unit* ao engatar a ré

O fluxograma representado na figura 12, exemplifica o teste a ser realizado onde a base do algoritmo será feita em Python.

Figura 12: Fluxograma Cenário 1.



Fonte: Elaborado pelo autor (2025).

3.3.1.1 Preparação do teste

O algoritmo inicia configurando o ambiente necessário para o teste. Nesta etapa, são carregados os recursos essenciais, como a imagem de referência que representa a tela esperada quando a câmera de ré é acionada, e a câmera que fará a captura das imagens exibidas na *Head Unit*. Também são definidos parâmetros básicos, como resolução das imagens e variáveis de controle, garantindo que o sistema esteja pronto para iniciar os ciclos de teste.

3.3.1.2 Início do teste

Após a preparação, o sistema liga a central multimídia e estabelece os temporizadores que serão usados para medir o tempo de cada ciclo. Essa configuração inicial assegura que o teste ocorra de forma organizada, permitindo controlar intervalos de espera e limites de tempo para cada etapa.

3.3.1.3 Ciclo principal

O algoritmo entra em um ciclo contínuo, que será repetido até que o operador decida encerrar o teste. Em cada iteração, uma imagem da tela é capturada e analisada. A lógica do ciclo se divide em dois caminhos: quando a marcha ré está engatada e quando não está. Essa estrutura garante que o sistema execute as ações corretas conforme o estado atual do teste.

3.3.1.4 Preparação para um novo ciclo

Quando a marcha ré não está engatada, o sistema aguarda um breve intervalo antes de iniciar um novo teste. Durante essa pausa, ele atualiza os contadores, registra informações em um arquivo de log e, em seguida, engata a marcha ré para começar a próxima verificação. Esse processo garante organização e rastreabilidade dos testes realizados.

3.3.1.5 Verificação da imagem

Com a marcha ré engatada, o algoritmo verifica se a tela da *Head Unit* exibe a imagem esperada. Caso a imagem seja reconhecida, o sistema aguarda alguns segundos e desengata a marcha ré, concluindo o ciclo com sucesso. Se a imagem não aparecer dentro do tempo limite, o sistema considera que houve uma falha.

3.3.1.6 Registo da falhas

Quando ocorre uma falha, o algoritmo registra essa ocorrência, salva um vídeo com as imagens capturadas durante o teste e atualiza o arquivo de log com os detalhes do ciclo. Esse registro é fundamental para análise posterior, permitindo identificar padrões e causas do problema.

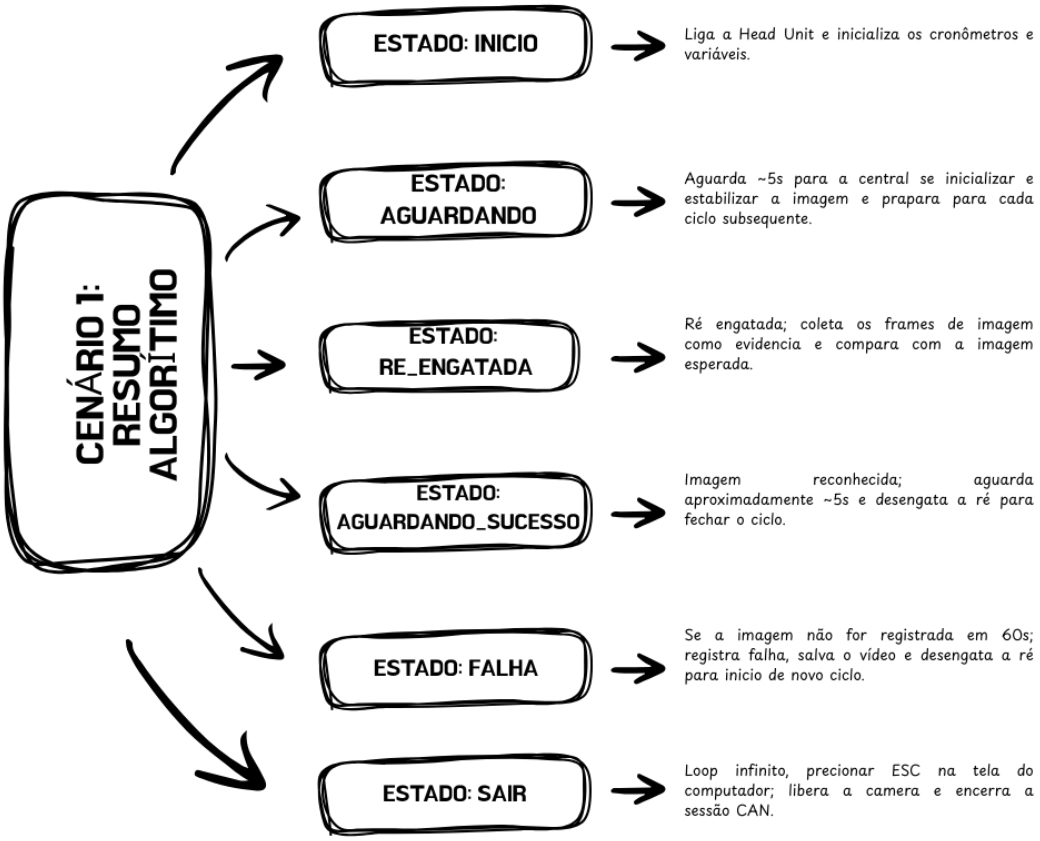
3.3.1.7 Interface e encerramento

Durante toda a execução, o sistema apresenta na tela informações como número de testes realizados, quantidade de falhas, status da marcha ré e instruções para encerrar o teste. Quando o operador decide finalizar, o algoritmo libera os recursos utilizados, fecha as janelas e encerra a comunicação com os sistemas, garantindo um término seguro e organizado.

3.3.1.8 Resumo do algorítimo

A Figura 13, ilustra de maneira didática como o algoritmo foi construído, para um melhor entendimento deste trabalho.

Figura 13: Resumo da lógica do algoritmo para o cenário 1.

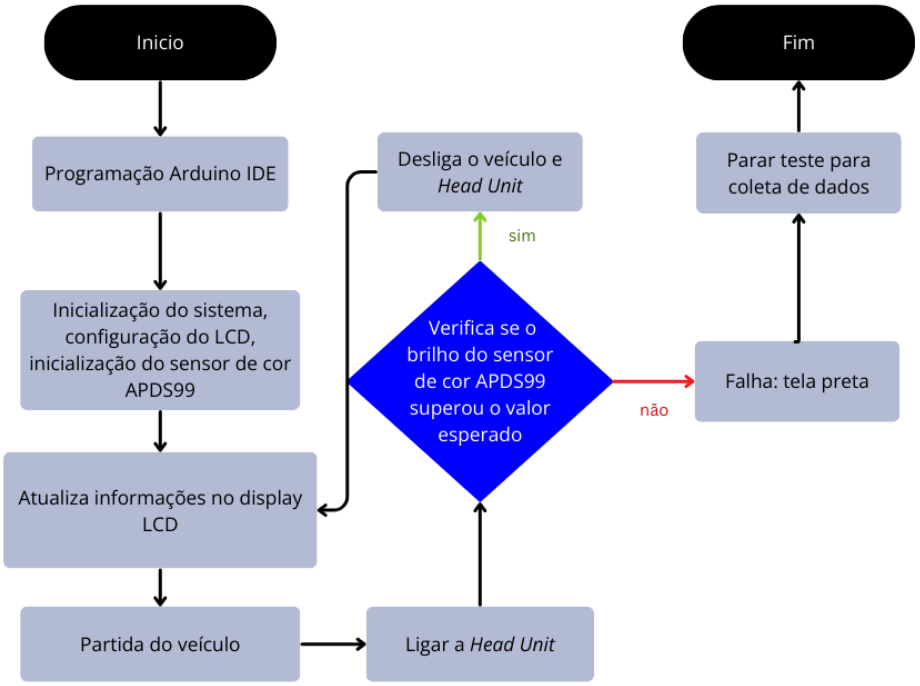


Fonte: Elaborado pelo autor (2025).

3.3.2 Cenário 2: Tela Preta na inicialização da *Head Unit*

O fluxograma representado na figura 14, mostra o teste a ser realizado onde a base do algoritmo será feita no Arduino IDE:

Figura 14: Fluxograma Cenário 2.



Fonte: Elaborado pelo autor (2025).

3.3.2.1 Inclusão de bibliotecas e definições iniciais

O código inicia incluindo as bibliotecas necessárias para o controle do display LCD e para a leitura do sensor de luminosidade APDS9930. Em seguida, são definidas constantes que padronizam o acionamento dos relés (ON e OFF) e os parâmetros de tempo do teste. Esses parâmetros determinam o tempo mínimo, máximo e o incremento do tempo dinâmico entre os ciclos, permitindo a variação controlada das condições de teste.

3.3.2.2 Definição dos estados

É criada uma numeração que define os estados do teste: início, partida do veículo, desligamento do veículo e verificação da inicialização da tela. Essa máquina de estados organiza a lógica do algoritmo, garantindo que cada etapa do teste seja executada em ordem e de forma controlada.

3.3.2.3 Configuração dos pinos e variáveis do sistema

O código define os pinos utilizados pelo display LCD, pelo botão de reinício do teste e pelos relés responsáveis por simular a ignição do veículo. Em paralelo, são inicializadas variáveis que controlam o estado atual do teste, o número de repetições, o valor de luminosidade lido pelo sensor e o tempo dinâmico entre os ciclos.

3.3.2.4 Funções auxiliares

São implementadas duas funções auxiliares: uma para escrever mensagens no display LCD, sempre limpando e atualizando as duas linhas, e outra para converter valores de tempo de milissegundos para segundos, facilitando a exibição das informações no display durante o teste.

3.3.2.5 Inicialização do sistema (setup)

Na função de inicialização, o display LCD é configurado, os pinos dos relés são definidos como saídas e o veículo é inicialmente mantido desligado. Em seguida, o sensor APDS9930 é inicializado e colocado em modo de leitura contínua de luz ambiente. Caso ocorra falha na inicialização do sensor, mensagens de erro são exibidas no console serial. Por fim, o sistema define o estado inicial do teste, zera o contador de ciclos e inicializa o tempo dinâmico com o valor mínimo configurado.

3.3.2.6 Rotina de reinício do sistema

Durante a execução contínua do programa, o código verifica se o botão de reinício foi pressionado. Caso isso ocorra, o teste é reiniciado imediatamente, retornando ao estado inicial, zerando o contador de repetições e redefinindo o tempo dinâmico para o valor mínimo.

3.3.2.7 Funcionamento do código do Cenário 2

No estado inicial, o display exibe a mensagem de início do teste. O sistema aguarda um curto período para estabilização e, em seguida, muda automaticamente para o estado de partida do veículo. A Figura 15, descreve esse estado no algorítimo.

Figura 15: Código estado inicial - Cenário 2

```
case START: // Procedimento inicial
  printLCD("Starting...", "                "); // Escreve o texto no display do Arduíno

  delay(2000); // Para o tempo do teste pelo tempo indicado

  testStep = CAR_START; // Define o próximo procedimento de teste que será executado
break;
```

Fonte: Elaborado pelo autor (2025).

No estado de partida do veículo (CAR_START), o display mostra o status da partida, o número da iteração atual e o valor do tempo dinâmico. Os dois relés são energizados, simulando a ignição ligada do veículo e iniciando a inicialização da *Head Unit*. Após um atraso fixo, o sistema avança para o estado de verificação da tela. A Figura 16 ilustra o código para o estado de CAR_START.

Figura 16: Código partida do veículo - Cenário 2

```
case CAR_START: // Procedimento de partida do veículo
  printLCD("START CAN", "#:" + (String)countRepetition + " T:" + (String)timeConvertToSecond(dynamicTimer) + "s");

  digitalWrite(pinRelayCh1, ON); // Energiza o primeiro relé para dar partida no veículo
  digitalWrite(pinRelayCh2, ON); // Energiza o segundo relé para ligar a Head Unit

  delay(2000); // Para o tempo do teste pelo tempo indicado

  testStep = WAIT_SCREEN_INITIALIZE; // Define o próximo procedimento de teste que será executado
break;
```

Fonte: Elaborado pelo autor (2025).

Quando o sistema está no estado de verificação da tela, no algoritimo chamado de WAIT_SCREEN_INITIALIZE , ele espera para a inicialização da *Head Unit* e exibe essa condição no display. Em seguida, o sensor de luminosidade realiza a leitura do brilho da tela.

O valor de referência de 0,45 lux foi determinado a partir de testes experimentais previamente realizados, nos quais foram analisadas leituras de luminosidade em condições normais de inicialização e em cenários de falha. A partir dessas medições, definiu-se esse limiar como critério confiável para distinguir entre tela inicializada e falha de inicialização.

- Se o valor lido for **superior a 0,45 lux**, o software interpreta que a tela foi corretamente inicializada, aguarda um tempo adicional para estabilização e avança para o desligamento do veículo consequentemente da *Head Unit.*
- Se o valor for **inferior ou igual a 0,45 lux**, o sistema interpreta que a tela não ligou corretamente. Nesse caso, o teste fica retido nesse estado, exibindo no display o valor de luminosidade, a iteração e o tempo dinâmico em que a falha

ocorreu, permitindo a coleta de logs e análise do problema.

A Figura 17 a mostra o cógido no algorítimo referente ao estado de WAIT_SCREEN_INITIALIZE.

Figura 17: Código leitura luminosidade da tela - Cenário 2

```
case WAIT_SCREEN_INITIALIZE: // Procedimento para verificar se a tela da multimídia ligou conforme esperado
  printLCD("Waiting HU ON...", "#:" + (String)countRepetition + " T:" + (String)timeConvertToSecond(dynamicTimer) + "s"); // Escreve o texto no

  // Read the light levels (ambient, red, green, blue)
  if (  !apds.readAmbientLightLux(ambient_light)) { // Realiza a leitura do brilho da tela da multimídia pelo sensor
    Serial.println(F("Error reading light values"));  // Escreve o texto no console do arduino caso a leitura tenha dado errado
  }
  else
  {
    Serial.println(ambient_light);  // Escreve o texto no console do arduino caso a leitura tenha dado certo
  }

  if (ambient_light > 0.45){ // Verifica se o valor do brilho da tela de multimídia corresponde ao valor de tela ligada (com boa margem de erro)
    delay(15000); // Para o tempo do teste pelo tempo indicado
    testStep = CAR_STOP; // Define o próximo procedimento de teste que será executado
  }else{ // A prova fica Estagnada nesse procedimento caso a tela da multimídia do carro não tenha ligado para a coleta dos logs (além do proced
    printLCD("Screen: " + (String)ambient_light, "#:" + (String)countRepetition + " T:" + (String)timeConvertToSecond(dynamicTimer) + "s"); // E

    delay(1000); // Para o tempo do teste pelo tempo indicado
  }
break;
```

Fonte: Elaborado pelo autor (2025).

Por fim, quando a tela é considerada ligada corretamente, o estado CAR_STOP desenergiza os dois relés, simulando o desligamento do veículo e da *Head Unit* . Em seguida, o sistema aguarda um período definido pelo tempo dinâmico, que varia entre 50 segundos e 3 minutos, simulando diferentes tempos de desligamento entre ciclos. A Figura 18 descreve essa parte no código.

Figura 18: Código desligamento da head unit - Cenário 2

```
case CAR_STOP: // Procedimento de desligar o veículo
  printLCD("STOP CAN",  "#:" + (String)countRepetition + " T:" + (String)timeConvertToSecond(dynamicTimer) + "s"); // Escreve o texto no display

  digitalWrite(pinRelayCh1, OFF); // Desenergiza o primeiro relé para desligar o veículo
  digitalWrite(pinRelayCh2, OFF); // Desenergiza o primeiro relé para desligar a head unit

  delay(dynamicTimer); // Para o tempo do teste pelo tempo dinâmico (varia entre 50s e 3min)

  testStep = CAR_START; // Define o próximo procedimento de teste que será executado (retorna para o primeiro procedimento da próxima iteração)

  countRepetition++; // Incrementa +1 ao número de repetição das interções

  // Atualiza o valor do tempo variável
  if(dynamicTimer >= STOPTIMER){ // Verifica se o tempo variável é igual ou maior que o tempo limite
    dynamicTimer = STARTTIMER; // Reinicializa o valor do tempo dinâmico para o inicial para a próxima iteração
  }else{
    dynamicTimer += DELTATIMER; // Incrementa o valor do tempo variável para a próxima iteração
  }
```

Fonte: Elaborado pelo autor (2025).

Após o desligamento, o contador de iterações é incrementado. O tempo é então atualizado: caso tenha atingido o valor máximo configurado, ele é reiniciado para o valor mínimo; caso contrário, é incrementado em passos de 5 segundos. Essa lógica garante a variação progressiva das condições de teste.

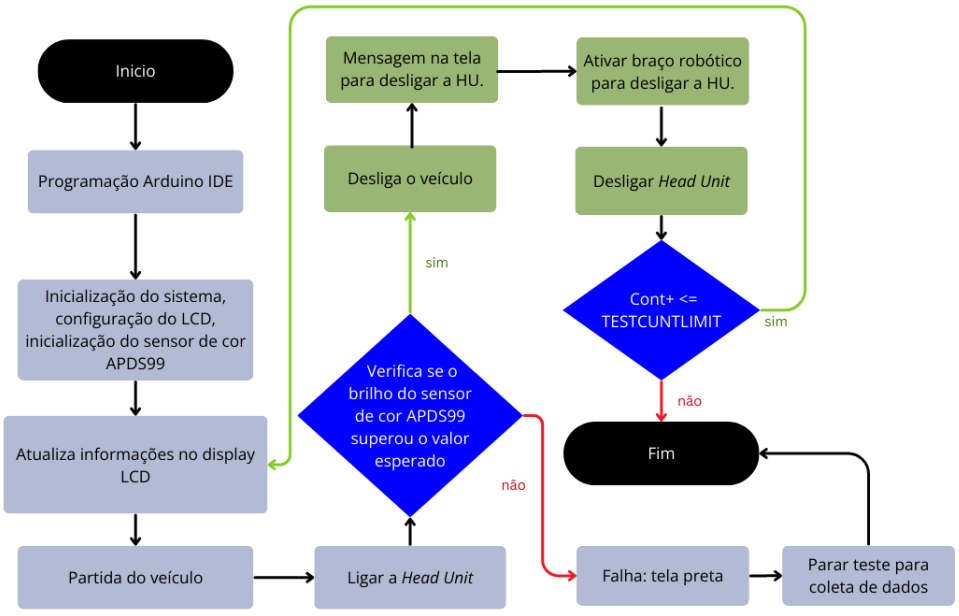
Finalizadas as atualizações, o sistema retorna ao estado de partida do veículo, iniciando automaticamente uma nova interação do teste. Esse ciclo se repete indefinidamente enquanto o sistema estiver ligado.

Por fim, o código proposto implementa um teste automatizado, cíclico e baseado em máquina de estados, que simula repetidas partidas e desligamentos do veículo, mede a luminosidade da tela da *Head Unit* após cada inicialização e identifica automaticamente falhas de “tela preta”. A variação controlada do tempo entre ciclos aumenta a robustez do teste e permite a detecção de falhas intermitentes sem necessidade de intervenção humana.

3.3.3 Cenário 3: Tela preta na inicialização da *Head Unit*, utilizando um braço robótico para desligar o dispositivo.

O fluxograma representado na figura 19, exemplifica a ideia do teste a ser realizado onde a base do algoritmo será feita no Arduino IDE, adicionando um braço robótico para confirmar o desligamento da Head Unit via toque no display.

Figura 19: Fluxograma cenário 3.



Fonte: Elaborado pelo autor (2025).

3.3.3.1 Funcionamento do código do cenário 3

O principal diferencial em relação ao Cenário 2 é a inclusão de um braço robótico para realizar o desligamento da central multimídia, conforme ilustrado no fluxograma apresentado na Figura 19. Essa adição permite automatizar uma etapa que, em condições normais, exigiria interação manual do usuário com a interface da

Head Unit.

A estrutura lógica do software permanece essencialmente a mesma descrita no cenário anterior. No entanto, neste cenário específico, quando o valor de luminosidade lido pelo sensor é superior a 0,45 lux, o algoritmo interpreta que a tela foi corretamente inicializada. Após essa confirmação, o sistema aguarda um tempo adicional de estabilização, garantindo que a interface gráfica esteja completamente carregada antes de prosseguir com a próxima etapa do teste.

Na sequência, o software avança para o processo representativo de desligamento do veículo. Nesse momento, a *Head Unit* exibe uma mensagem de confirmação ao usuário, questionando se o desligamento da central deve ser realizado. Para tratar essa interação de forma automatizada, um pulso elétrico controlado pela saída analógica do Arduino é enviado ao braço robótico, que executa o toque na tela da *Head Unit*, confirmando o comando de desligamento.

O ciclo de teste prossegue de forma contínua até que seja atingido o número de iterações previamente definido pelo validador. Caso seja detectado um evento de tela preta em qualquer uma das iterações, o teste é imediatamente interrompido, permitindo a coleta de informações e registros de log, os quais serão utilizados posteriormente para análise detalhada da falha. A Figura 20 descreva essa automação feita no código.

Figura 20: Código cenário 3

```
    case ROBOT:
      printLCD("STOP HU (robot)",  "#:" + (String)countRepetition + " T:" + (String)timeConvertToSecond(dynamicTimer) + "s");

// Desliga o relé ou o sinal enviado (OFF)
      digitalWrite(requestRoboticArm, OFF);
        delay(500);
// Liga o relé ou o sinal enviado (ON)
      digitalWrite(requestRoboticArm, ON);

      delay(dynamicTimer);

      countRepetition++;

      if(dynamicTimer >= STOPTIMER){
        dynamicTimer = STARTTIMER;
      }else{
        dynamicTimer += DELTATIMER;
      }

      if (countRepetition == TESTCOUNTLIMIT){
        testStep = -1;
        printLCD("Test end!",  "#:" + (String)countRepetition + " T:" + (String)timeConvertToSecond(dynamicTimer) + "s");
      }

    break;
  }
```

Fonte: Elaborado pelo autor (2025).

4 RESULTADOS

Nesta seção são apresentados os resultados obtidos com a implementação dos testes automatizados propostos para a detecção do problema de “Tela Preta” na *Head Unit*. As soluções desenvolvidas foram aplicadas em cenários definidos previamente no capítulo de Metodologia, e buscaram validar o comportamento do sistema de *infotainment* em momentos críticos do seu ciclo de funcionamento.

4.1 Cenário 1:Tela Preta na Head Unit ao engatar a ré

O objetivo principal deste cenário foi desenvolver uma solução capaz de verificar, de forma automática e precisa, se o display da central multimídia (*Head Unit*) está funcionando corretamente quando determinadas ações do veículo ocorrem. Especificamente, busca-se identificar se, ao engatar a marcha ré, a imagem apropriada é exibida na tela do sistema de *infotainment* ou se ocorre um evento de tela preta. Para isso, foi elaborada uma arquitetura de testes baseada nas linguagens de programação Python e CAPL.

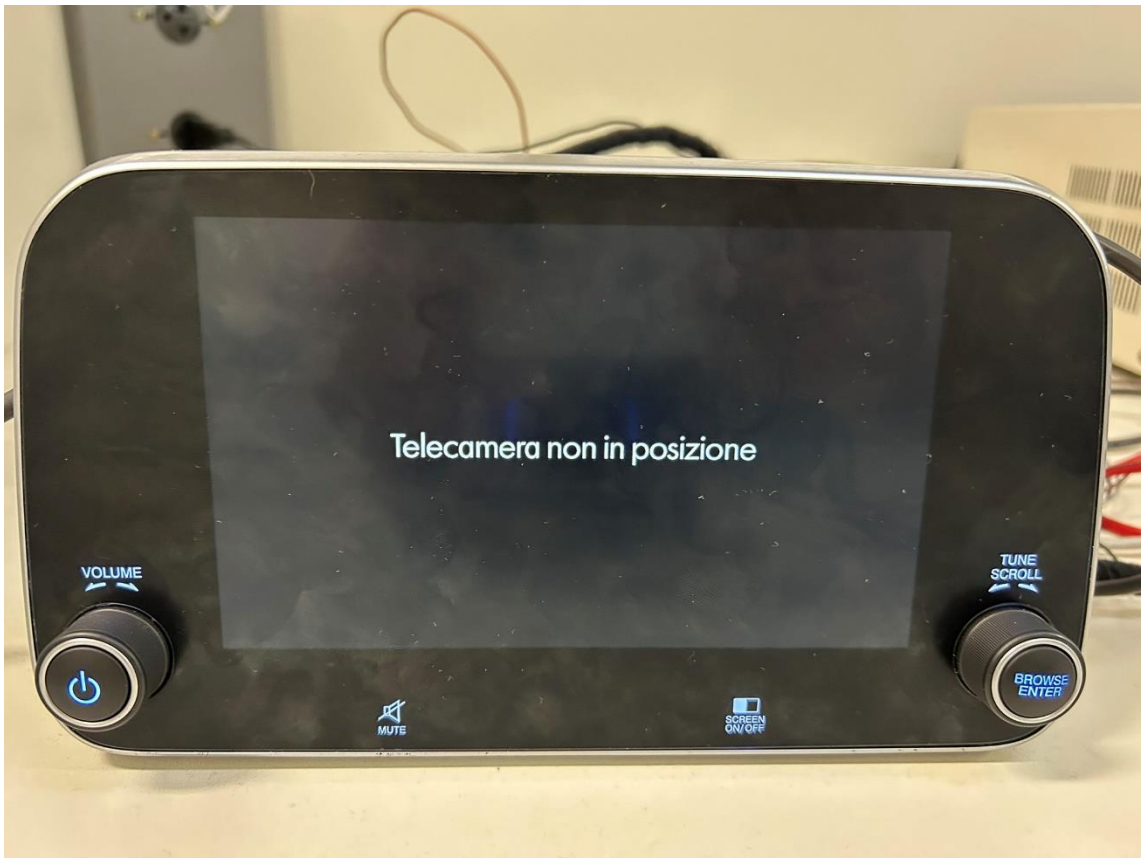
A lógica emprega uma câmera posicionada diante da tela da central multimídia, controlada por um *script* em Python que captura imagens em tempo real. Essas imagens são comparadas a uma referência previamente armazenada, utilizando os algoritmos SIFT (*Scale-Invariant Feature Transform*), responsável pela extração de pontos de interesse invariantes a escala e rotação (LOWE, 2004), e FLANN (*Fast Library for Approximate Nearest Neighbors*), que realiza a correspondência eficiente desses pontos em alta velocidade (MUJA; LOWE, 2009). Ambos os algoritmos são integrados via biblioteca OpenCV, amplamente adotada em aplicações de visão computacional pela sua robustez e suporte a aplicações em tempo real (ITSEEZ, 2015).

Os algoritmos utilizados para reconhecimento de imagem, SIFT e FLANN, foram aplicados por meio da biblioteca OpenCV no algoritimo para este cenário. A escolha se justifica por OpenCV ser uma biblioteca de código aberto e amplamente utilizada na área de Visão Computacional, o que permite a implementação e a integração rápida e eficiente desses algoritmos em sistemas como o desenvolvido neste trabalho.

O teste foi realizado em bancada e a câmera não estava instalada no veículo, ao acionar a marcha ré, a tela da central exibe a mensagem “*Telecamera non in*

posizione”. Essa é a condição esperada. Caso a imagem exibida no rádio seja diferente dessa ao engatar a ré, configura-se uma condição de falha. A Figura 21 mostra a central multimídia com a imagem de referência.

Figura 21: Imagem referência utilizada para o reconhecimento no display da Head Unit.



Fonte: Elaborado pelo autor (2025).

Simultaneamente, a comunicação é estabelecida entre o script em Python e o software CANalyzer por meio de um script CAPL. Essa integração permite simular, via comandos digitais, o engate e desengate da marcha ré, assim como o acionamento da central multimídia, criando um ambiente de testes automatizados fiel à realidade veicular. A sequência de comandos segue uma lógica de teste contínuo: engata-se a ré, aguarda-se a exibição da imagem, e, se a imagem esperada for reconhecida pelo Open CV, a marcha ré é desengatada. Caso a imagem não seja reconhecida após um tempo limite de 60 segundos, o sistema considera a ocorrência de uma falha.

Quando uma falha é detectada, o sistema automaticamente salva um vídeo contendo os últimos segundos de imagens capturadas, armazenando esse material em um diretório específico para posterior análise. Além disso, é gerado um arquivo de texto a cada ciclo de teste, registrando a data, a hora, e o número total de testes realizados e quantidade de falhas detectadas. A Figura 22 ilustra os arquivos gerados.

Figura 22: Imagem do repositório de dados amazanados durante a falha.

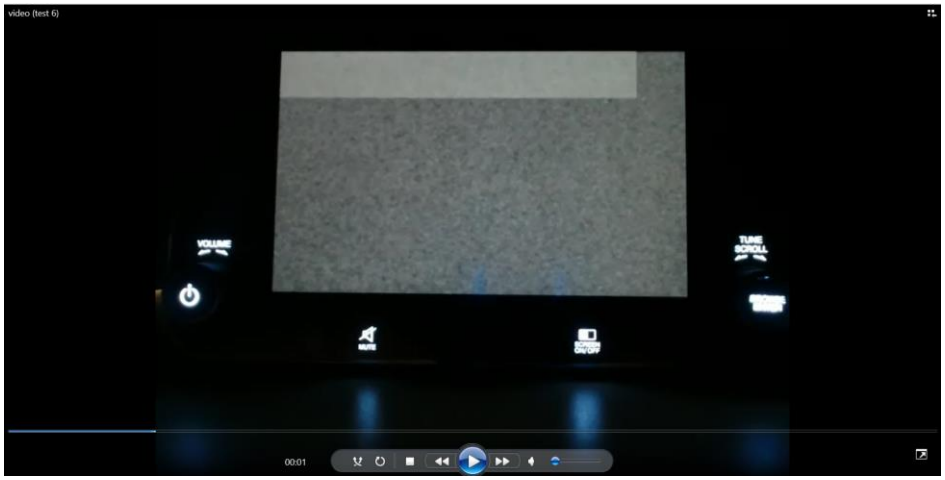
testCount	Arquivo TXT	1 KB	Não	1 KB	0%	09/09/2024 15:04
video (test 3)	Vídeo MP4	1.857 KB	Não	1.872 KB	1%	09/09/2024 15:03
video (test 6)	Vídeo MP4	1.788 KB	Não	1.800 KB	1%	09/09/2024 15:04

Fonte: Elaborado pelo autor (2025).

Os vídeos gerados durante os testes podem ser analisados posteriormente para um melhor entendimento da falha ocorrida. Em uma situação ideal, além desses registros visuais, os arquivos de *log* da *Head Unit* também seriam armazenados para complementar a análise. No entanto, o sistema em questão utiliza *logs* criptografados e requer uma rotina específica para a sua coleta. Por essa razão, a análise e a captura desses logs não fazem parte do escopo deste trabalho.

A Figura 23 apresenta uma imagem ilustrativa do vídeo coletado durante o teste. Observa-se que, ao simular o engate da marcha ré, a imagem exibida na tela da central multimídia não corresponde à imagem de referência esperada em condições normais de funcionamento. Devido a essa divergência, o sistema identificou uma falha, que foi registrada e teve o respectivo vídeo salvo automaticamente.

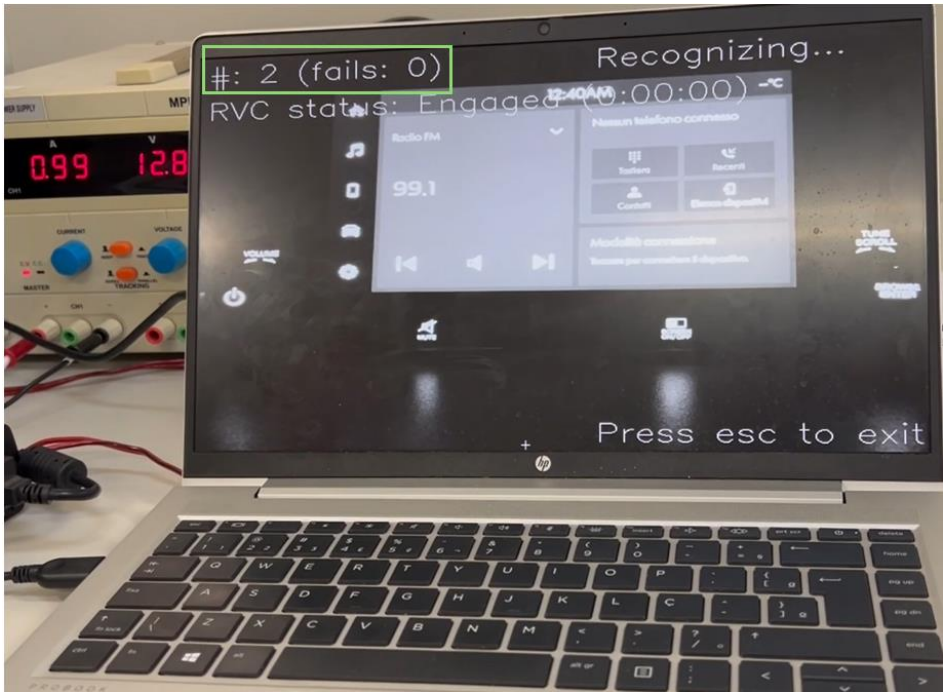
Figura 23: Imagem do vídeo gravado apresentado a falha.



Fonte: Elaborado pelo autor (2025).

As imagens 24, 25, 26 e 27 evidenciam o êxito do teste proposto. A Figura 24 ilustra como o procedimento é realizado na prática. No canto superior esquerdo da imagem, é possível observar a contagem de testes realizados (duas ocorrências, no caso) e o número de falhas identificadas (nenhuma até o momento). A mensagem “RVC Status” indica se a marcha ré está ou não engatada. No caso da imagem apresentada, a ré não está engatada, e por essa razão a central multimídia encontra-se em sua tela principal, operando em estado funcional.

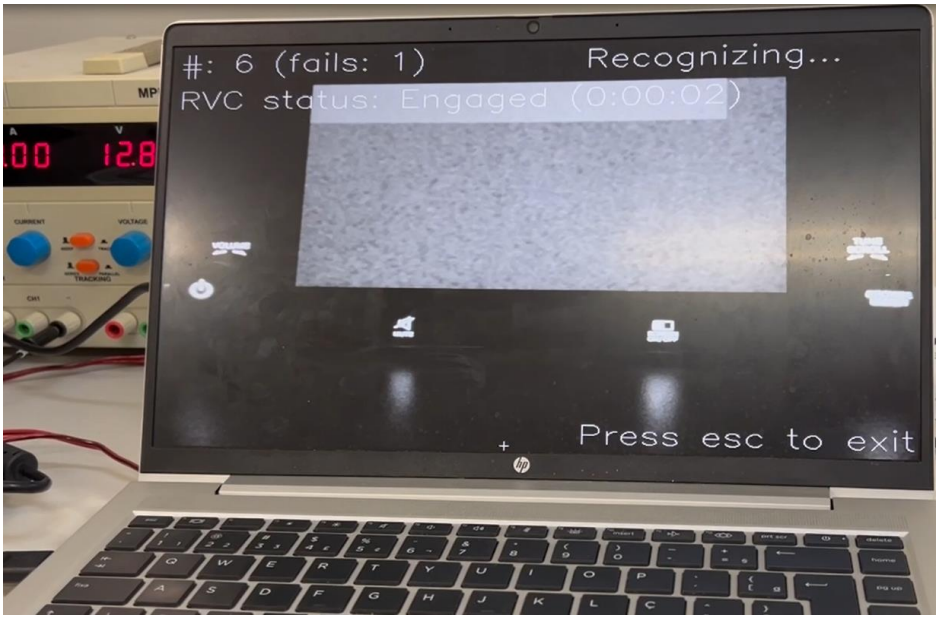
Figura 24: Monitoramento durante a execução de testes.



Fonte: Elaborado pelo autor (2025).

A Figura 25 mostra a execução do teste já com seis interações realizadas, sendo que a primeira falha foi detectada no teste 3, conforme observado na Figura 22. Observa-se, pela mensagem “RVC Status”, que a marcha ré está engatada, e a imagem exibida na tela da central multimídia corresponde a uma condição de falha. Por esse motivo, o sistema contabiliza a segunda ocorrência ao final do teste 6.

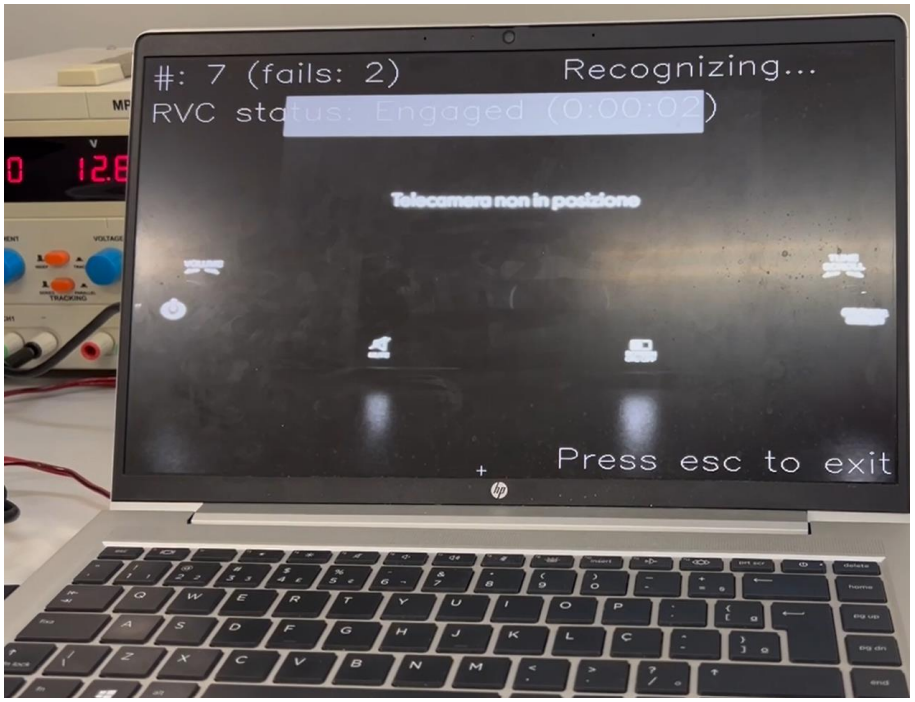
Figura 25: Teste com falha detectada (6 ciclos, 1 falha).



Fonte: Elaborado pelo autor (2025).

A Figura 26 ilustra a continuidade do teste na sétima interação, com duas falhas já contabilizadas até o momento. No entanto, nesta imagem não há falhas registradas, uma vez que a tela da central multimídia exibe corretamente a imagem de referência esperada quando a marcha ré é acionada. Isso indica que a central encontra-se em pleno funcionamento.

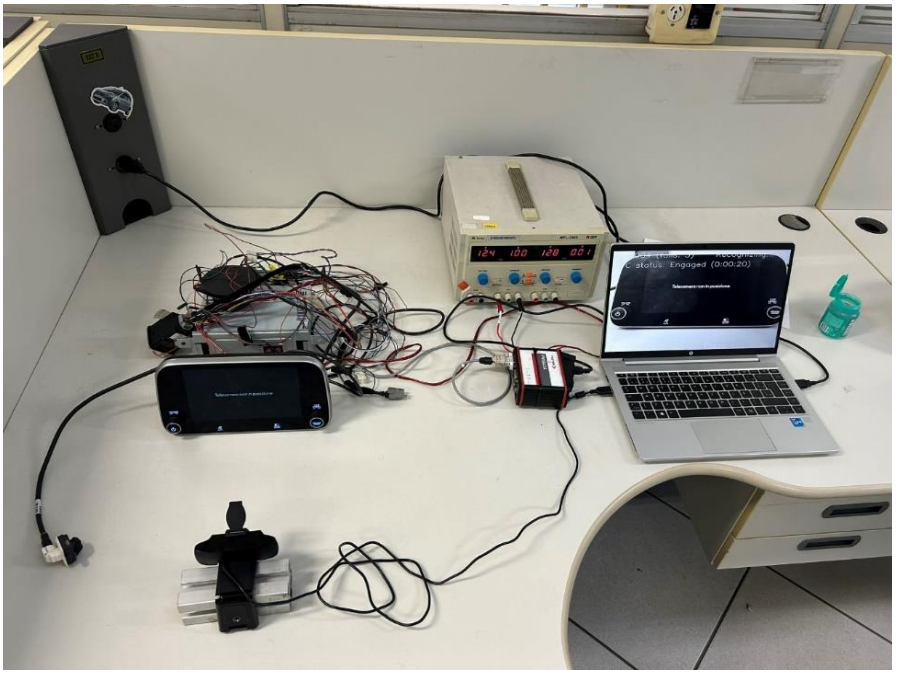
Figura 26: Teste com sucesso na validação (7 ciclos, 2 falhas).



Fonte: Elaborado pelo autor (2025).

Por fim, o teste proposto pode ser executado de forma contínua por horas ou dias, com todas as falhas sendo devidamente registradas para análise posterior. Trata-se de uma automação, implementada em uma bancada ilustrada na Figura 27, o que torna o processo de validação robusto, amplia o nível de estresse aplicado ao sistema e otimiza significativamente os testes, os quais antes eram realizados manualmente e com limitações quanto à quantidade de execuções possíveis, otimizando horas de engenharia.

Figura 27: Bancada de teste.



Fonte: Elaborado pelo autor (2025).

Diante dos resultados apresentados, é possível concluir que o Cenário 1 cumpre de forma eficaz sua proposta de automatizar a validação do funcionamento da câmera de ré da *Head Unit*. A utilização de técnicas de visão computacional, aliada à integração com o ambiente CANalyzer via CAPL, permitiu a criação de uma solução precisa, escalável e replicável. O sistema desenvolvido demonstrou capacidade de identificar corretamente falhas visuais na tela da central, registrar evidências, contabilizar ciclos e gerar logs para análise posterior. Além de reduzir significativamente a dependência da intervenção humana, essa abordagem aumenta a confiabilidade e a rastreabilidade do processo de validação, contribuindo diretamente para a melhoria da qualidade do sistema embarcado testado. Assim, o Cenário 1 reforça a importância e a viabilidade da aplicação de automações inteligentes em ambientes de validação no setor automotivo.

4.2 Cenário 2:Tela Preta na inicialização da Head Unit

O Cenário 2 foi desenvolvido com o objetivo de realizar a validação automática do funcionamento da central multimídia durante inicialização do sistema veicular, simulando o comportamento de ligar e desligar do carro. A principal hipótese a ser testada neste cenário era a ocorrência da falha conhecida como tela preta, onde a central permanece com o *display* apagado mesmo após o acionamento da ignição.

A repetição automatizada dessa situação visa identificar se a falha ocorre de forma intermitente, aleatória ou sistemática.

A bancada de testes foi composta por uma *Head Unit*, um Arduino Uno, dois relés, um sensor de luz APDS9930 e um display LCD. O Arduino foi programado para controlar o ciclo de ignição simulada do carro, ligando e desligando os relés que alimentam a central. Após cada ciclo de inicialização, o sensor de luz é acionado para medir a luminosidade da tela da *Head Unit*. Caso a luz emitida seja inferior a 0.45 lux, o sistema entende que ocorreu uma falha e interrompe o teste, registrando o evento como uma ocorrência de tela preta. Isso possibilita que o validador, ao encontrar a *Head Unit* nesse estado, consiga gravar vídeos, coletar logs da rede e sistêmicos para posterior analise e melhor entendimento do problema.

A lógica de funcionamento do algoritmo foi organizada em uma máquina de estados simples. A sequência inicia no estado START, transita para CAR_START ao simular a partida, em seguida para WAIT_SCREEN_INITIALIZE onde o sensor de luz verifica se a tela acendeu corretamente. Caso a leitura esteja dentro do valor esperado, o sistema avança para o estado CAR_STOP, desliga os relés, aguarda um tempo predeterminado e reinicia o ciclo. No entanto, se a tela não acender (brilho ≤ 0.45), o sistema exibe “BLACK SCREEN” no visor LCD e encerra o teste, indicando falha.

Durante os testes contínuos, o sistema demonstrou estabilidade e precisão na detecção de falhas. Foram realizados múltiplos ciclos com a central multimídia funcionando normalmente, e todos os ciclos foram concluídos com sucesso. A Figura 28 apresenta um exemplo do teste sendo realizado de maneira contínua. Em aproximadamente 1 hora de teste, foram realizados 298 ciclos de chave, sem que nenhuma falha fosse encontrada.

Figura 28: Exemplo de teste contínuo com 298 ciclos.

Fonte: Elaborado pelo autor (2025).

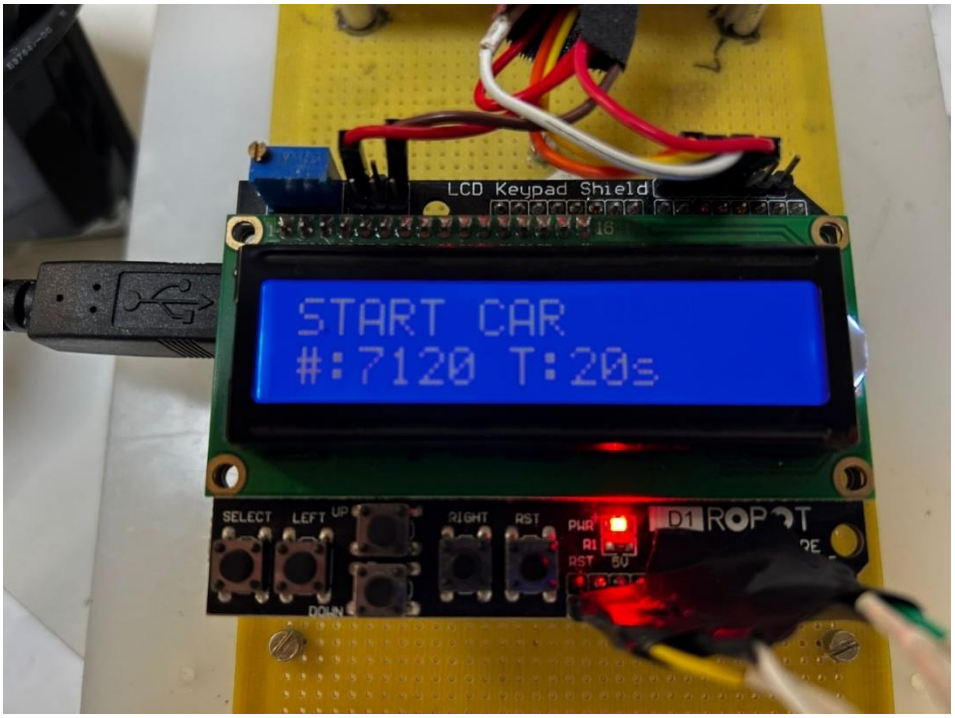
Como o teste é contínuo, após aproximadamente 12 horas de execução, foram realizados 3.560 ciclos de chave. Isso indica que a Head Unit foi ligada e desligada 3.560 vezes sem ocorrência de falhas. A Figura 29 a seguir ilustra o estado do teste após esse período.

Figura 29: Estado final do teste contínuo após 3.560 ciclos sem falhas.

Fonte: Elaborado pelo autor (2025).

Após aproximadamente 24 horas de teste contínuo, foram realizados 7.120 ciclos de chave. Isso indica que a *Head Unit* foi submetida a essa quantidade de inicializações sem apresentar problemas. Caso o teste fosse realizado manualmente, não seria possível atingir essa quantidade de ciclos em um único dia. A Figura 30 abaixo mostra a quantidade de ciclos alcançada após um dia de teste.

Figura 30: Quantidade de ciclos após um dia de teste.



Fonte: Elaborado pelo autor (2025).

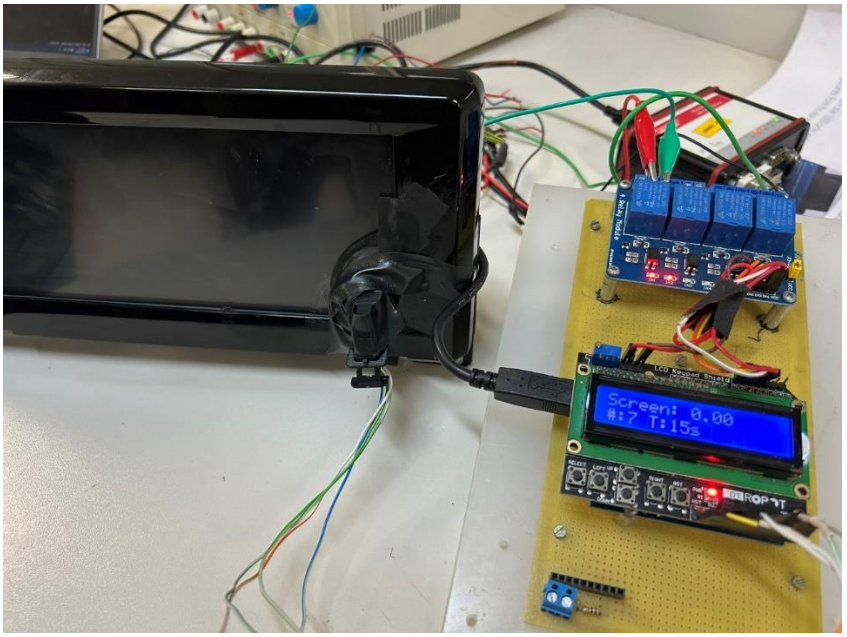
Portanto, o teste automatizado demonstrou ser eficaz para ampliar a quantidade de ciclos realizados, aumentando significativamente a confiabilidade do sistema. Isso ocorre porque o algoritmo permite a execução contínua e ininterrupta de testes, realizando centenas de ciclos em um intervalo de tempo reduzido, de um dia para o outro. Em uma rotina convencional de validação manual, ao longo de uma semana de trabalho com 5 dias úteis, um validador consegue realizar, em média, cerca de 100 ciclos por dia. Esse número limitado se deve a diversos fatores, como a jornada de 8 horas de trabalho, o tempo necessário para aguardar o desligamento completo da central e da rede veicular, bem como pausas naturais para alimentação, necessidades fisiológicas e execução de outras tarefas paralelas.

Com o uso do algoritmo proposto neste trabalho, é possível que o validador execute outras atividades enquanto a bancada realiza os testes de maneira autônoma e contínua. Considerando o mesmo período de cinco dias, a automação permite

realizar aproximadamente 35.600 ciclos de inicialização da *Head Unit*, sem ocorrência de falhas. Essa abordagem não apenas melhora a eficiência do processo de validação, como também possibilita uma análise mais abrangente e confiável do sistema testado.

No cenário da falha a ser identificada, a tela preta, o sensor de luminosidade instalado na tela da *Head Unit* desempenha um papel fundamental. Conforme definido na lógica do algoritmo, ao detectar um nível de brilho igual ou inferior a 0,45 lux, o sistema interpreta essa condição como uma falha e encerra imediatamente o teste. Nesse momento, o ciclo é registrado como falho, permitindo que o validador identifique com precisão o instante da ocorrência. A partir dessa marcação, torna-se possível coletar logs detalhados do sistema para uma análise técnica mais aprofundada, auxiliando na identificação da causa raiz da falha. A Figura 31 a seguir ilustra essa condição.

Figura 31: Detecção automática da falha, tela preta, pelo teste proposto.



Fonte: Elaborado pelo autor (2025).

Dessa forma, os resultados obtidos com a implementação do Cenário 2 comprovam a eficácia da proposta deste trabalho. O objetivo é automatizar o processo de validação da Head Unit, com foco na detecção de falhas críticas, como a ocorrência da tela preta durante a inicialização. A automação não apenas aumentou significativamente o número de testes executados em um curto intervalo de tempo, como também assegurou maior confiabilidade, rastreabilidade e repetibilidade ao processo de validação. Ao reduzir a dependência da intervenção humana e ampliar

significativamente a cobertura dos testes, o sistema desenvolvido contribui diretamente para a robustez do produto final, consolidando-se como uma ferramenta valiosa para o setor automotivo e alinhada às práticas modernas de testes automatizados em bancada.

4.3 Cenário 3: Tela preta na inicialização da Head Unit, utilizando um braço robótico para desligar o dispositivo

O Cenário 3 foi desenvolvido com o objetivo de validar automaticamente a ocorrência de falhas na tela da central multimídia durante sua inicialização, assim como no Cenário 2. A principal diferença é a adição do braço robótico Dobot Magician, acionado automaticamente pelo Arduino para interagir fisicamente com a interface da central no momento do desligamento.

Durante o processo de desligamento do veículo, a *Head Unit* não realiza o desligamento automático, exibindo uma mensagem solicitando confirmação do usuário. Para garantir a continuidade correta do ciclo de teste, é necessário acionar o botão “Sim” na tela, tarefa que anteriormente exigia intervenção manual do validador. Com a automação, essa etapa passou a ser executada de maneira precisa e repetitiva pelo braço robótico.

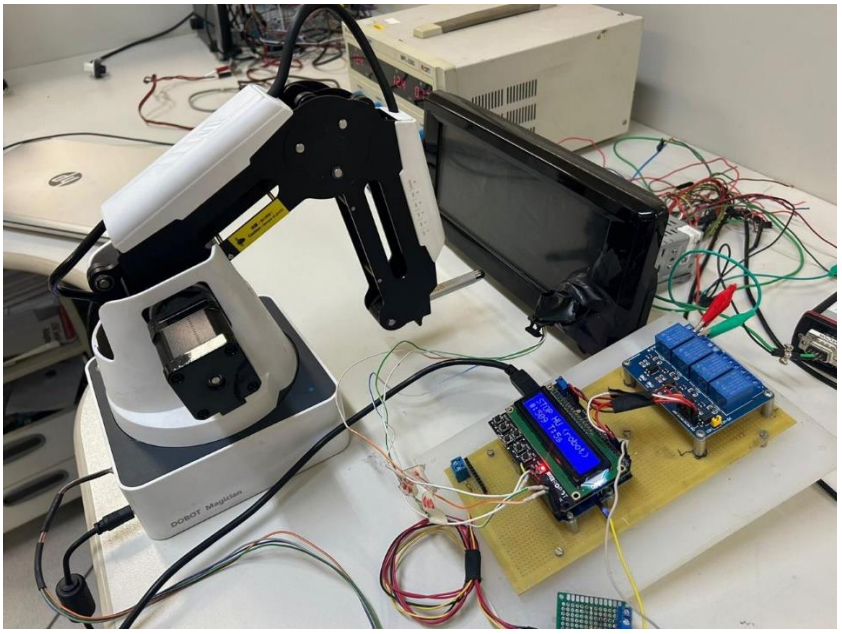
Neste cenário, o algoritmo foi configurado para executar um número predeterminado de ciclos e, ao final, encerrar automaticamente o teste. Caso seja detectado um evento de tela preta durante a execução, o sistema interrompe o ciclo imediatamente, permitindo que o validador colete informações adicionais para análise posterior.

Complementando esse cenário, o braço robótico Dobot Magician foi responsável por realizar, de forma automática e precisa, a interação física com o display da *Head Unit*, simulando o toque no botão “Sim” exibido no processo de desligamento do sistema. Para isso, foi utilizada a funcionalidade Modo Playback do software DobotStudio, que permite gravar movimentos realizados manualmente e executá-los posteriormente de forma automatizada. A sequência gravada foi salva em um arquivo .playback, garantindo que o robô repetisse exatamente o mesmo movimento sempre que o teste fosse executado. Essa abordagem trouxe mais confiabilidade à validação, ao eliminar variações humanas e assegurar que a interação com a interface ocorresse sempre nas mesmas condições. A utilização do Dobot neste cenário demonstrou ser uma solução eficaz para automatizar ações físicas específicas

que não poderiam ser simuladas eletronicamente.

A lógica do algoritmo foi implementada na plataforma Arduino IDE, de forma similar ao Cenário 2. Quando é detectado um nível de brilho igual ou inferior a 0,45 lux, o sistema interpreta a ocorrência da tela preta e interrompe o teste. Caso contrário, o algoritmo continua executando os ciclos até atingir um número predeterminado, acionando o braço robótico sempre que a central inicia seu processo de desligamento, o qual exibe uma mensagem na tela solicitando a confirmação do usuário. A Figura 32 ilustra a bancada de testes montada para execução deste cenário.

Figura 32: Bancada de testes com integração do braço robótico.



Fonte: Elaborado pelo autor (2025).

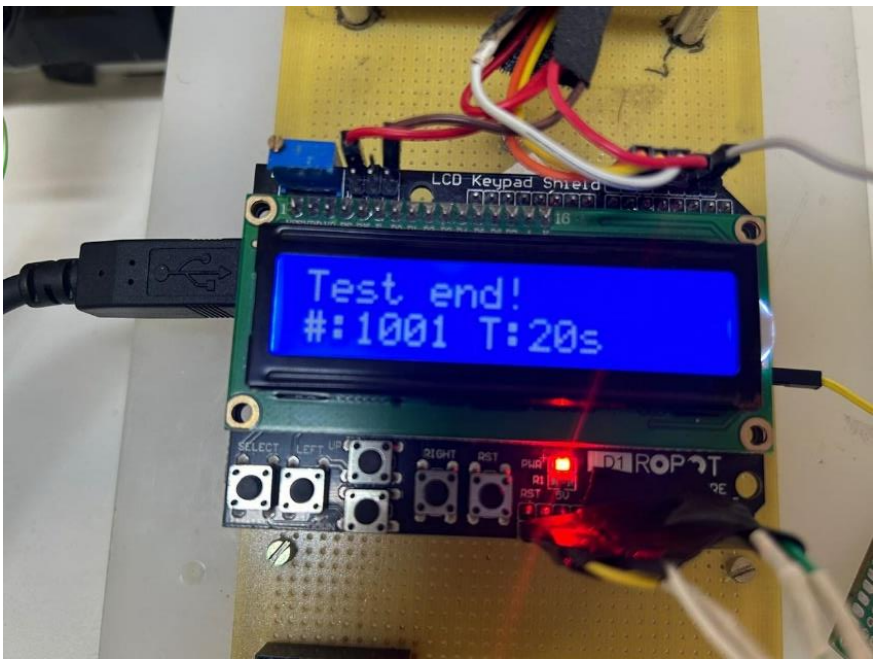
A Figura 33 apresenta a imagem exibida no display LCD da bancada após a execução de 101 ciclos consecutivos, sem ocorrência da falha tela preta. Esse resultado evidencia a estabilidade do sistema durante os testes automatizados, reforçando a confiabilidade do processo de validação.

Figura 33: Display LCD indicando conclusão de 101 ciclos sem ocorrência da falha tela preta.

Fonte: Elaborado pelo autor (2025).

A Figura 34 apresenta o display da bancada após a execução de 1.001 ciclos consecutivos, sem detecção da falha de tela preta. Esse resultado reforça a robustez do sistema e a eficácia da automação na realização de testes prolongados com alta repetibilidade.

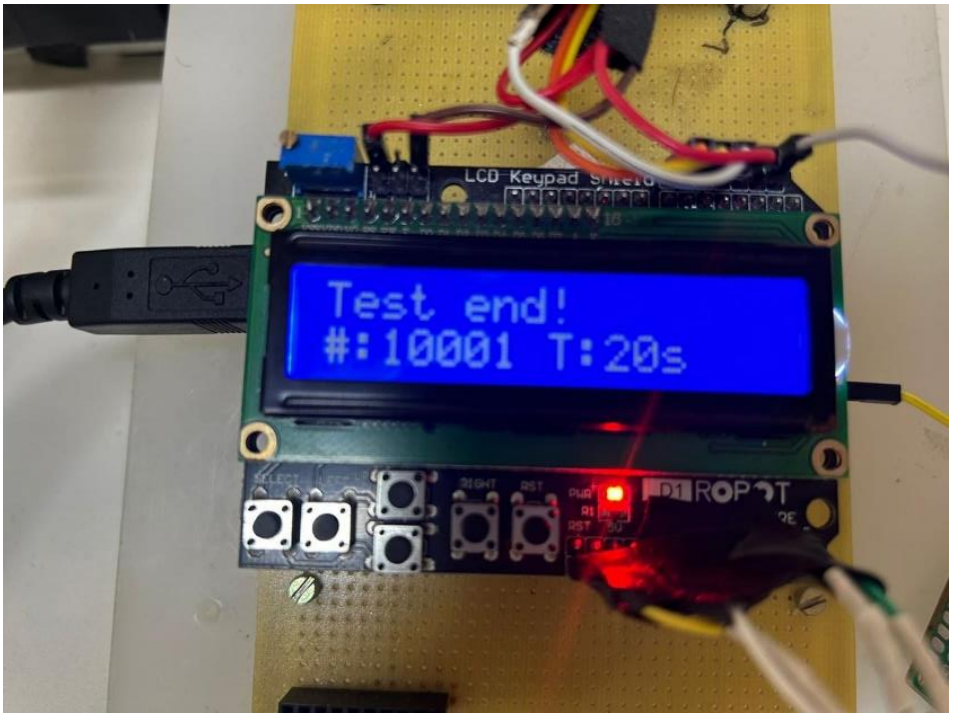
Figura 34: Display indicando conclusão de 1.001 ciclos sem ocorrência da falha tela preta.



Fonte: Elaborado pelo autor (2025).

A Figura 35 apresenta o display da bancada após a execução de 10.001 ciclos consecutivos, todos concluídos com sucesso e sem ocorrência da falha tela preta. Esse resultado demonstra a alta confiabilidade do sistema e a eficiência da automação para testes de longa duração, garantindo repetibilidade e robustez no processo de validação.

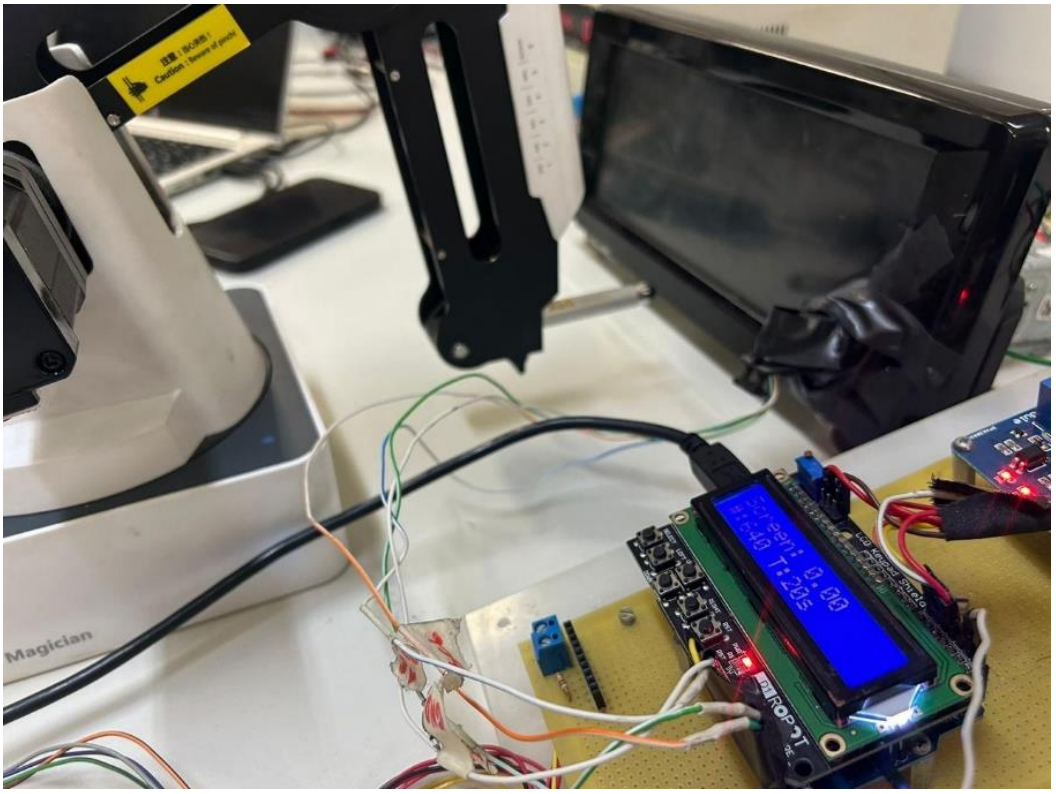
Figura 35: Display indicando conclusão de 10.001 ciclos sem ocorrência da falha tela preta.



Fonte: (Elaborado pelo autor, 2025).

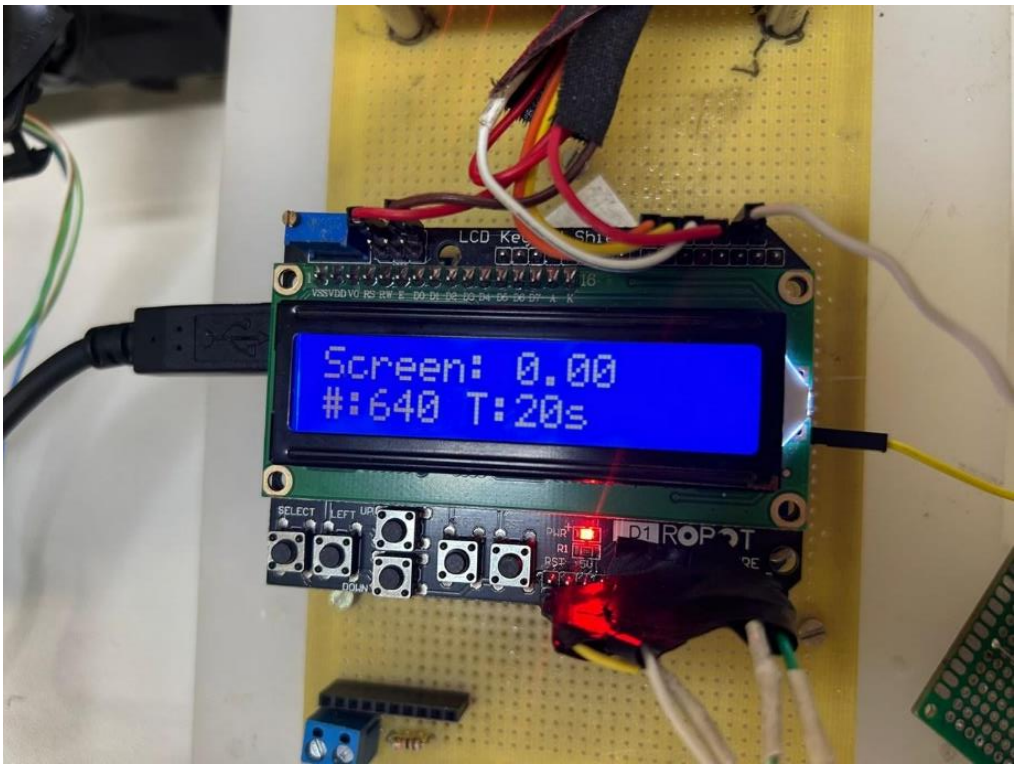
As Figuras 36 e 37 ilustram a interrupção automática do teste no ciclo 640, após a detecção da falha denominada tela preta. Esse evento confirma a eficácia do algoritmo na identificação precisa da condição de falha, garantindo que o processo seja interrompido imediatamente para permitir a coleta de dados adicionais e análise detalhada. A resposta rápida do sistema demonstra sua robustez e confiabilidade, assegurando que falhas críticas não passem despercebidas durante a validação automatizada.

Figura 36: Teste interrompido devido a tela preta após 640 ciclos.



Fonte: Elaborado pelo autor (2025).

Figura 37: Imagem do Shield Display indicando o momento falha.



Fonte: Elaborado pelo autor (2025).

Por fim, os testes realizados demonstraram que a automação proposta no Cenário 3 foi eficaz na identificação da falha e na execução de ações físicas automatizadas com alta precisão. A integração entre o Arduino e o Dobot Magician permitiu ampliar o nível de automação do processo de validação, reduzindo a necessidade de intervenção humana e aumentando a confiabilidade dos testes realizados em bancada.

5 CONCLUSÃO E TRABALHOS FUTUROS

O presente estudo teve como propósito central desenvolver soluções automatizadas voltadas à identificação de falhas caracterizadas como tela preta no processo de validação de *Head Units*. Para isso, foi estruturada uma bancada de testes composta por sensores, módulos de relé, plataforma Arduino e integração com ferramentas de programação como Python e CAPL. A partir dessa infraestrutura, foram concebidos, implementados e avaliados três cenários distintos de teste, os quais diferem entre si quanto ao nível de automação e à complexidade.

No Cenário 1, foi demonstrada uma abordagem simplificada baseada em lógica programada em CAPL, com integração ao ambiente CANalyzer, permitindo simular a interação do usuário ao engatar a marcha ré e validar se a imagem esperada é exibida na tela. Nesse cenário, foi possível realizar os testes de maneira continua, sem intervenção humana. A cada falha identificada, os dados eram armazenados para posterior análise. O algoritmo se mostrou robusto para executar o teste de maneira ininterrupta e eficiente para detecção da falha quando ela ocorria.

Já o Cenário 2 evoluiu para um sistema totalmente autônomo, utilizando o Arduino Uno, sensores de luminosidade e um *display* LCD, criando um ciclo automático de validação que detecta eventos de falha durante a inicialização do sistema. O teste poderia ser feito de forma contínua, foi possível deixá-lo rodando por mais de 24 horas e mais de 7.000 ciclos realizados. Para fins de comparação, se o teste rodasse continuamente por uma semana de 5 dias, mais de 35.000 ciclos poderiam ser testados caso não houve falha. Do mesmo modo, o algoritmo se mostrou robusto o suficiente para interromper o teste em condição de falha, permitindo uma análise posterior por parte do validador.

Por fim, o Cenário 3 agregou ainda mais robustez ao processo, com a inclusão do braço robótico Dobot Magician, capaz de executar ações físicas na interface da *Head Unit*, eliminando a necessidade de interação humana para desligar a central

nesse caso. Diferentemente do Cenário 2, onde a *Head Unit* desligava automaticamente, neste cenário havia uma mensagem que precisava ser confirmada pelo validador para realizar o desligamento completo da central. Com essa automação, foi possível executar todo esse procedimento sem intervenção humana.

Além disso, o teste foi parametrizado para finalizar após um número definido de ciclos. No teste mais longo, foram realizados 10.001 ciclos sem ocorrência de falhas. A confiabilidade e robustez do algoritmo também foram verificadas em condição de falha, na qual o sistema interrompeu o ciclo correspondente e aguardou a coleta de informações pelo validador para análise posterior.

Por fim, o uso do braço robótico no Cenário 3 demonstrou ser uma alternativa viável para a automação de ações físicas que não podem ser simuladas eletronicamente, como o toque em botões virtuais exibidos na interface da central.

Os resultados obtidos evidenciam que as soluções desenvolvidas foram eficazes em detectar falhas visuais de forma precisa e reprodutível, além de trazerem ganhos significativos de produtividade, confiabilidade e rastreabilidade ao processo de validação. A utilização de sensores para mensurar o brilho da tela, também foi uma excelente alternativa para identificar o fenômeno de tela preta permitindo criar uma arquitetura de testes escalável e adaptável e robusta.

Como proposta para trabalhos futuros, sugere-se a expansão da arquitetura desenvolvida para contemplar novos tipos de falhas funcionais, como congelamento de tela, atrasos excessivos na inicialização ou falhas na ativação da câmera de ré, bem como a adaptação desses testes para outras ECUs, como o *Cluster* e os *Displays*. Além disso, pode-se explorar a implementação de visão computacional para análise da imagem exibida na tela, substituindo os sensores de luminosidade por câmeras associadas a algoritmos de reconhecimento de padrões utilizando OpenCV.

Posto isso, conclui-se que o trabalho atingiu o objetivo esperado, propondo testes automatizados no ambiente de validação automotivo, garantindo maior robustez, confiabilidade e rastreabilidade ao processo, além de promover avanços significativos na área de validação automotiva e abrir caminho para futuras evoluções.

REFERÊNCIAS BIBLIOGRÁFICAS

AUTOESPORTE. *Peugeot 208: veja os problemas mais comuns do hatch compacto.* Disponível em: https://autoesporte.globo.com/servicos/noticia/2024/07/peugeot-208-veja-os-problemas-mais-comuns-do-hatch-compacto.ghtml. Acesso em: 15 ago. 2024.

AUTOMOTIVE SPICE. *Automotive SPICE Process Assessment Model.* Version 3.1. [S.l.]: intacs™, 2017. Disponível em: https://www.intacs.info. Acesso em: 10 jul. 2025.

BANDUR, Victor et al. Making the case for centralized automotive E/E architectures. *IEEE Transactions on Vehicular Technology*, v. 70, n. 2, p. 1230-1245, 2021.

BANZI, Massimo; SHILOH, Michael. *Primeiros passos com o Arduino.* São Paulo: Novatec, 2011.

BITTENCOURT, S. *O que é Arduino: Tudo o que você precisa saber.* HostGator, 31 jan. 2017. Disponível em: https://www.hostgator.com.br/blog/o-que-e-arduino/. Acesso em: 9 ago. 2024.

BOSCH. *Automated Testing of Infotainment and HMI Systems.* [S.l.]: Bosch Engineering Services, 2022. Disponível em: https://www.boschengineering.com. Acesso em: 10 jul. 2025.

CHEN, X.; LI, D.; ZHANG, Y. Application of Dobot robot in automated software testing for touch-screen embedded systems. In: *2022 IEEE International Conference on Mechatronics and Automation (ICMA).* IEEE, 2022. DOI: 10.1109/ICMA54875.2022.9859990.

CHO, Y.; KIM, J. Vision-based automated test framework for automotive infotainment systems using low-cost components. In: *IEEE International Conference on Consumer Electronics (ICCE).* IEEE, 2023. DOI: 10.1109/ICCE53466.2023.10025879.

DA SILVA, J. et al. Automated testing framework for infotainment validation using OpenCV and Arduino. In: *IEEE ETFA 2020 – 25th Conference on Emerging Technologies and Factory Automation.* IEEE, 2020. DOI: 10.1109/ETFA46521.2020.9212078.

DOBOT. *Dobot Magician.* Disponível em: https://www.dobot.ca/product/dobot-magician/. Acesso em: 15 ago. 2024.

EBERT, C.; FAVARO, J. Automotive Software. *IEEE Software*, v. 34, n. 3, p. 33-39, 2017.

ELETROGATE. *Identificando movimentos e cores com o Arduino.* Disponível em: https://blog.eletrogate.com/identificando-movimentos-e-cores-com-o-arduino/. Acesso em: 15 ago. 2024.

HOSTINGER. *Python: O que é, para que serve e como começar a programar.* Disponível em: https://www.hostinger.com.br/tutoriais/python-o-que-e. Acesso em: 15 ago. 2024.

HUANG, Yingping et al. Development of an automated testing system for vehicle infotainment system. *The International Journal of Advanced Manufacturing Technology*, v. 51, p. 233–246, 2010.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO 26262: Road vehicles – Functional safety.* Genebra: ISO, 2018.

ITSEEZ. *OpenCV: Open Source Computer Vision Library.* [S.l.], 2015. Disponível em: https://opencv.org. Acesso em: 8 jul. 2025.

J.D. POWER. *2023 U.S. Vehicle Dependability Study.* Costa Mesa: J.D. Power, 2023. Disponível em: https://www.jdpower.com/business/press-releases/2023-us-vehicle-dependability-study. Acesso em: 10 jul. 2025.

JETBRAINS. *PyCharm: Python IDE for Professional Developers.* Disponível em: https://www.jetbrains.com/pt-br/pycharm/. Acesso em: 15 ago. 2024.

LOWE, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, v. 60, n. 2, p. 91–110, 2004.

LUNETTA, Danilo. *Robotic platform for testing the Infotainment system of various vehicles.* 2021. Dissertação (Mestrado em Engenharia Automotiva) – Politecnico di Torino, Torino, 2021.

MAHFOUDHI, I.; CHEIKHROUHOU, N.; KARMANI, M. A review on automotive software testing: challenges and practices. *Journal of Manufacturing Systems*, v. 60, p. 343–358, 2021. DOI: https://doi.org/10.1016/j.jmsy.2021.02.002.

MAKERHERO. *Como utilizar o Display LCD Shield com teclado para Arduino.* Disponível em: https://www.makerhero.com/blog/como-utilizar-o-display-lcd-shield-com-teclado-para-arduino/. Acesso em: 15 ago. 2024.

MENEZES, Nilo Ney Coutinho. *Introdução à programação com Python.* São Paulo: Novatec, 2010.

MOBIAUTO. *Peugeot 208: os principais problemas segundo os donos.* Disponível em: https://www.mobiauto.com.br/revista/peugeot-208-os-principais-problemas-segundo-os-donos/2341. Acesso em: 15 ago. 2024.

MODY, Mihir et al. Understanding vehicle E/E architecture topologies for automated driving: System partitioning and tradeoff parameters. *Electronic Imaging*, v. 30, p. 1-5, 2018.

MUJA, M.; LOWE, D. G. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In: *International Conference on Computer Vision Theory and Applications (VISAPP).* Portugal: INSTICC, 2009.

NICOLAESCU, Sergiu Stefan et al. A new project management approach for R&D software projects in the automotive industry – continuous V-model. *International Journal of Web Engineering and Technology*, v. 12, n. 2, p. 120-142, 2017.

PEUGEOT DO BRASIL. *Câmbio Peugeot 208 danificado.* Reclame Aqui. Disponível em: https://www.reclameaqui.com.br/peugeot-do-brasil/cambio-peugeot-208-danificado_9vq7RuVP6hpyKj9E/. Acesso em: 15 ago. 2024.

PRETSCHNER, A. et al. Model-based testing for software-intensive systems: The V-Model in the era of DevOps. In: *2020 IEEE International Conference on Software Testing.* IEEE, 2020.

SINI, Jacopo et al. An automatic approach to integration testing for critical automotive software. In: *2018 13th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS).* IEEE, 2018.

SOLIMENE, Raffaele. *Using image recognition to automate the testing process of instrument panels on commercial vehicles.* 2020. Dissertação (Mestrado em Engenharia Automotiva) – Politecnico di Torino, Torino, 2020.

STA ELETRÔNICA. *Como utilizar o sensor de gestos APDS-9960 com Arduino.* Disponível em: https://www.sta-eletronica.com.br/artigos/arduinos/como-utilizar-o-sensor-de-gestos-apds-9960-com-arduino. Acesso em: 15 ago. 2024.

TIERNO, Antonio et al. Open issues for the automotive software testing. In: *2016 12th IEEE International Conference on Industry Applications (INDUSCON).* IEEE, 2016.

TSAI, Pu-Sheng et al. Drawing System with Dobot Magician Manipulator Based on Image Processing. *Machines*, v. 9, n. 12, p. 302, 2021.

VECTOR. *Introdução à programação em CAPL.* Disponível em: https://www.vector.com/br/pt/eventos/vbr-vector-brazil/2020/webinar-recordings/introducao-a-programacao-em-capl/. Acesso em: 15 ago. 2024.

VECTOR. *VN16xx Interface.* Disponível em: https://www.vector.com/br/pt/produtos/products-a-z/hardware/network-interfaces/vn16xx/#c348249. Acesso em: 15 ago. 2024.

WIKIMEDIA COMMONS. *Arduino-uno-perspective-transparent.* Disponível em: https://commons.wikimedia.org/wiki/File:Arduino-uno-perspective-transparent.png. Acesso em: 15 ago. 2024.